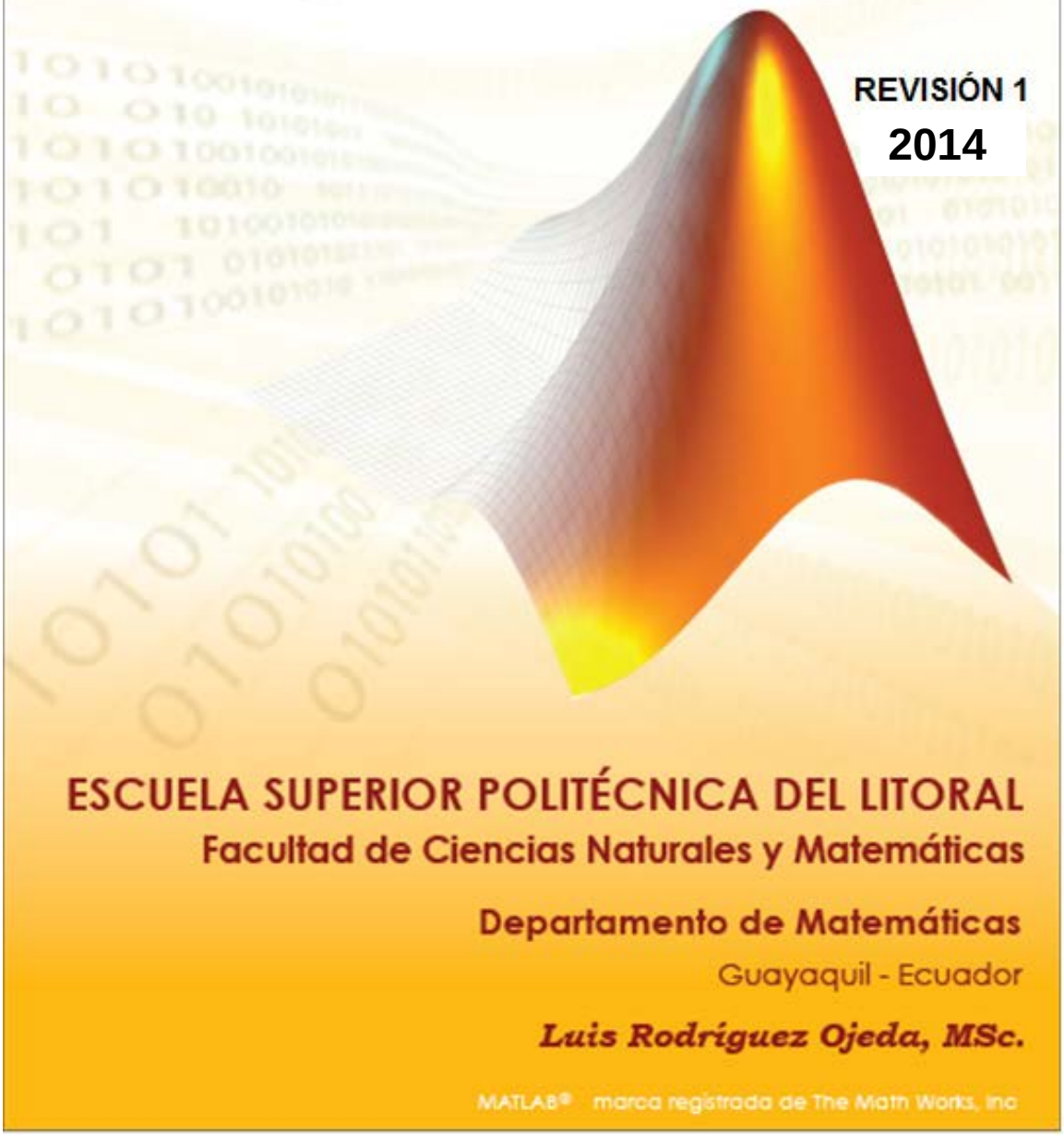
MATLAB
Programación



REVISIÓN 1

2014

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL
Facultad de Ciencias Naturales y Matemáticas

Departamento de Matemáticas

Guayaquil - Ecuador

Luis Rodríguez Ojeda, MSc.

MATLAB PROGRAMACIÓN

Prefacio

Este documento es una contribución bibliográfica para los estudiantes que toman un primer curso de Programación de Computadoras a nivel universitario con el soporte de MATLAB. El contenido de esta obra no tiene pre-requistos, solamente el interés en conocer un lenguaje actual que sea simple y versátil para resolver computacionalmente problemas de diferente nivel de complejidad. El enfoque didáctico utilizado en este documento es el aprendizaje mediante ejemplos.

El origen es la experiencia desarrollada por el autor impartiendo estos cursos en el Departamento de Matemáticas de la ESPOL para estudiantes de ingeniería. Contiene el material desarrollado en las clases basado principalmente en el uso de ejemplos típicos para describir los conceptos algorítmicos en forma práctica.

El énfasis es el componente algorítmico en la solución de problemas con el apoyo del programa MATLAB que contiene un amplio repertorio de funciones para aplicaciones numéricas, gráficas y simbólicas y dispone además de un lenguaje de programación muy simple y general. Combinando estos componentes se tiene un instrumento computacional muy potente y efectivo para resolver problemas en áreas muy diversas en ingeniería, matemáticas y otras ciencias. El único inconveniente con el uso de este programa computacional es que todavía es un software con licencia, aunque ya se están desarrollando versiones similares de uso libre y público-

En este documento se describe el componente programable de MATLAB el cual requiere usar instrucciones cuya sintaxis es simple pero adecuada y suficiente para resolver problemas más complejos. El componente interactivo de MATLAB se describe en un anexo a este documento. El componente de programación es obligatorio para los estudiantes que desean desarrollar su capacidad lógica para enfrentar la solución de problemas para cuya solución no es suficiente el componente interactivo de MATLAB.

Otro objetivo importante al escribir estos documentos es el desarrollo de textos digitales para ser usados en línea, reduciendo el consumo de papel y tinta, contribuyendo así con el cuidado del medio ambiente. Una ventaja adicional de los libros virtuales es la facilidad para actualizar y mejorar continuamente su contenido.

Este documento ha sido compilado en formato pdf. Este formato permite controlar el tamaño del texto en la pantalla y agregar un índice electrónico para facilitar la búsqueda de temas y, dependiendo de la versión del programa de lectura de este formato, se pueden usar las facilidades disponibles para resaltar digitalmente texto, insertar comentarios, notas, enlaces, revisiones, búsqueda por contenido, lectura, etc.

Luis Rodríguez Ojeda, M.Sc.
lrodrig@espol.edu.ec

Contenido

MATLAB Programación

1 Introducción

1.1 Objetivo y requisitos

Esta obra es una contribución para el desarrollo de una metodología computacional para resolver problemas basada en los principios de la construcción de algoritmos.

El soporte computacional es el producto MATLAB con el que se explora y se adquiere la práctica y el conocimiento suficiente para la programación de computadoras aplicada a la resolución de problemas matemáticos, de ingeniería y otras ciencias. Se supondrá que los interesados tienen al menos algun conocimiento elemental de la lógica matemática.

1.2 Metodología

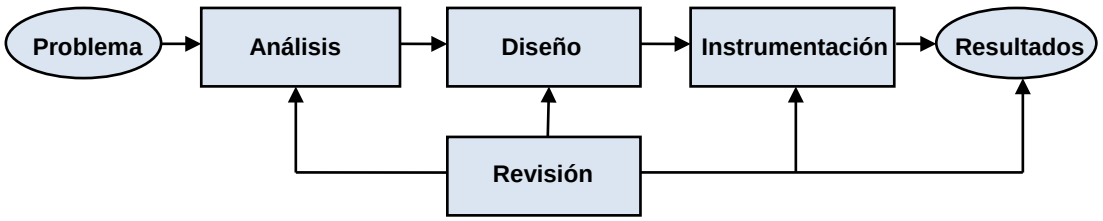
Mediante explicaciones basadas en ejemplos, el usuario puede adquirir en forma progresiva los conocimientos necesarios para resolver problemas y su programación en MATLAB. El desarrollo de variados ejercicios proporcionará la base para extenderlos y aplicarlos a la resolución de problemas más complejos. Al mismo tiempo, el usuario podrá desarrollar su propio estilo de programación.

1.3 Un modelo para resolver problemas con el computador

El análisis y diseño de soluciones computacionales es una ciencia que facilita el uso eficiente del poder de las computadoras para resolver problemas.

Para facilitar el desarrollo de estas soluciones, es adecuado usar un lenguaje computacional simple, general y eficiente como el que proporciona MATLAB.

El siguiente gráfico describe los pasos en la solución computacional de un problema



Primero, es necesario asegurarnos que el problema que intentamos resolver está en nuestro ámbito de conocimiento. No es recomendable intentar resolver problemas si no tenemos el conocimiento y la práctica para enfrentar su solución.

En la etapa de **Análisis** se estudia el problema en forma detallada: sus características, las variables y los procesos que intervienen. Asimismo, se deben definir los datos que se requieren y cual es el objetivo esperado. El resultado de esta etapa son las **especificaciones** detalladas de los requerimientos que en algunos casos se pueden expresar en forma matemática.

En la etapa de **Diseño** se procede a elaborar los procedimientos necesarios para cumplir con los requerimientos especificados en el análisis, incluyendo fórmulas, tablas, etc. El objeto resultante se denomina **algoritmo**.

En la etapa de **Instrumentación,** si el problema es simple, se puede obtener la solución interactuando directamente mediante instrumentos disponibles en el entorno computacional. Si el problema es más complejo, deben construirse **programas** y definir el ingreso y la organización de los datos necesarios.

Al concluir la etapa de la instrumentación, se usan datos para realizar pruebas de los programas. Es necesario que se realice una revisión en cada etapa de este proceso y que se validen los resultados obtenidos antes de aceptarlos.

Posteriormente se efectúa la instalación y operación. Debe preverse la necesidad de mantenimiento y cambios en los programas para ajustarlos al entorno en el que se usarán.

Este proceso necesita ser planificado y sistematizado para que su desarrollo sea eficiente siendo imprescindible seguir normas, utilizar metodologías de programación y mantener una documentación adecuada.

Los instrumentos computacionales modernos tales como MATLAB disponen de facilidades para probar interactivamente instrucciones y programas a medida que son desarrollados. También ofrece librerías que facilitan el desarrollo de proyectos de programación y mejoran la productividad.

2 Algoritmos

Un algoritmo es una descripción ordenada de las instrucciones que deben realizarse para resolver un problema en un tiempo finito.

Para crear un algoritmo es necesario conocer en forma detallada el problema, las variables, los datos que se necesitan, los procesos involucrados, las restricciones, y los resultados esperados.

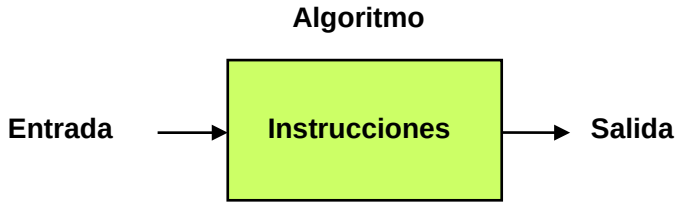
La descripción del algoritmo debe orientarse a la instrumentación computacional final. pero cuando los problemas son simples, puede omitirse la elaboración detallada del algoritmo e ir directamente a la codificación en el lenguaje computacional.

Es muy importante desarrollar el pensamiento algorítmico progresivamente con la ayuda de ejemplos y con la práctica. El objetivo final es estructurar una metodología para facilitar la resolución de problemas, investigando sus componentes y sus requerimientos.

2.1 Estructura de un algoritmo

Un algoritmo es un objeto que debe comunicarse con el entorno. Por lo tanto debe incluir facilidades para el ingreso de datos y la salida de resultados.

Dentro de un algoritmo se describe un procedimiento para realizar la transformación que produce los resultados esperados.



2.2 Lenguajes para escribir algoritmos

Para escribir algoritmos se pueden usar diferentes notaciones: lenguaje natural, lenguajes gráficos, lenguajes simbólicos, etc.

Para que una notación sea útil debe poseer algunas características que permitan producir algoritmos fáciles de construir, entender y aplicar:

1) Las instrucciones deben ser simples para facilitar su uso.
2) Las instrucciones deben ser claras y precisas para evitar ambigüedades.
3) Debe incluir suficientes instrucciones para describir la solución de problemas.
4) Preferentemente, las instrucciones deben tener orientación computacional.

Los algoritmos deben ser reproducibles, es decir que al ejecutarse deben entregar los mismos resultados si se utilizan los mismos datos.

2.3 Definiciones

a) Proceso

Conjunto de acciones realizadas al ejecutar las instrucciones descritas en un algoritmo.

b) Estado

Situación de un proceso en cada etapa de su realización, desde su inicio hasta su finalización. En cada etapa, las variables pueden modificarse.

c) Variables

Símbolos con los que se representan los valores que se producen en el proceso.

Componentes de una variable:

Nombre:	Identificación de cada variable
Tipo:	Conjunto de valores o dominio asociado a la variable
Contenido:	Valor asignado a una variable
Celda:	Dispositivo que almacena el valor asignado a una variable

2.4 Introducción a la construcción de algoritmos

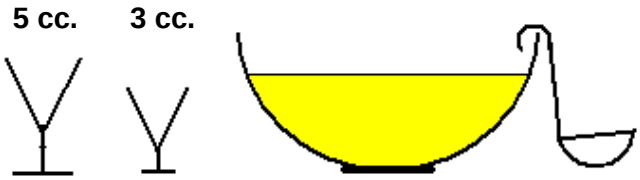
El desarrollo de algoritmos constituye una metodología para resolver problemas en forma organizada.

Primero debe definirse el objetivo al que se desea llegar, luego deben identificarse los componentes o variables y escribir las instrucciones necesarias para poder realizar los cambios que permitirán llegar al objetivo propuesto. Para validar el algoritmo debe realizarse alguna prueba con la cual se puede verificar que el resultado es correcto.

A continuación se propone un problema y se describe un procedimiento algorítmico para llegar a la solución.

Ejemplo. En el gráfico siguiente se muestra un recipiente grande con algún refresco y se propone el problema de obtener exactamente **4 cc.** usando solamente los instrumentos mostrados los cuales tienen indicada su capacidad. Se supondrá que el recipiente grande contiene suficiente cantidad de refresco. Es posible trasladar el contenido entre recipientes pero no se dispone de ningún dispositivo adicional para medición.

Describa un algoritmo para llegar a la solución.



Objetivo propuesto: Que el recipiente de **5 cc.** contenga **4 cc.** del refresco

Variables

Sean **A**: Representación del recipiente cuya capacidad es **5 cc.**
B: Representación del recipiente cuya capacidad es **3 cc.**
C: Representación del recipiente grande con cantidad suficiente de refresco.

Algoritmo

1. Llene **A** con el refresco de **C**
2. Vierta **A** en **B** hasta llenarlo
3. Vierta todo el contenido de **B** en **C**
4. Vierta el resto del contenido de **A** en **B**
5. Llene **A** con el refresco de **C**
6. Vierta el contenido de **A** en **B** hasta llenarlo
7. El recipiente **A** contendrá **4 cc**.

Prueba

Recorrer el algoritmo anotando los valores que toman las variables **A** y **B**

Instrucción	Contenido de A	Contenido de B
Inicio	**0**	**0**
1	**5**	**0**
2	**2**	**3**
3	**2**	**0**
4	**0**	**2**
5	**5**	**2**
6	**4**	**3**

Resultado

Se verifica que en el recipiente **A** quedarán **4 cc.**

Observe los componentes que intervienen en la construcción del algoritmo:

- **a)** Propuesta del objetivo
- **b)** Definición de variables
- **c)** Lista de instrucciones
- **d)** Prueba del algoritmo
- **e)** Verificación del resultado obtenido

Ejemplo. Describa un algoritmo para revisar un vehículo antes de un viaje.

Algoritmo

1 **Si el nivel de agua del radiador está bajo**
   Complete el nivel de agua del radiador
2 **Si el nivel de gasolina es bajo**
   Acuda a la estación de gasolina y llene el tanque
3 **Si el nivel de aceite del motor es bajo**
   Acuda a la estación de servicio para chequear el vehículo
4 **Para cada llanta repita la siguiente instrucción**
   Compruebe la presión del aire
5 **Si alguna llanta registró presión baja**
   Acuda a la estación de servicio para revisión de llantas

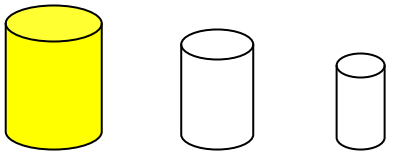
Este algoritmo pretende ser un algoritmo para la revisión del vehículo. Contiene **acciones condicionadas** y también una instrucción para **repetir una acción** varias veces. Sin embargo, el uso del lenguaje común no permite que la descripción sea suficientemente precisa para facilitar el seguimiento de las instrucciones. Tampoco se puede constatar que se cumple el objetivo propuesto como en el ejemplo anterior. Por lo tanto, se lo puede considerar simplemente como un isntructivo de ayuda.

Los lenguajes algorítmicos deben ser descripciones claras, de tal manera que no haya posibilidad de interpretar las instrucciones de más de una manera.

2.5 Ejercicios de creación de algoritmos

Para cada ejercicio proponga un algoritmo para obtener la solución.

1. Se tienen 3 recipientes cilíndricos, opacos y sin marcas, de 12, 7, y 5 galones de capacidad. El recipiente de 12 galones está lleno de combustible. El objetivo es repartir el combustible en dos partes iguales usando únicamente los tres recipientes. Considere que puede trasladar el combustible entre recipientes pero no se dispone de algún instrumento de medición.



a) Describa gráficamente el resultado esperado
b) Asigne símbolos a las variables (Representan la cantidad de combustible)
c) Construya un algoritmo para obtener la solución. Numere las instrucciones
d) Ejecute las instrucciones y registre los cambios del contenido de las variables
e) Verifique que el algoritmo produce la solución esperada.

Para probar su algoritmo puede completar una tabla como la siguiente. Suponga que A, B, C representan a los recipientes con la capacidad y en el orden dados en el gráfico anterior.

Instrucción	**A**	**B**	**C**
Inicio	**12**	**0**	**0**
1			
2			
. . .			

Nota: Existe una solución en 12 pasos (en cada paso se traslada de un recipiente a otro).

2. Describa un algoritmo para resolver el siguiente conocido problema. Defina las variables, escriba y numere las instrucciones y luego efectúe una prueba para verificar que funciona**:**

Tres misioneros y tres caníbales deben atravesar un río en un bote en el que sólo caben dos personas. Pueden hacer los viajes que quieran, pero en en las orillas y en el bote el número de caníbales no debe ser mayor al de los misioneros porque ya podemos suponer lo que ocurriría. El bote no puede cruzar el río si no hay al menos una persona dentro para que lo dirija.

Sugerencia: Defina los misioneros como **M1, M2, M3** y los caníbales como **C1, C2, C3.** Las variables **R1, R2** son las orillas del río y **B** el bote. El contenido de estas variables cambiará mediante las instrucciones del algoritmo. Después de construir el algoritmo puede completar una tabla como la siguiente para verificar el resultado:

Instrucción	R1	B	R2
Inicio	**M1,M2,M3,C1,C2,C3**		
1			
2			
. . .			
Final			**M1,M2,M3,C1,C2,C3**

3. Describa un algoritmo para resolver el siguiente problema, también muy conocido. Defina las variables, escriba y numere las instrucciones y luego efectúe una prueba para verificar que funciona**:**

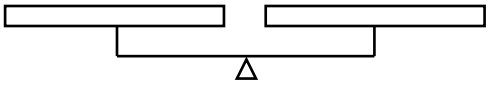
Había un pastor que cuidaba a un lobo, una oveja y una canasta de lechugas. El pastor tenía que cruzar un río, para lo cual disponía de un pequeño bote en el que solamente cabían él y un animal, o él y la canasta de lechugas. El problema es conseguir que pasen todos al otro lado del río sanos y salvos, sin que nadie se coma a nadie. Al lobo no le gustan las lechugas, pero como se puede suponer, el lobo no puede quedarse a solas con la oveja y tampoco la oveja puede quedarse sola con las lechugas. El pastor debe guiar al bote en cada viaje.

Sugerencia: Defina símbolos para los datos **P**: pastor, **L**: lobo, **O**: oveja, **C**: canasta**.** Las variables **R1, R2** son las orillas del río y **B** el bote. El contenido de estas variables cambiará mediante las instrucciones del algoritmo. Después de construir el algoritmo puede completar una tabla como la siguiente para verificar el resultado:

Instrucción	R1	B	R2
Inicio	**P, L, O, C**		
1			
2			
. . .			
Final			**P, L, O, C**

4. Describa un algoritmo para resolver el siguiente problema. Defina las variables, escriba y numere las instrucciones y luego efectúe una prueba para verificar que funciona**:**

Se tiene una caja con nueve bolas, semejantes en apariencia, entre las cuales hay una más pesada que las otras ocho. No se sabe cuál es y se trata de hallarla efectuando solamente dos pesadas en una balanza de dos platillos en equilibrio.



Después de construir el algoritmo puede completar una tabla como la siguiente para verificar el resultado, en donde a, b, c, d, e, f, g, h, i representan a las nueve bolas.

Instrucción	Caja	Platillo izquierdo	Platillo derecho
Inicio	**a, b, c, d, e, f, g, h, i**		
1			
2			
. . .			
Final			

5. Describa en forma precisa las instrucciones necesarias para preparar una fiesta sorpresa para su amiga o su amigo. En las instrucciones debe incluir los días y horas en los que serán desarrolladas las actividades. Haga referencia a la fecha y hora cero en la que ocurrirá el evento. Verifique su algoritmo mediante un cuadro con fechas y horas. En este cuadro anote el desarrollo de las actividades siguiendo las instrucciones de su algoritmo. Note que este tipo de algoritmos no se puede verificar que cumplen el objetivo propuesto como en los ejercicios anteriores. Pueden considerarse únicamente como instructivos para organizar el desarrollo de actividades.

3. Notación para construir algoritmos computacionales

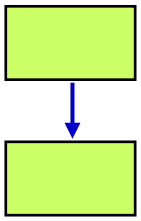
En esta sección se describirá una notación para construir algoritmos computacionales. La notación es suficientemente simple y clara para ser usada en problemas básicos, facilitando la construcción de su solución, contribuyendo además al desarrollo del pensamiento algorítmico y la lógica de programación de computadoras. Esta notación es útil especialmente en la etapa inicial de aprendizaje de la programación. Posteriormente se puede prescindir de ella.

Un algoritmo es la descripción ordenada de la idea que se propone para resolver un problema. Esta idea debe desarrollarse identificando los componentes que permitirán llegar a solución. Cada componente puede incluir una instrucción simple o un conjunto de instrucciones.

Para desarrollar una metodología representaremos cada componente gráficamente mediante un bloque:

Bloque para representar componentes del algoritmo

Un algoritmo puede contener varios componentes que son ejecutados en forma secuencial. Este orden se lo puede indicar explícitamente mediante líneas de flujo que unen los bloques**:**

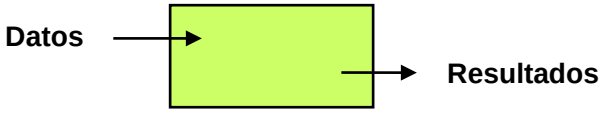


Bloque con el primer componente

Bloque con el segundo componente

3.1 Instrucciones u operaciones elementales

Los componentes de un algoritmo contienen instrucciones u operaciones con las que se especifican cálculos y otros procesos. Si el algoritmo debe comunicarse con el entorno entonces debe incluir instrucciones para la entrada de datos y la salida de los resultados.

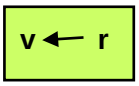


Las flechas que entran o salen del bloque representan la interacción del algoritmo con el exterior.

La siguiente es la simbología con la que se describirán las instrucciones u operaciones elementales para construir los componentes de un algoritmo.

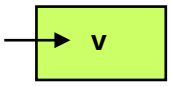
Sean **v** una variable y **r** algún valor que se desea usar en el algoritmo.

a) Instrucción de asignación



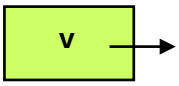
Asigna dentro del bloque el valor **r** a la variable **v**

b) Instrucción de entrada



Ingresa un valor desde afuera del bloque para la variable **v**

c) Instrucción de salida



Muestra fuera del bloque el valor que contiene la variable **v**

Es una buena práctica documentar la construcción del algoritmo. El algoritmo y cada variable utilizada deberían tener alguna descripción.

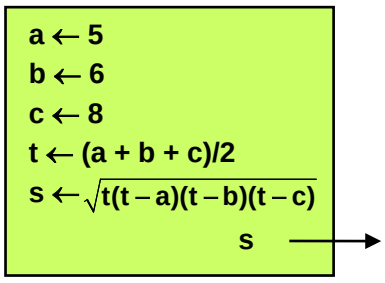
A continuación se desarrolla un ejemplo simple que se resuelve en varias etapas para ilustrar el uso de la notación algorítmica y su mejoramiento hasta llegar a una forma conocida.

Ejemplo. Describa un algoritmo para calcular el área de un triángulo conocidos sus tres lados, suponer que son **5**, **6**, y **8**.

Algoritmo: Área de un triángulo con sus lados conocidos

Variables

a, b, c: Lados del triángulo (Son datos conocidos, suponer que son **5**, **6**, y **8**)
s: Área del triángulo (Es el resultado esperado)
t: semiperímetro (Valor usado para la fórmula del área
$s = \sqrt{t(t-a)(t-b)(t-c)}$, (Fórmula del área del triángulo)
siendo **t = (a + b + c)/2**



En los algoritmos las instrucciones deben escribirse en el orden en el cual deben ejecutarse. En el ejemplo anterior, primero debe calcularse el valor de **t** antes de calcular y luego mostrar el valor de **s**

Prueba del algoritmo

Para probar un algoritmo, se recorren las instrucciones y se registran los valores que toman las variables

Prueba del algoritmo del ejemplo anterior

a	b	c	t	s
5	6	8	9.5	14.9812

El algoritmo producirá el resultado **14.9812** cuando sean ejecutadas las instrucciones.

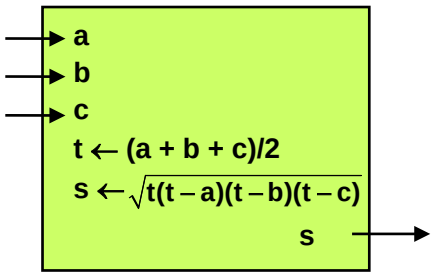
Observación importante: Es preferible que los datos para las variables sean ingresados al algoritmo desde fuera del bloque. De esta manera el algoritmo se independiza de los datos y no hay que cambiar las instrucciones dentro del bloque para realizar pruebas con nuevos datos. Modificamos el ejemplo anterior para desarrollar este concepto.

Ejemplo. Modifique el algoritmo anterior para que los **datos ingresen desde fuera del bloque** cuando el algoritmo sea probado o ejecutado.

Algoritmo: Área de un triángulo

Variables

a, b, c: Lados del triángulo (Datos desconocidos)
s: Área del triángulo (Es el resultado esperado)
t: semiperímetro (Valor usado para la fórmula del área)
$s = \sqrt{t(t-a)(t-b)(t-c)}$, (Fórmula del área del triángulo)
siendo $t = (a + b + c)/2$



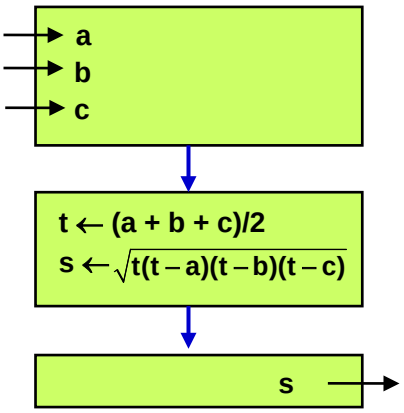
Los datos ya no están dentro del bloque. El resultado que producirá el algoritmo dependerá de los valores que entrarán para las variables en cada prueba o ejecución.

Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del bloque:

Prueba	a	b	c	t	s
1	5	6	8	9.5	14.9812
2	4	7	6	8.5	11.9765
3	8	6	9	11.5	23.5252

El algoritmo permanece invariante y produce los resultados esperados para los datos dados en cada prueba.

Para mejorar la claridad del algoritmo se puede separar el bloque en varios bloques. Esto permite identificar y agrupar las acciones que se realizan:

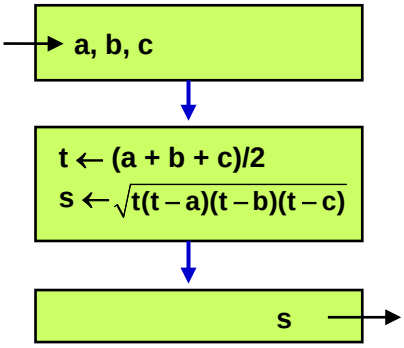


Bloque de entrada de datos

Bloque de cálculos

Bloque de salida de resultados

Se puede hacer una simplificación de la notación para la entrada o salida:



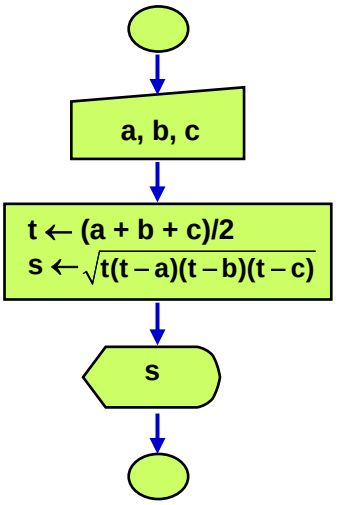
Bloque de entrada de datos

Bloque de cálculos

Bloque de salida de resultados

Si se usan símbolos especiales de diagramas de flujo, el resultado es un objeto muy conocido, un diagrama de flujo para describir algoritmos con orientación computacional. Esta notación será usada en varios ejemplos de este documento.

Diagrama de flujo



Inicio del algoritmo

Entrada manual de datos (por teclado)

Instrucciones de cálculos y asignación

Salida de resultados (por pantalla)

Final del algoritmo

Los bloques y las líneas de flujo de los diagramas ayudan a la comprensión de la lógica de los algoritmos, especialmente cuando se deben incluir instrucciones cuya ejecución está condicionada o cuando se necesita describir la ejecución repetida de bloques de instrucciones. Esta notación gráfica se usará para describir las instrucciones básicas del lenguaje computacional y en algunos ejemplos iniciales.

Finalmente, con la práctica y cuando se acepta y comprende la lógica algorítmica, se podrá prescindir del dibujo de bloques y líneas de flujo y escribir el algoritmo mediante algún **seudo lenguaje** o directamente con un **lenguaje de programación.**

Los seudo lenguajes no tienen reglas fijas para escribir instrucciones pero deberían tener la claridad suficiente para expresar el algoritmo en forma precisa, usualmente orientándolo a su futura instrumentación computacioal. La siguiente descripción en seudo lenguaje sería una versión aceptable para el ejemplo anterior:

Seudo lenguaje

Entrar a, b, c
t ← (a + b + c)/2
s ← $\sqrt{t(t-a)(t-b)(t-c)}$
Mostrar s

En el diseño y construcción de un algoritmo, lo más importante es el conocimiento del problema y la concepción de la idea para resolverlo.

La notación algorímica es solamente un instrumento para expresar de manera estructurada la idea propuesta con la que se espera llegar a la solución del problema.

3.1.1 Algunas instrucciones típicas de asignación en notación algorítmica

Los siguientes ejemplos se proponen para explicar la notación y algunos aspectos del uso de las instrucciones de asignación en los algoritmos.

a) Asigne a la variable **n** la raiz cuadrada de **5.**

n ← $\sqrt{5}$

b) Asigne a la variable **s** el valor **0**.

s ← 0

Normalmente se inician variables con cero para luego agregar valores. Es costumbre denominar a estas variables con el nombre de **"acumulador"** o **"sumador".**

Algunas variables también se inician con cero para luego incrementarlas con un valor unitario. Estas variables se usan para conteos y se las distingue con el nombre de **"contador".**

c) Modifique el valor actual de la variable **s** incrementándolo con el valor de **u**.

s ← s + u

Si las variables **u** o **s** no tuviesen asignadas algún valor previo, será un error.
En esta instrucción la misma variable **s** aparece a la izquierda y a la derecha.
La asignación modifica el valor de esta variable.
Es importante distinguir la **asignación algorítmica** de la **igualdad que se usa en el lenguaje matemático** en el cual sería incorrecto escribir: **s = s + u**

d) Modifique el valor actual de la variable **k** incrementándolo en **1**.

k ← k + 1

Si **k** no ha sido asignada previamente, será un error.

e) Modifique el valor de la variable **r** reduciendo su valor actual en **2**

r ← r – 2

f) Modifique el valor de la variable **n** duplicando su valor actual

n ← 2n

g) Modifique el valor de la variable **x** incrementando su valor actual en **20%**

x ← 1.2 x

h) Modifique el valor de la variable **t** reduciendo su valor actual en **5%**

t ← 0.95 t

j) Intercambie el contenido de las variables **a** y **b**

v ← a
a ← b
b ← v

Se requiere usar una variable adicional para no perder uno de los valores.

Es un error realizar la asignación de la siguiente manera**:**

a ← b
b ← a

Se perdería el valor que contenía la variable **a**

Cada variable puede contener un solo valor en cualquier momento de la ejecución del algoritmo. Este valor es el que ha sido asignado más recientemente.

Algunos ejemplos contienen la misma variable a la izquierda y a la derecha. La variable a la derecha contiene el valor actual. Con este valor se realiza alguna operación y el resultado es asignado a la variable que también aparece a la izquierda. Este último valor es el que conserva la variable al continuar el algoritmo.

Este tipo de asignación es usado frecuentemente en los algoritmos pues permite cambiar el contenido de las variables durante la ejecución.

3.1.2 Ejercicios con la notación algorítmica

Algoritmos con bloques secuenciales

Para cada ejercicio escriba una solución en notación algorítmica y realice una prueba

1. Dados el radio y altura de un cilindro calcule el área total y el volumen

2. Se tiene un recipiente cilíndrico con capacidad en litros. Su altura es un dato en metros. Determine el diámetro de la base

3. Dadas las tres dimensiones de un bloque rectangular calcule y muestre su área total y su volumen

4. La siguiente fórmula proporciona el enésimo término **u** de una progresión aritmética:

$$u = a + (n - 1)\, r$$

en donde **a** es el primer término, **n** es el la cantidad de términos y **r** es la razón entre dos términos consecutivos. Calcular el valor de **r** dados **u, a, n**

5. El examen de una materia es el 70% de la nota total. Las lecciones constituyen el 20% y las tareas el 10% de la nota total. Ingrese como datos la nota del examen calificado sobre 100 puntos, la nota de una lección calificada sobre 10 puntos, y las notas de tres tareas calificadas cada una sobre 10 puntos. Calcule la calificación total sobre 100 puntos.

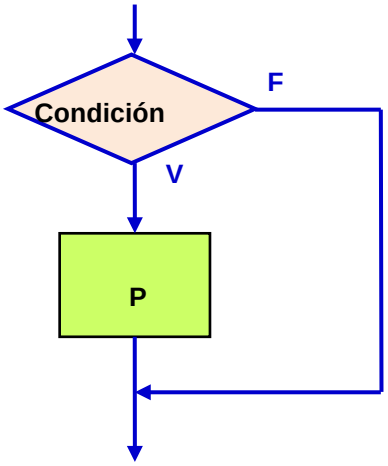
3.2 Estructuras de control de flujo de un algoritmo

La ejecución de los bloques de un algoritmo es secuencial de arriba hacia abajo, pero este orden puede alterarse mediante estructuras de control de flujo que permiten establecer un orden especial en la ejecución. Para describirlas se usará una representación gráfica.

3.2.1 Decisones

Describen la ejecución selectiva de bloques usando como criterio el resultado de una condición.

a) Ejecución condicionada de un bloque



Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutarán las instrucciones en el bloque **P** caso contrario, si el resultado es falso (**F**) el bloque no será ejecutarlo. En ambos casos el algoritmo continua debajo del bloque.

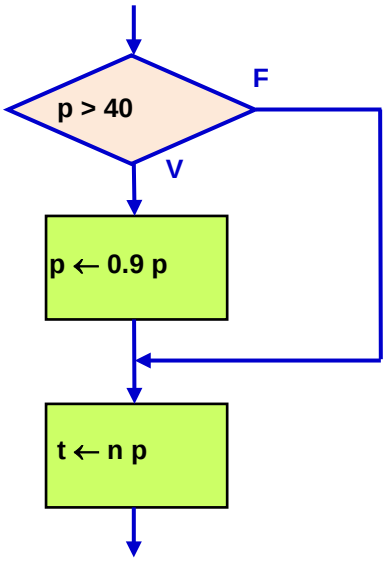
La condición es cualquier expresión cuyo resultado puede ser únicamente verdadero (**V**) o falso (**F**). Puede incluir operadores para comparar el contenido de variables y también se pueden usar los conectores de la lógica matemática.

Ejemplo. Expresiones que pudieran ser usadas como una condición

$n > 0$
$a \leq 5$
$x \neq 4$
$a < 3 \vee x > 1$

Para que una expresión pueda evaluarse y ser usada como una condición, las variables incluidas en la expresión deben tener asignado algún valor, caso contrario será un error pues la condición no podría evaluarse.

Ejemplo. Describa en notación algorítmica como reducir en **10%** el valor que contiene la variable **p**, en caso de que su valor actual sea mayor a **40**. Despues obtenga el resultado de la multiplicación de **n** por el valor de **p** (con su valor inicial o con su valor corregido).



Antes del bloque, la variable **p** debe haber sido asignada con algún valor, caso contrario sería un error.

Ejemplo. Describa en notación algorítmica una solución al siguiente problema.

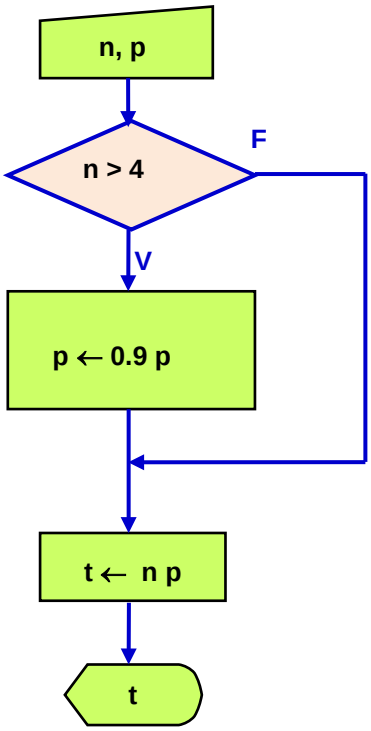
Calcular el valor total que una persona debe pagar por la compra de llantas en un almacén que tiene la siguiente promoción: Si la cantidad de llantas comprada es mayor a **4**, el precio unitario tiene un descuento de **10%.** El algoritmo debe ingresar como datos la catidad de llantas y el precio inicial de cada llanta. Mediante una comparación el algoritmo deberá aplicar el descuento.

Algoritmo: Compra de llantas con descuento

Variables

- **n**: Cantidad de llantas
- **p**: Precio inicial de cada llanta
- **t**: Valor a pagar

Diagrama de flijo



Ingreso de datos por teclado

Verificar el contenido de **n**

Si compra más de 4 llantas

el precio se reduce en 10%

Cálculo del valor a pagar

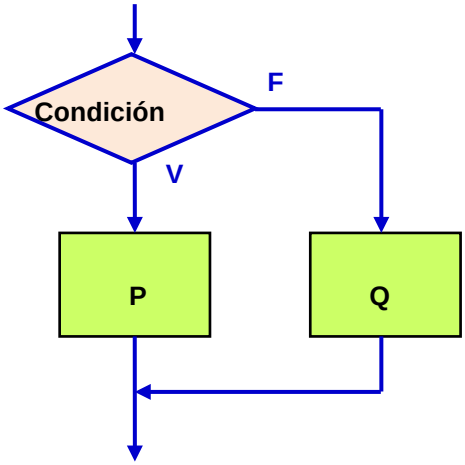
Salida del resultado a pantalla

Prueba. Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del bloque:

Pruebas	n	p	t	Salida
1	2	70	140	140
2	6	80	432	432
3	5	120	540	540

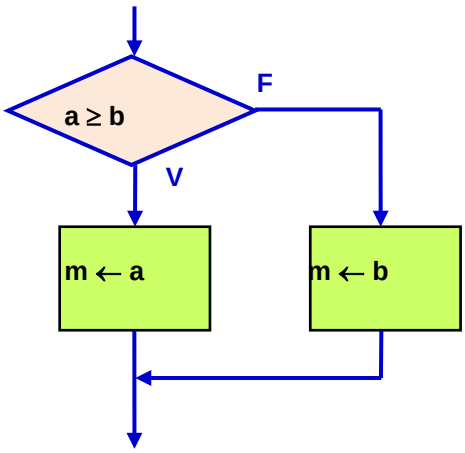
El algoritmo produce los resultados esperados con los datos de cada prueba

b) Ejecución selectiva de uno entre dos bloques



Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutará el bloque **P** asociado al valor verdadero, caso contrario, si el resultado es falso (**F**) se ejecutará el bloque **Q**. El algoritmo continua abajo, despues de ejecutar alguno de los dos bloques.

Ejemplo. Describa en notación algorítmica como asignar a la variable **m** el mayor entre dos valores almacenados respectivamente en las variables **a** y **b**



Antes del bloque, las variables **a** y **b** deben haber sido asignadas algún valor, caso contrario sería un error.

Ejemplo. Describa en notación algorítmica una solución al siguiente problema.

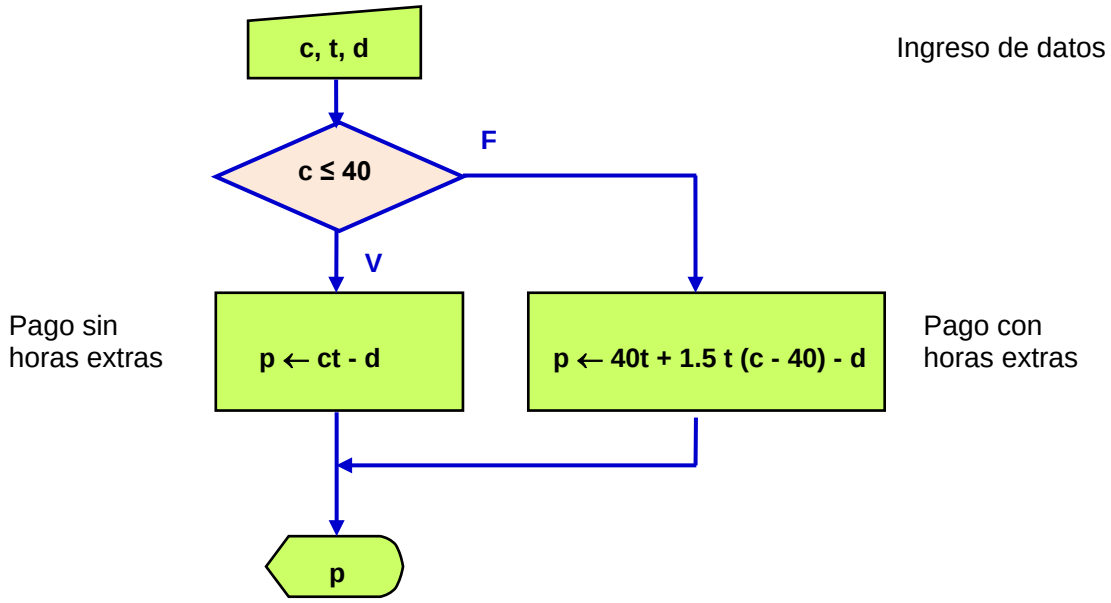
Para el pago semanal a un obrero se consideran los siguientes datos: horas trabajadas, tarifa por hora y descuentos. Si la cantidad de horas trabajadas en la semana es mayor a 40, se le debe pagar las horas en exceso con una bonificación de 50% adicional a la tarifa normal.

Algoritmo: Pago semanal a un obrero

Variables

- **c**: Cantidad de horas trabajadas en la semana
- **t**: Tarifa por hora
- **d**: Descuentos que se aplican al pago semanal
- **p**: Pago que recibe el obrero

Diagrama de flujo



Prueba. Realice algunas pruebas del algoritmo anterior. En cada una ingrese los datos necesarios desde fuera del bloque.

Pruebas	c	t	d	p	Salida
1	40	5	20	180	180
2	35	4	25	115	115
3	42	5	30	185	185

El algoritmo produce los resultados esperados con los datos de cada prueba

3.2.2 Ejercicios con la notación algorítmica

Algoritmos con decisiones

Para cada ejercicio desarrolle una solución en notación algorítmica y realice una prueba

1. Dados el radio y altura de un cilindro, si la altura es mayor al radio calcule y muestre el valor del volumen del cilindro, caso contrario muestre el valor del área del cilindro.

2. Lea la cantidad de Kw que ha consumido una familia y el precio por Kw. Si la cantidad es mayor a 700, incremente el precio en 5% para el exceso de Kw sobre 700. Muestre el valor total a pagar.

3. Lea un valor de temperatura **t** y un código **p** que puede ser **1** o **2**. Si el código es **1** convierta la temperatura **t** de grados **f** a grados **c** con la fórmula **c=5/9(t-32).** Si el código es **2** convierta la temperatura **t** de grados **c** a **f** con la fórmula: **f=32+9t/5.** Muestre el resultado.

4. Dadas las dimensiones de un bloque rectangular, calcule las diagonales de las tres caras diferentes. Muestre el valor de la mayor diagonal.

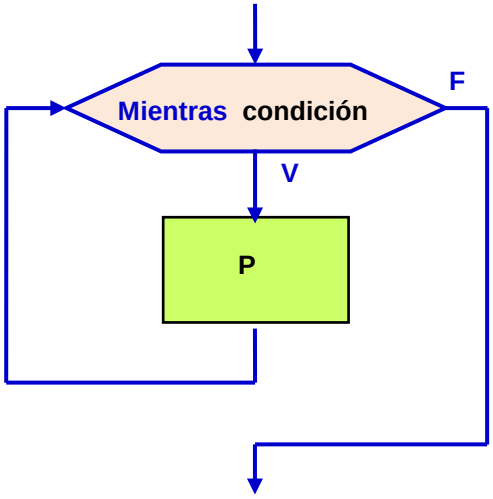
5. Dadas las tres calificaciones de un estudiante, encuentre y muestre la calificación mas alta.

3.2.3 Ciclos

Los ciclos o repeticiones son estructuras de control que se usan para describir la ejecución repetida de bloques de instrucciones..

Hay dos formas comunes de ciclos que se usan en la construcción de algoritmos.

a) Ejecución repetida de un bloque mediante una condición



Al entrar a esta estructura se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutarán las instrucciones dentro del bloque y regresará nuevamente a evaluar la condición.

Mientras la condición mantenga el valor verdadero (**V**), el bloque de instrucciones se ejecutará nuevamente. Esto significa que en algún ciclo al evaluar la condición deberá obtenerse el resultado falso (**F**) para salir de la estructura y continuar la ejecución después del bloque. Al diseñar el algoritmo deberán escribirse las instrucciones necesarias.

La condición es cualquier expresión cuyo resultado puede ser únicamente verdadero (**V**) o falso (**F**). Puede incluir operadores para comparar el contenido de variables y también se pueden usar los conectores de la lógica matemática.

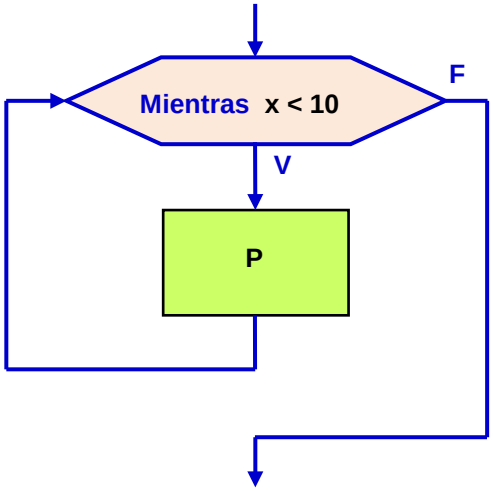
Ejemplo de expresiones que pudieran ser usadas como una condición. El resultado de cada una dependerá del contenido de las variables:

n > 0
a ≤ 5
x ≠ 4
a < 3 ∨ x > 1

Las estructuras de repetición son necesarias cuando las instrucciones de un bloque del algoritmo deben ejecutarse más de una vez para construír la solución.

Si el algoritmo que se propone para resolver un problema requiere repetir un bloque pero no se puede anticipar la cantidad de ciclos que deben realizarse, entonces deberia usarse la repetición controlada con una condición.

Ejemplo. Describa algorítmicamente la repetición de un bloque de instrucciones mientras la variable **x** tenga un valor menor a **10**



Antes del bloque, la variable **x** debe haber sido asignada con algún valor, caso contrario sería un error.

Es necesario que la variable **x** cambie su contenido dentro del bloque de instrucciones que se repiten para que en algún ciclo la repetición pueda terminar y la ejecución continue después del bloque. Caso contrario sería un error lógico pues el algoritmo permanecería en el ciclo.

Ejemplo. Describa en notación algorítmica una solución para el siguiente problema.

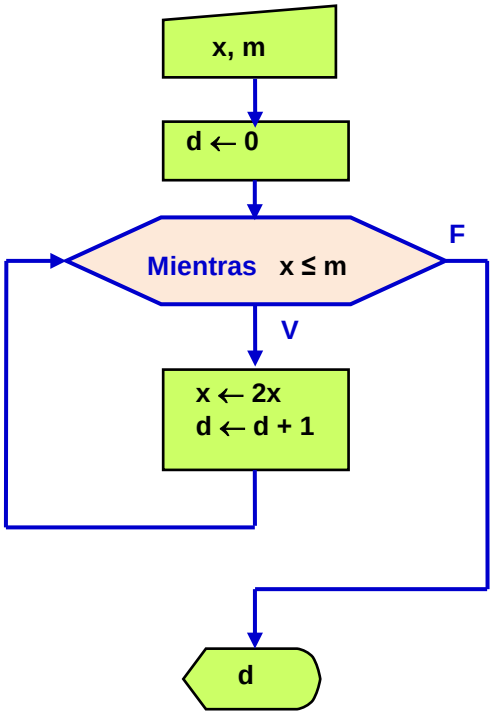
En un cultivo se tiene una cantidad inicial de bacterias. Cada día esta cantidad se duplica. Determine en que día la cantidad excede a un valor máximo.

Algoritmo: Crecimiento de la cantidad de bacterias

Variables

x: Cantidad inicial de bacterias
m: Cantidad máxima de bacterias
d: Día

Diagrama de flujo



Ingreso de los datos

Iniciar el día en cero

El ciclo se repetirá mientras la cantidad de bacterias no exceda al máximo.

Dentro del bloque se modifica la cantidad de bacterias y el día.

Mostrar el valor final del día

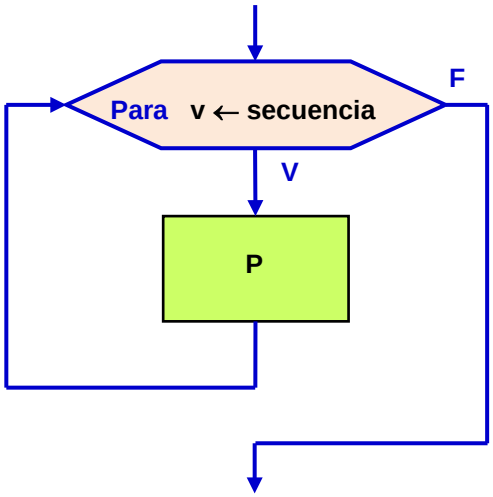
Note que se debe usar el ciclo condicionado pues no se puede anticipar la cantidad de repeticiones necesarias para que la cantidad de bacterias exceda al valor máximo.

Prueba. Realice una prueba del algoritmo anterior. Ingrese los datos desde fuera del bloque y registre los cambios en el contenido de las variables.

Prueba	x	m	d	Salida
	200	5000	0	
	400		1	
	800		2	
	1600		3	
	3200		4	
	6400		5	5

Se puede verificar que es el resultado esperado y que coincide con el valor que se puede calcular matemáticamente con la expresión $2^n x > m$

b) Ejecución repetida de un bloque mediante una secuencia



Para usar esta estructura de control es necesario especificar una variable para el conteo de repeticiones y una lista de valores o secuencia que puede tomar. El ciclo se repetirá con cada valor especificado para la variable. Al ejecutarse cada ciclo el valor de la variable cambiará siguiendo la especificacion.

Al entrar a esta estructura, se inicia la variable de control. Si esta variable no excede al valor final, se ejecuta el bloque y regresa nuevamente al inicio del ciclo y la variable toma el siguiente valor de la secuencia. Cuando el valor de la variable llegue al valor final, el ciclo finalizará y la ejecución continuará después del bloque.

La variable de control del ciclo puede especificarse con alguna notación que exprese cuales son los valores que puede tomar.

Ejemplos

Mediante una lista de valores:

v ← [2, 5, 4, 7, 6]

Indicando el valor inicial, el valor final de la secuencia y el incremento:

v ← 1, 10, 1

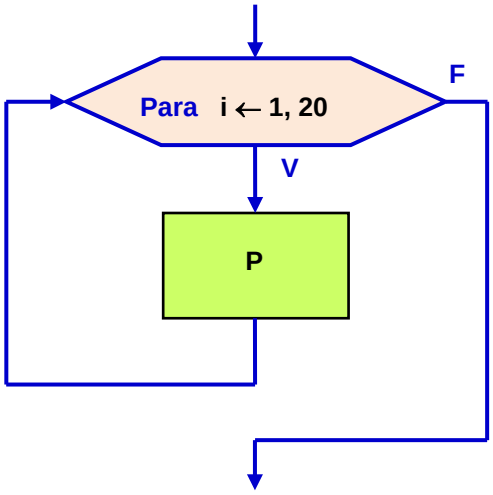
Equivale a la secuencia: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

v ← 1, 15, 2

Equivale a la secuencia: **1, 3, 5, 7, 9, 11, 13, 15**

Si no se escribe el incremento, se supondrá que es la unidad

Ejemplo. Especificar una variable de control para que el bloque de instrucciones **P** se repita **20** veces



Ejemplo. Describa en notación algorítmica una solución al siguiente problema.

Dado un entero positivo **n**, se desea verificar que la suma de los primeros **n** números impares es igual a $\mathbf{n^2}$

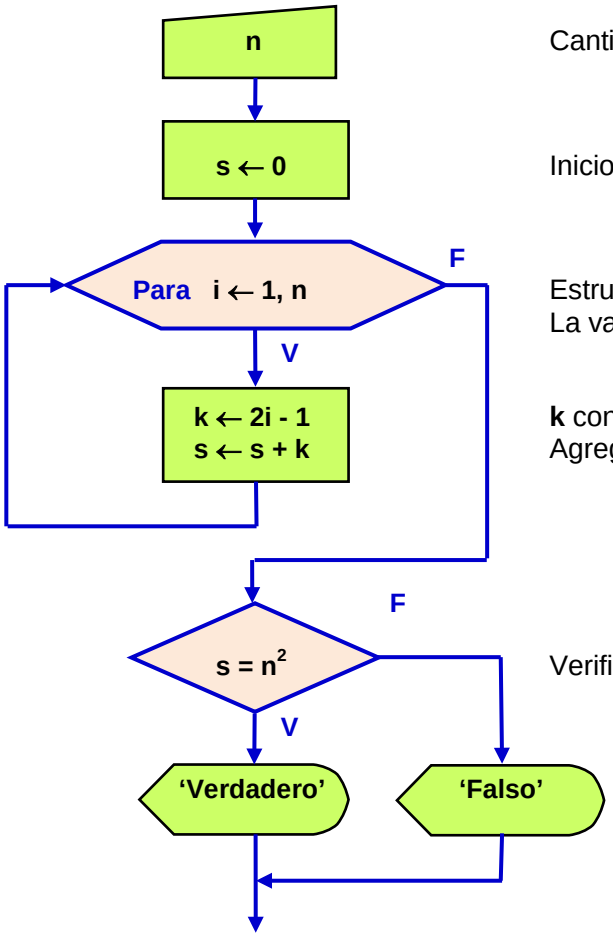
Ej. n = 5: $\mathbf{1 + 3 + 5 + 7 + 9 = 5^2}$

Algoritmo: Suma de impares

Variables

n: Cantidad de números impares
k: Cada número impar
s: Suma de impares
i: Conteo de ciclos

Diagrama de flujo



Prueba. Realice una prueba del algoritmo anterior. Ingrese un dato desde fuera del bloque y registre los cambios en el contenido de las variables.

Prueba	n	Ciclo i	Impar k	s	Salida
	5			0	
		1	1	1	
		2	3	4	
		3	5	9	
		4	7	16	
		5	9	25	**'Verdadero'**

Se verifica que el resultado es **'Verdadero'.**

Note que este algoritmo no constituye una demostración matemática. Solo verfica que la propiedad se cumple para valores específicos.

Las estructuras de repetición son necesarias cuando un bloque del algoritmo debe ejecutarse más de una vez para construír la solución.

Si se puede anticipar la cantidad de ciclos que se deben realizar, entonces conviene usar la repetición controlada con un conteo de ciclos.

3.2.4 Ejercicios con la notación algorítmica

Algoritmos con ciclos

Para cada ejercicio desarrolle una solución en notación algorítmica y realice una prueba

1. Calcule el mayor valor de los pesos de **n** paquetes en una bodega. Estos datos ingresan uno a la vez dentro de un ciclo. Al inicio ingrese el valor de **n** para especificar la cantidad de ciclos que se realizarán

2. Lea los votos de **n** personas en una consulta. Cada voto es un número **0,** o **1** correspondiente a la opción a favor (**1**) o en contra (**0**). Al inicio lea el valor de **n** para especificar la cantidad de ciclos que se realizarán. Muestre el resultado de la consulta.

3. Determine la suma de los **n** primeros números de la serie: **1, 1, 2, 3, 5, 8, 13, 21, ...** en la cual cada término, a partir del tercero, se obtiene sumando los dos términos anteriores

4. Calcule un valor aproximado para la constante **π** usando la siguiente expresión:

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 \ldots$$

La cantidad de términos es un dato que debe ser ingresado al inicio del algoritmo.

5. Determine la cantidad de términos que deben sumarse de la serie $1^2 + 2^2 + 3^3 + 4^4 + \ldots$ para que el valor de la suma sea mayor a un número **x** ingresado al inicio.

6. El inventor del juego del ajedréz pidió a su rey que como recompensa le diera por la primera casilla 2 granos de trigo, por la segunda, 4 granos, por la tercera 8, por la cuarta 16, y así sucesivamente hasta llegar a la casilla 64. El rey aceptó. Suponga que cada Kg. de trigo consta de 20000 granos de trigo. Si cada tonelada tiene 1000 Kg. describa un algoritmo para calcular la cantidad de toneladas de trigo que se hubiesen necesitado.

En el ciclo describa la suma $2^1 + 2^2 + 2^3 + 2^4 + \ldots + 2^{64}$

4 Desarrollo de algoritmos en el lenguaje MATLAB

En esta sección se revisa el componente programable de MATLAB. Para una mejor comprensión del tema es conveniente tener alguna experiencia previa con el componente interactivo de MATLAB. Con este propósito puede revisar el **Anexo** a este documento: **MATLAB INTERACTIVO** o alguna otra referencia disponible.

Un programa es la descripción de un algoritmo en un lenguaje computacional. Para escribir un programa es necesario tener estructurado el algoritmo para resolver el problema de interés y conocer las reglas de sintaxis y semántica del lenguaje computacional que será usado. También deberá tener instalado en su computador el traductor del lenguaje. En este documento se usará MATLAB.

4.1 Algunos elementos básicos para escribir algoritmos en MATLAB

Esta sección resumida se describe con mayor detalle y con ejemplos en el anexo a este documento: **MATLAB INTERACTIVO**.

4.1.1 Variables o identificadores

Son los símbolos para representar los valores y otros componentes de los programas. Para escribir variables se pueden usar letras, mayúsculas y minúsculas, dígitos y el sub-guión. No se pueden usar tildes ni la letra ñ y deben comenzar con una letra.

Se recomienda no usar nombres para variables que coincidan con las palabras reservadas que tienen un significado especial para el lenguaje de programación. Se sugiere usar nombres significativos, relacionados con los valores que representarán.

4.1.2 Símbolos especiales

, ; : %

4.1.3 Operadores aritméticos

+ - * / \ ^

4.1.4 Símbolos de agrupación

( ) [ ] { }

4.1.5 Funciones matemáticas

sin, cos, tan,…exp,log,log10,sqrt,…abs,fix,mod,sign,…

4.1.6 Símbolos numéricos especiales

pi, eps, Inf, NaN, …

4.1.7 Comandos para mostrar resultados numéricos en pantalla

format short
format long
format bank
format bank

4.1.8 Comandos del sistema operativo

clear, cls, date, clock, tic, toc, dir, ...

4.1.9 Operadores relacionales

< , <=, >, >=, ==, ~=

4.1.10 Operadores lógicos

&, |, ~

4.1.11 Un ejemplo introductorio en modo interactivo

En esta sección se resuelve un problema geométrico simple escribiendo y ejecutando directamente cada instrucción en la ventana de comandos de MATLAB (modo interactivo) para comparar con el uso del componente programable de MATLAB el cual requiere escribir y almacenar las instrucciones en un archivo antes de la ejecución.

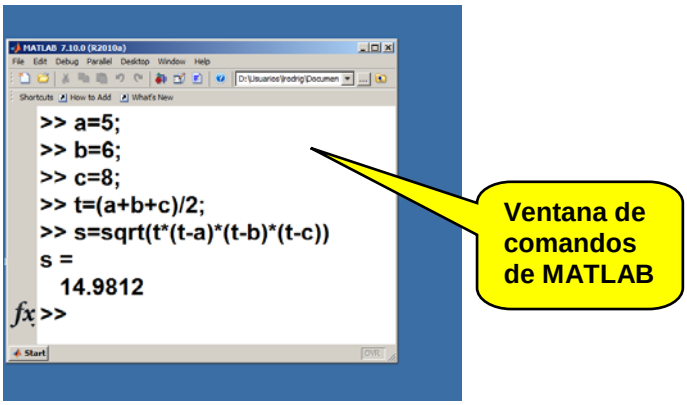
Ejemplo: Calcule el área de un triángulo dados sus tres lados

Variables

Datos: **a, b, c:** Lados del triángulo (suponer que miden miden **5**, **6**, y **8** cm.)
Resultado: **s:** Área del triángulo
Fórmula: $\mathbf{s = \sqrt{t(t-a)(t-b)(t-c)}}$, siendo **t = (a + b + c)/2**

Solución en modo interactivo en la ventana de comandos

Ingrese a la ventana de comandos de MATLAB, digite cada instrucción y observe el resultado de la ejecución de cada instrucción inmediatamente después de ser ingresada. Si omite el punto y coma al final de cada línea se pueden ver los resultados de las asignaciones.



La desventaja del modo interactivo es que cada vez que requiera resolver un problema similar pero con datos diferentes, tiene que digitar nuevamente las instrucciones y si sale de MATLAB deberá escribir completamente todas las instrucciones.

4.2 Instrucciones básicas para escribir programas en MATLAB

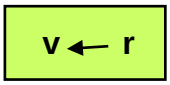
Cada instrucción de MATLAB se describirá relacionándola con su descripción en notación algorítmica desarrollada en la sección anterior de este documento.

a) Instrucción de asignación

Esta instrucción se usa para definir variables y asignar un valor a su contenido

Sean **v** una variable y **r** un valor

Notación algorítmica



Asigna el valor **r** a la variable **v**

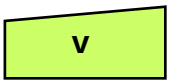
Lenguaje MATLAB

```
v = r
```

b) Instrucción para ingreso de datos

Esta instrucción se usa para describir la acción de ingresar algún valor para una variable desde fuera del algoritmo cuando este sea ejecutado. Esta instrucción permite que los datos no requieran ser asignados dentro del algoritmo y así pueden realizarse pruebas con diferentes datos que entran desde afuera del algoritmo.

Notación algorítmica



Ingresar desde afuera del algoritmo un valor para la variable **v**

Lenguaje MATLAB

```
v = input('algún mensaje ');
```

En esta instrucción de MATLAB se debe incluir un mensaje que se mostrará al usuario para indicarle que es el momento de ingresar el dato.

c) Instrucción para salida de resultados

Esta instrucción se usa para describir la acción de mostrar afuera del algoritmo mensajes o el contenido de variables con los resultados de interés. Para mostrar el contenido de una variable se debe escribir su nombre. Los mensajes deben escribirse entre comillas.

Notación algorítmica

Mostrar afuera del algoritmo el valor que contiene la variable **v**

Lenguaje MATLAB

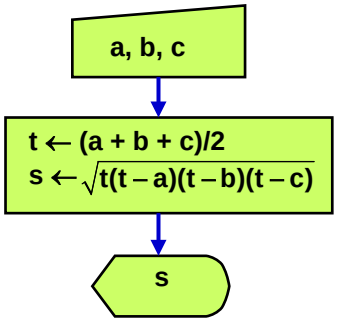
```
disp( v );
```

Ejemplo. Escriba en el lenguaje MATLAB el algoritmo para calcular el área de un triángulo dados sus tres lados

Algoritmo: Área de un triángulo

Variables

Datos: **a, b, c:** Lados del triángulo
Resultado: **s:** Área del triángulo
Fórmula: $s = \sqrt{t(t-a)(t-b)(t-c)}$, siendo $t = (a + b + c)/2$



Lenguaje MATLAB

```
%Area de un triángulo
a=input('Primer lado ');
b=input('Segundo lado ');
c=input('Tercer lado ');
t = (a + b + c)/2;
s = sqrt(t*(t - a)*(t - b)*(t - c));
disp('El área es');
disp(s);
```

Los programas se los escribe en la ventana de edición de MATLAB. Cuando está completo se lo almacena con algún nombre y posteriormente se lo podrá ejecutar escribiendo el nombre del programa almacenado.

En el lenguaje MATLAB para designar nombres de variables normalmente se usan minúsculas, pero pueden usarse mayúsculas y nombres con más caracteres.

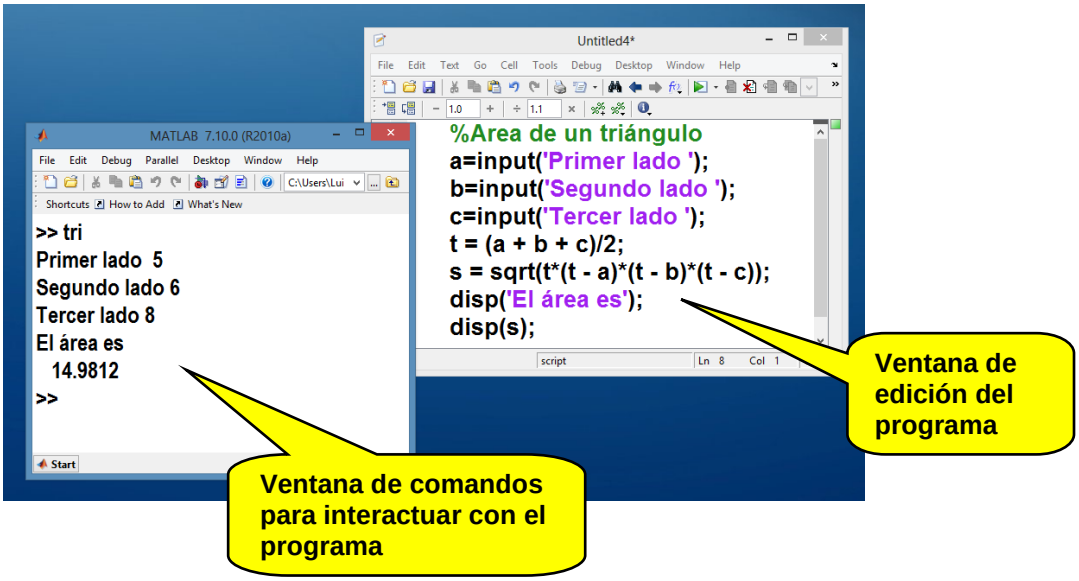
Es recomendable escribir alguna anotación para identificar al programa. Estas anotaciones o comentarios se los escribe comenzando con el símbolo **%**

Al final de cada línea se escribe el punto y coma. El punto y coma evita que salga a la pantalla cada asignación o cálculo que realiza el programa cuando es ejecutado.

Los colores en el programa son asignados por MATLAB, pero pueden personalizarse, como también el tipo y tamaño de las letras, etc.

Para probar el programa se escribe en la ventana de comandos el nombre con el cual fue almacenado. Suponer que es **tri**. Luego se ingresan los datos como se muestra en el siguiente gráfico copiado de como luce la pantalla de MATLAB

Interacción con MATLAB



Para realizar pruebas con diferentes datos, es muy conveniente que los programas sean independientes de los datos, es decir, los datos deben ingresarse desde fuera del programa. Esto se consigue con el uso de las instrucciones de entrada.

Adicionalmente, es una buena práctica de programación insertar algún **comentario** para identificar el problema que se está resolviendo. Para incluir comentarios o anotaciones en el programa inicie la línea con el símbolo **%**

Para realizar cambios en el programa que está abierto en la ventana de edición, posicione el cursor en el lugar respectivo, realice los cambios, almacene el archivo y ejecútelo nuevamente. Puede cargar y modificar los programas que están almacenados, escribiendo su nombre.

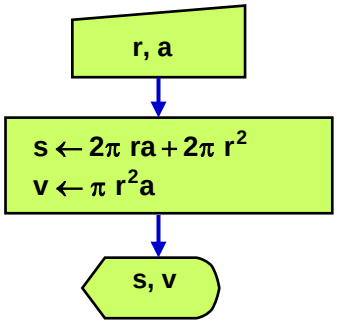
Ejemplo. Calcule el área y el volumen de un cilindro dados su radio y altura.

Algoritmo: Área y volumen de un cilindro

Variables

Datos: **r:** Radio del cilindro
a: Altura del cilindro
Resultados: **s:** Área del cilindro
v: Volumen del cilindro

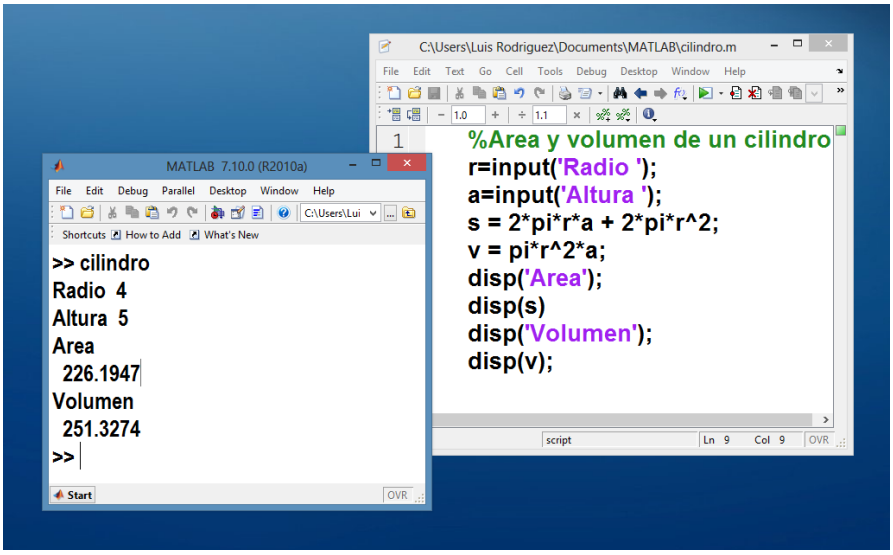
Fórmulas: $s = 2\pi ra + 2\pi r^2$
$v = \pi r^2 a$



Lenguaje MATLAB

```
%Area y volumen de un cilindro
r=input('Radio ');
a=input('Altura ');
s = 2*pi*r*a + 2*pi*r^2;
v = pi*r^2*a;
disp('Area');
disp(s);
disp('Volumen');
disp(v);
```

Interacción en MATLAB

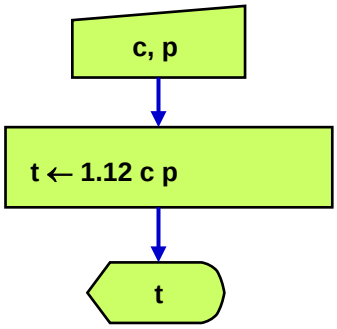


Ejemplo. Calcular el valor de una compra con los datos de la cantidad comprada y el precio unitario. Agregar el impuesta del IVA.

Algoritmo: Área y volumen de un cilindro

Variables

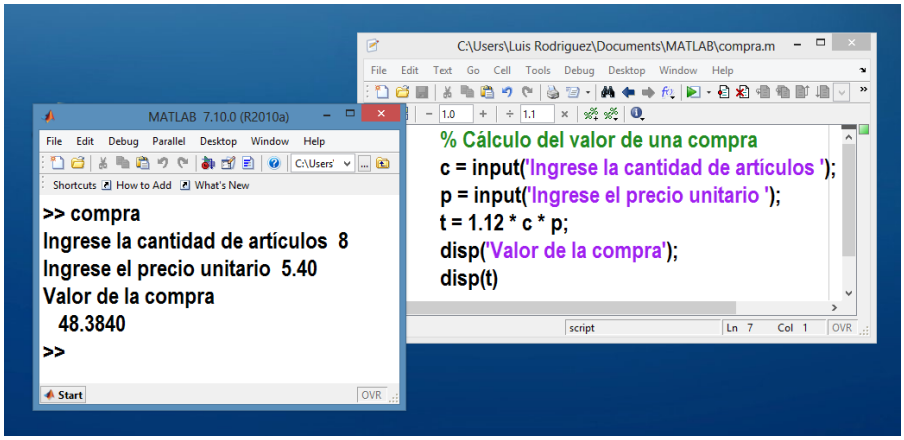
Datos:	**c:**	Cantidad de artículos
	p:	Precio unitario
Resultado:	**t:**	Valor a pagar
Cálculo:		Agregar 12% al valor de la compra, por concepto de IVA



Lenguaje MATLAB

```
% Cálculo del valor de una compra
c = input('Ingrese la cantidad de artículos ');
p = input('Ingrese el precio unitario ');
t = 1.12 * c * p;
disp('Valor de la compra');
disp(t);
```

Interacción con MATLAB



4.2.1 Ejercicios de programación con las instrucciones básicas

Para cada ejercicio escriba y pruebe un programa en el lenguaje MATLAB

1. Dados el radio y altura de un cilindro calcule el área total y el volumen

2. Se tiene un recipiente cilíndrico con capacidad en litros. Su altura es un dato en metros. Determine el diámetro de la base

3. Dadas las tres dimensiones de un bloque rectangular calcule y muestre su área total y su volumen.

4. La siguiente fórmula proporciona el enésimo término **u** de una progresión aritmética:

$$u = a + (n - 1)\, r$$

en donde **a** es el primer término, **n** es el la cantidad de términos y **r** es la razón entre dos términos consecutivos. Calcular el valor de **r** dados **u, a, n**

5. En el ejercicio anterior, calcular el valor de: **n** dados **u, a, r**

6. El examen de una materia es el 70% de la nota total. Las lecciones constituyen el 20% y las tareas el 10% de la nota total. Ingrese como datos la nota del examen calificado sobre 100 puntos, la nota de una lección calificada sobre 10 puntos, y las notas de tres tareas calificadas cada una sobre 10 puntos. Calcule la calificación total sobre 100 puntos.

7. Un modelo de crecimiento poblacional está dado por: $n = 5t + 2e^{0.1t}$, en donde **n** es el número de habitantes, **t** es tiempo en años.

Calcule el número de habitantes que habrían en los años 5, 10 y 20

8. Un ingeniero desea tener una cantidad de dólares acumulada en su cuenta de ahorros para su retiro luego de una cantidad de años de trabajo. Para este objetivo planea depositar un valor mensualmente. Suponga que el banco acumula el capital mensualmente mediante la siguiente fórmula:

$$A = P\left[\frac{(1+x)^n - 1}{x}\right], \text{ en donde}$$

A: Valor acumulado
P: Valor de cada depósito mensual
n: Cantidad de depósitos mensuales
x: Tasa de interés mensual

Calcule el valor acumulado ingresando como datos valores para **P**, **n**, **x**

9. Una empresa produce fertilizantes. Cada mes el ingreso por ventas en miles de dólares se describe con $v = 0.4x\,(30 - x)$ mientras que el costo de producción en miles de dólares es $c = 5 + 10\ln(x)$, siendo **x** la cantidad producida en toneladas, $1<x<30$.

Determine el valor del ingreso neto, dado un valor para **x**.

4.2.2 Funciones para aritmética con enteros

Muchas aplicaciones computacionales requieren el uso de operaciones especiales. Un caso frecuente es la obtención del cociente entero de la división entre dos números y el resíduo de esta división

Las funciones **FIX** y **MOD** de MATLAB se pueden usar para estas operaciones aritméticas.

FIX: Trunca los decimales de un número y entrega la parte entera.

```
>> c=fix(20/6)
c =
    3
```

MOD: Entrega el residuo de la división entera entre dos números

```
>> r=mod(20,6)
r =
    2
```

Ejemplo. Dado un número entero de dos cifras, sumar sus cifras.

```
%Sumar las cifras de un número de dos cifras
n=input('Ingrese un numero de dos cifras ');
d=fix(n/10);
u=mod(n,10);
s=d+u;
disp('Suma ');
disp(s);
```

Ejemplo. Dado un número entero de dos cifras, mostrarlo con las cifras en orden opuesto.

```
%Intercambiar las cifras de un número entero de dos cifras
n=input('Ingrese un numero de dos cifras ');
d=fix(n/10);
u=mod(n,10);
r=10*u+d;
disp('Numero invertido ');
disp(r);
```

Ejemplo. Dado un número entero (cantidad de dólares), mostrar el valor equivalente usando la menor cantidad de billetes de 100, 50, 20, 10, 5 y 1.

x: cantidad de dinero
c: cantidad de billetes

```
%Cambiador de billetes
x=input('Ingrese la cantidad de dinero ');
c=fix(x/100);
disp(c);
disp('billetes de 100');
x=mod(x,100);
c=fix(x/50);
disp(c);
disp('billetes de 50');
x=mod(x,50);
c=fix(x/20);
disp(c);
. . .                                    (continuar)
```

4.2.3 Ejercicios de programación con las funciones fix y mod

Para cada ejercicio escriba y pruebe un programa con MATLAB

1. Escriba un programa para ingresar un número entero (dias). Determine y muestre el equivalente en meses y días sobrantes. Por simplicidad suponga que cada mes tiene 30 días. Use **fix** para convertir a meses y **mod** para obtener días sobrantes**.** Ej. 198 días equivalen a 6 meses y 18 días

2. Lea dos números de tres cifras cada uno. Sume la cifra central del primer número con la cifra central del segundo número y muestre el resultado.

3. Dado un número entero de tres cifras. Muestre el mismo número pero con las cifras en orden opuesto.

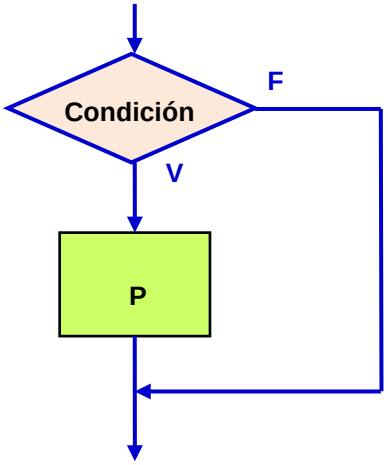
4.3 Decisiones

Las decisiones son operaciones computacionales que permiten condicionar la ejecución de instrucciones dependiendo del resultado de una expresión cuyo resultado únicamente puede ser: **verdadero** o **falso**

4.3.1 Ejecución condicionada de un bloque de instrucciones

Esta estructura de control se usa para condicionar la ejecución de un bloque de instrucciones utilizando como criterio el resultado de una condición

Notación algorítmica



Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutarán las instrucciones en el bloque **P** caso contrario, si el resultado es falso (**F**) el bloque no será ejecutarlo. En ambos casos el algoritmo continua abajo del bloque.

Lenguaje MATLAB

```
if  Condición
    Instrucciones en el bloque  P
end
```

Note la relación entre la instrucción del lenguaje computacional (**color azul**) y la estructura de control del lenguaje algorítmico (**color azul**). El color se ha elegido para visualizar mejor la equivalencia. Ambas describen la misma acción.

4.3.2 Operadores relacionales y lógicos

Para escribir la expresión que condiciona la ejecución del bloque de instrucciones se pueden usar operadores relacionales y conectores lógicos. Para que una expresión pueda ser usada como una condición, las variables incluidas en la expresión deben tener asignado algún valor, caso contrario será un error pues la condición no podría evaluarse.

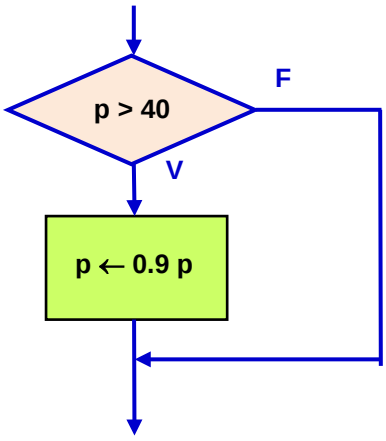
En el lenguaje MATLAB las expresiones usan una simbología especial:

Matemáticas	MATLAB
Operadores relacionales	
<, >, ≤, ≥, =, ≠	**<, >, <=, >=, ==, ~=**
Conectores lógicos	
∧, ∨, ¬	**&, |, ~**

Ejemplo. **x<3 & t>2** es una condición, y su valor (**verdadero** o **falso**) dependerá del contenido de las variables **x** y **t**

Si los valores que se relacionan con los conectores lógicos **∧, ∨** son valores escalares simples, se pueden usar los símbolos **&&** y **||** como conectores lógicos en lugar de **&** y **|**

Ejemplo. Describa en notación algorítmica como reducir en **10%** el valor almacenado en la variable **p** en caso de que su valor actual sea mayor a **40**



Lenguaje MATLAB

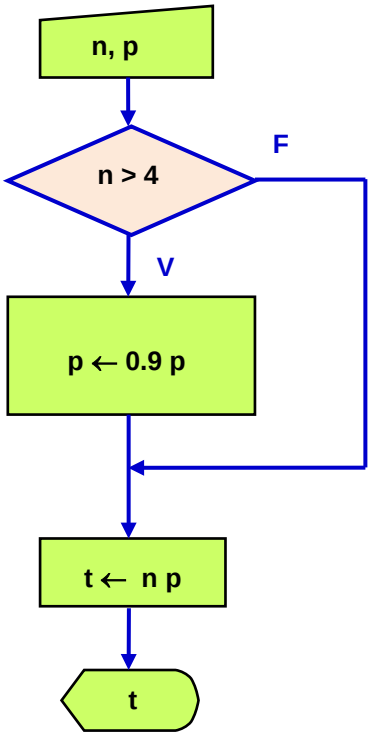
```
If  p > 40
    p = 0.9*p;
end
```

Ejemplo. Calcular el total debe pagar una persona en un almacén de llantas. Los datos son la cantidad de llantas compradas y el precio unitario. Si la cantidad comprada es mayor a 4 llantas, el precio unitario se reduce en el 10%.

Algoritmo: Compra de llantas con descuento

Variables

n: Cantidad de llantas
p: Precio unitario
t: Valor a pagar



Prueba. Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del bloque:

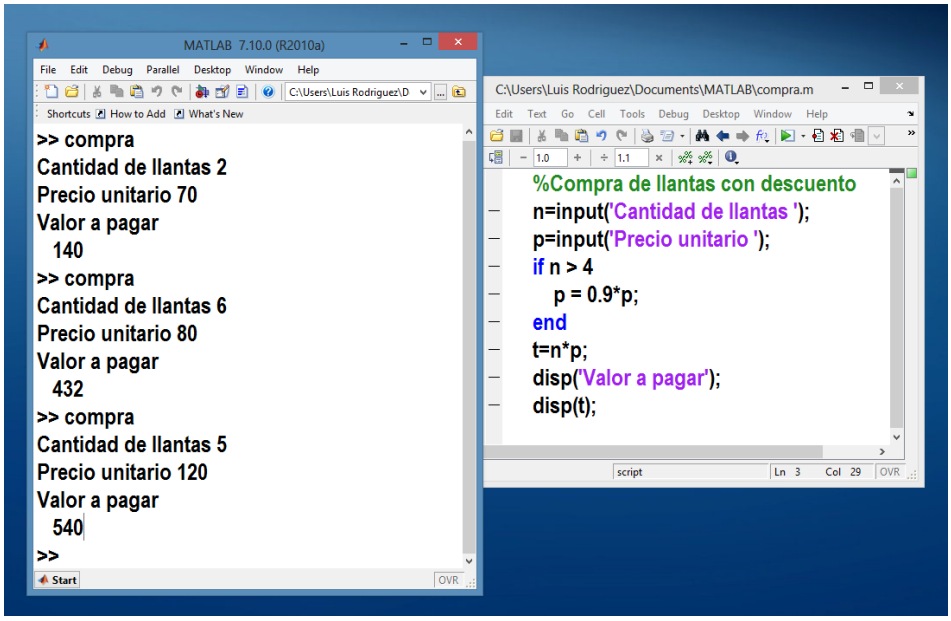
Pruebas	n	p	t	Salida
1	2	70	140	140
2	6	80	432	432
3	5	120	540	540

El algoritmo produce los resultados esperados con los datos de cada prueba

Lenguaje MATLAB

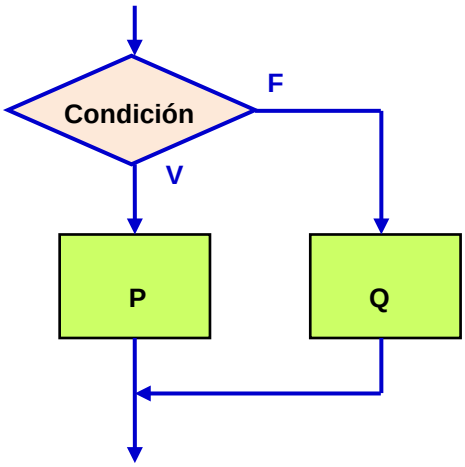
```
%Compra de llantas con descuento
n=input('Cantidad de llantas ');
p=input('Precio unitario ');
if n > 4
   p = 0.9*p;
end
t=n*p;
disp('Valor a pagar');
disp(t);
```

Interacción con MATLAB



4.3.3 Ejecución selectiva de uno entre dos bloques de instrucciones

Esta estructura de control del flujo del algoritmo chequea una condición y dependiendo del resultado realiza las instrucciones incluidas en una de las dos opciones

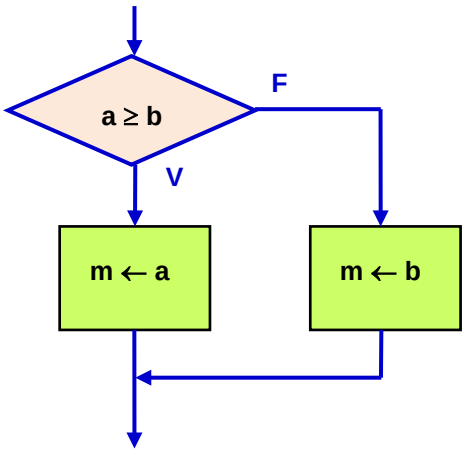


Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutará el bloque **P** asociado al valor verdadero, caso contrario, si el resultado es falso (**F**) se ejecutará el bloque **Q**. El algoritmo continua abajo, despues de ejecutar alguno de los dos bloques.

Lenguaje MATLAB

```
if   condición
     instrucciones en el bloque  P
else
     instrucciones en el bloque  Q
end
```

Ejemplo. Describa en notación algorítmica como asignar a la variable **a** el mayor entre dos valores almacenados respectivamente en las variables **a** y **b**



Lenguaje MATLAB

```
if   a >= b
     m = a;
else
     m = b;
end
```

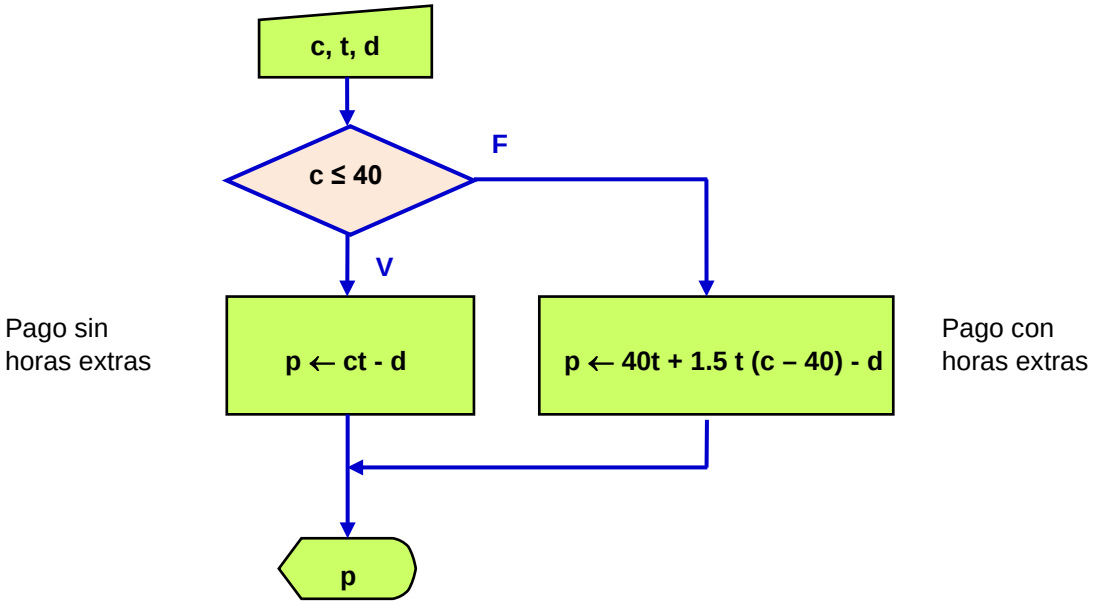
Ejemplo. Describa en el lenguaje MATLAB el algoritmo para resolver el siguiente problema.

Para el pago semanal a un obrero se consideran los siguientes datos: horas trabajadas, tarifa por hora y descuentos. Si la cantidad de horas trabajadas en la semana es mayor a 40, se le debe pagar las horas en exceso de 40 con una bonificación de 50% adicional al pago normal.

Algoritmo: Pago semanal a un obrero

Variables

c: Cantidad de horas trabajadas en la semana
t: Tarifa por hora
d: Descuentos que se aplican al pago semanal
p: Pago que recibe el obrero



Prueba. Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del bloque:

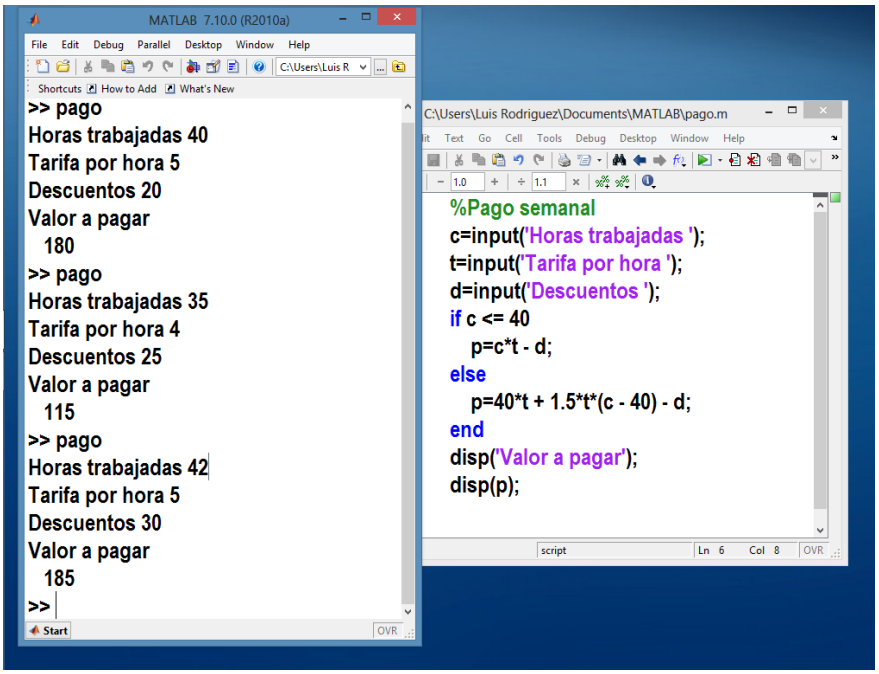
Pruebas	c	t	d	p	Salida
1	40	5	20	180	180
2	35	4	25	115	115
3	42	5	30	185	185

El algoritmo produce los resultados esperados con los datos de cada prueba

Lenguaje MATLAB

```
%Pago semanal
c=input('Horas trabajadas ');
t=input('Tarifa por hora ');
d=input('Descuentos ');
if t <= 40
   p=c*t - d;
else
   p=40*t + 1.5*t*(c - 40) - d;
end
disp('Valor a pagar');
disp(p);
```

Interacción con MATLAB



Práctica con decisiones en MATLAB

Ejemplo. Dado un número entero determinar si es un número par o impar.

Lenguaje MATLAB

```
%Determinar si un número es para o impar
x=input('Ingrese un numero ');
r=mod(x, 2);
if r == 0
   disp('Par');
else
   disp('Impar');
end
```

Ejemplo. Dados dos números enteros, determinar si uno es múltiplo del otro

Lenguaje MATLAB

```
%Determinar si un número es múltiplo de otro
a=input('Ingrese el primer numero ');
b=input('Ingrese el segundo numero ');
if mod(a,b) == 0 | mod(b,a) == 0
   disp('son multiplos');
else
   disp('no son multiplos');
end
```

4.3.4 Decisiones múltiples

Para describir la selección de una acción entre varias opciones posibles se pueden estructurar decisiones dentro de otras decisiones

Ejemplo. Suponga que un local vende llantas para cierto tipo de carro con la siguiente política:

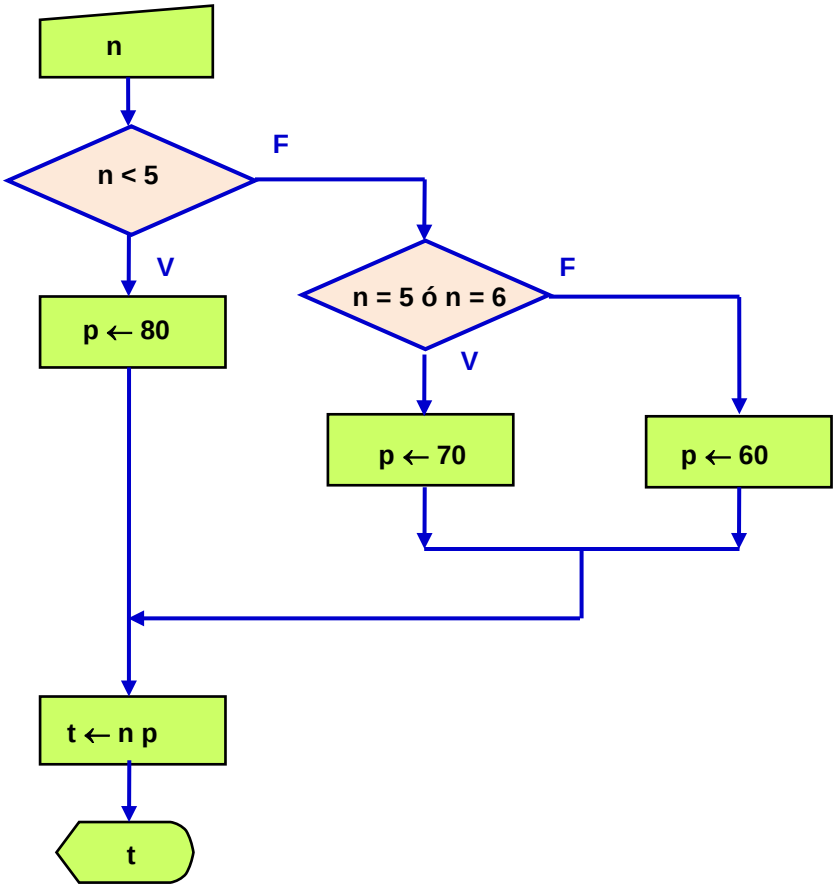
Cantidad	**Precio unitario**
Menos de 5	80
5 o 6	70
Más de 6	60

Lea el número de llantas de una compra y muestre el valor que debe pagar

Algoritmo: Compra de llantas con descuento

Variables

n: Cantidad de llantas compradas
p: Precio unitario (80, 70, o 60)
t: Valor de la compra



Prueba. Realice algunas pruebas del algoritmo anterior. En cada prueba ingrese los datos necesarios desde fuera del bloque:

Pruebas	n	t	Salida
1	4	320	320
2	6	420	420
3	8	480	480

El algoritmo produce los resultados esperados con los datos de cada prueba

Lenguaje MATLAB.

```
% Compra de llantas con descuento
n=input('Cantidad de llantas ');
if n<5
   p=80;
else
   if n==5 | n==6
      p=70;
   else
      p=60;
   end
end
t=n*p
disp('Valor a pagar ');
disp(t);
```

En todos estos ejemplos se supone que los datos ingresados son correctos.

Ejemplo. El precio de una pizza depende de su tamaño según la siguiente tabla:

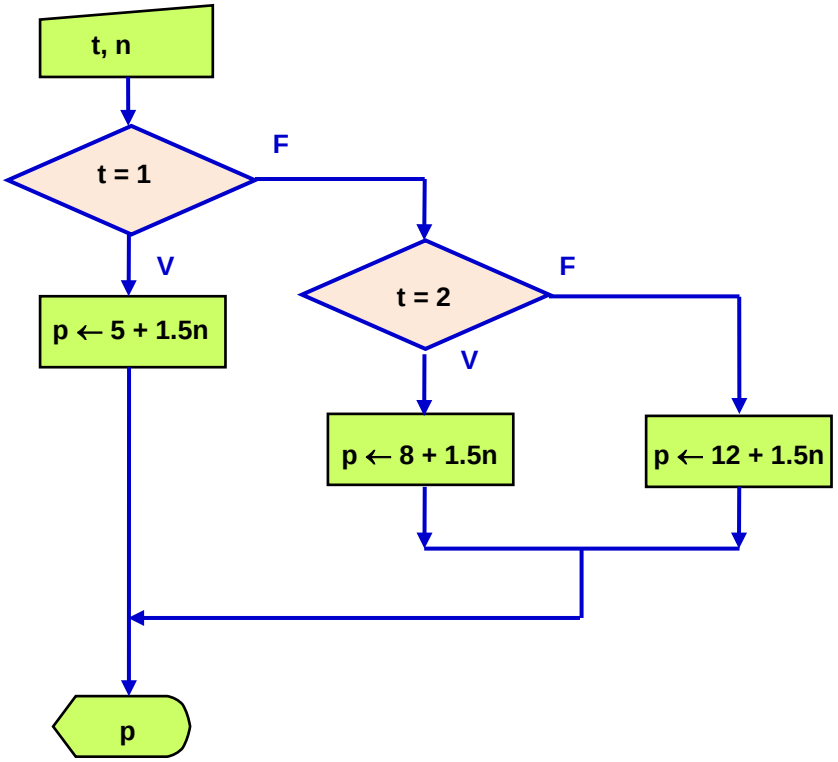
Tamaño	Precio
1	$5
2	$8
3	$12

Cada ingrediente adicional cuesta $1.5. Lea el tamaño de la pizza y el número de ingredientes y muestre el precio que debe pagar

Algoritmo: Compra de pizza

Variables

t: Tamaño de pizza
n: Número de ingredientes
p: Valor de a pagar



Lenguaje MATLAB.

```
%Compra de una pizza
t=input('Tamaño de la pizza ');
n=input('Número de ingredientes ');
if t==1
   p=5+1.5*n;
else
   if t==2
      p=8+1.5*n;
   else
      p=12+1.5*n;
   end
end
disp('Valor a pagar ');
disp(p);
```

¿Qué ocurriría si el dato del tamaño de la pizza no fuese correcto?

En la siguiente solución se considera la posibilidad de que el dato ingresado sea incorrecto. En este caso el valor a pagar será cero

```
%Compra de una pizza
t=input('Tamaño de la pizza ');
n=input('Número de ingredientes ');
if t==1
   p=5+1.5*n;
else
   if t==2
      p=8+1.5*n;
   else
      if t==3
         p=12+1.5*n;
      else
         p=0;
      end
   end
end
disp('Valor a pagar ');
disp(p);
```

4.3.5 La instrucción SWITCH

Cuando un algoritmo tiene decisiones múltiples, en ciertos casos una estructura alternativa es la instrucción **SWITCH** de **MATLAB.** Esta estructura de decisión permite realizar una o más instrucciones agrupadas en casos, dependiendo del valor de una varfiable de control:

```
switch variable de control
        case valor,
                instrucciones
        case valor,
                instrucciones
        case valor,
                instrucciones
         ...

        otherwise,
                instrucciones
end
```

Para usar esta instrucción la **variable de control** debe ser de tipo cardinal, es decir que se pueda contar, por ejemplo, números, letras, etc, pero no puede ser de tipo real con decimales.

La sección **otherwise** es opcional y es ejecutada cuando no se activa ninguno de los casos incluidos en la instrucción **switch**.

Se pueden omitir las comas luego del valor de cada caso si se escriben las instrucciones en la siguiente línea después de **case**

Ejemplo. Resuelva el problema de la pizza usando la instrucción SWITCH

```
%Compra de una pizza
t=input('tamaño de la pizza ');
n=input('número de ingredientes ');
switch t
   case 1
      p=5+1.5*n;
   case 2
      p=8+1.5*n;
   case 3
      p=12+1.5*n;
   otherwise
       p=0;
end
disp('Valor a pagar ');
disp(p);
```

Esta solución incluye la posibilidad de que el dato ingresado sea incorrecto. En este caso el valor a pagar será cero.

Ejemplo. Dado un número entero entre 1 y 5 mostrar el día de la semana.

Use la instrucción **switch.**

d: Contendrá el número del día

La salida es un mensaje

Una solución en MATLAB

```
%Día de la semana
d=input('Numero del dia ');
switch d
   case 1, disp('Lunes');
   case 2, disp('Martes');
   case 3, disp('Miercoles');
   case 4, disp('Jueves');
   case 5, disp('Viernes');
end
```

En la siguiente solución, si el dato no es correcto, se desea mostra un mensaje:

```
d=input('Numero del dia ');
switch d
   case 1, disp('Lunes');
   case 2, disp('Martes');
   case 3, disp('Miercoles');
   case 4, disp('Jueves');
   case 5, disp('Viernes');
   otherwise, disp('Error');
end
```

Si se desean asignar varios valores de la variable de control a algún caso, los valores deben encerrarse entre llaves como se muestra en el siguiente ejemplo:

```
d=input('Numero del dia ');
switch d
   case 1, disp('Lunes');
   case 2, disp('Martes');
   case 3, disp('Miercoles');
   case 4, disp('Jueves');
   case 5, disp('Viernes');
   case {6,7}, disp('Feriado');
   otherwise, disp('Error');
end
```

4.3.6 Ejercicios de programación con decisiones

Para cada ejercicio escriba y pruebe un programa en la ventana de edición de MATLAB

1. Dados el radio y altura de un cilindro, si la altura es mayor al radio calcule y muestre el valor del volumen del cilindro, caso contrario muestre el valor del área del cilindro.

2. Dados el radio y altura de un cilindro, si la altura es mayor al radio calcule y muestre el valor del volumen, caso contrario muestre el mensaje: 'Error'

3. Dadas las dimensiones de un bloque rectangular, calcule las diagonales de las tres caras diferentes. Muestre el valor de la mayor diagonal.

4. Lea un número de dos cifras. Determinar si la suma de ambas cifras es un número par o impar. Muestre un mensaje

5. Lea un número. Determine si es entero y múltiplo de 7

6. Lea la cantidad de Kw que ha consumido una familia y el precio por Kw. Si la cantidad es mayor a 700, incremente el precio en 5% para el exceso de Kw sobre 700. Muestre el valor total a pagar.

7. Lea un valor de temperatura **t** y un código **p** que puede ser **1** o **2**. Si el código es **1** convierta la temperatura **t** de grados **f** a grados **c** con la fórmula **c=5/9(t-32).** Si el código es **2** convierta la temperatura **t** de grados **c** a **f** con la fórmula: **f=32+9t/5**, caso contrario muestre un mensaje de error.

8. Dadas las tres calificaciones de un estudiante, encuentre y muestre su calificación final, la cual es la suma de las dos mejores calificaciones.

9. Dadas las tres calificaciones de un estudiante, encuentre y muestre la calificación mas alta.

10. Dados los tres lados de un triángulo determine su tipo: Equilátero, Isòsceles, o Escaleno

11. Dadas la abscisa y ordenada de dos puntos, calcule su distancia al origen y determine cual de los dos puntos (primero o segundo) está más cerca del origen. La respuesta es un mensaje: 'Punto 1' o 'Punto 2'

Punto	Abscisa	Ordenada
1	a	b
2	c	d

Fórmula de la distancia del punto (x, y) al origen: $d = \sqrt{x^2 + y^2}$

12. Juan, Pedro y José trabajan en una empresa que paga semanalmente. Ingrese para cada uno los siguientes datos del trabajo semanal: horas trabajadas, salario por hora, y descuentos. Calcule el pago semanal que recibirá cada uno y determine cual de los tres recibirá el mayor pago semanal. **NO** considere el pago de horas extras.

13. Lea las dimensiones de un bloque rectangular (largo, ancho y altura del bloque), y el diámetro de un agujero. Determine si es posible que el bloque pueda pasar por el agujero. Sugerencia: Compare el diámetro del agujero con cada una de las diagonales del bloque. Si en alguna comparación el diámetro tiene un valor menor a la diagonal muestre el mensaje: 'Si pasa por el agujero', caso contrario muestre 'No pasa por el agujero'.

14. Un código de tres cifras debe cumplir la siguiente regla para que sea vàlido: La tercera cifra debe ser igual al mòdulo 10 del producto de las dos primeras cifras. Escriba un programa que lea un código y verifique si cumple la regla anterior. Muestre un mensaje correspondiente.

Ej. 384 es un código vàlido pues mod(3x8,10) es igual a 4

15. El número de pulsaciones que debe tener una persona por cada 10 segundos de ejercicio aeróbico se calcula con la fórmula:

Género femenino (1): número de pulsaciones = (220 - edad en años)/10
Género masculino (2): número de pulsaciones = (210 - edad en años)/10

Lea la edad y el género y muestre el número de pulsaciones.

16. El índice de masa corporal IMC de una persona se calcula con la fórmula $IMC=P/T^2$ en donde P es el peso en Kg. y T es la talla en metros.

Lea un valor de P y de T, calcule el IMC y muestre su estado según la siguiente tabla:

IMC	**Estado**
Menos de 18.5	Desnutrido
[18.5 a 25.5]	Peso Normal
Más de 25.5	Sobrepeso

17. Otro reporte de salud muestra una tabla diferente del índice de masa corporal IMC de una persona que se calcula con la fórmula $IMC=P/T^2$ en donde P es el peso en Kg. y T es la talla en metros.

Lea un valor de P y de T, calcule el IMC y muestre su estado según la siguiente tabla:

IMC	**Estado**
20-24.9	Normal
25-29.9	Sobrepeso
30-34.9	Obesidad Grado 1
35-39.9	Obesidad Grado 2
Mayor a 40	Obesidad Grado 3

18. Lea la cantidad de Kw que ha consumido una familia y el precio por Kw. Si la cantidad es mayor a 700, incremente el precio en 5% para el exceso de Kw sobre 700. Muestre el valor total a pagar.

19. En un concurso hay tres jueces. La opinión del juez es 1 si está a favor y 0 si está en contra. Para que un participante pueda continuar en el concurso debe tener al menos dos votos favorables. Escriba un algoritmo que lea los votos de los tres jueces y muestre el resultado mediante un mensaje: CONTINUA o ELIMINADO. No sume votos. Debe compararlos.

20. Dadas las dimensiones de un bloque rectangular, calcule y muestre el área de la cara de mayor dimensión.

21. Desarrolle un algoritmo que lea un dato con la cantidad de días. El resultado debe ser el equivalente en meses, semanas y días sobrantes. Suponga que cada mes tiene treinta días. Ejemplo. Si el dato es 175 el resultado será 5 meses, 3 semanas y 4 días

22. Se conocen tres de los cuatro números de una matriz cuadrada de tamaño 2. Lea estos tres números y determine cual debe ser el cuarto número para que el determinante de la matriz sea igual a 0.

23. Lea un número **x** y los números **a, b**. Suponga que **a<b**. y que $x \neq a$, $x \neq b$. Determine en que lugar se encuentra el número **x**, antes de **a**, entre **a** y **b** o después de **b**. Muestre un mensaje.

24. Lea las tres calificaciones que obtuvo un estudiante en una materia. No suponga que estos tres números están ordenados. Describa como ordenarlos en forma ascendente y muestre los números ordenados

25. Lea los números de matrícula de tres estudiantes que toman la materia A y los números de matrícula de tres estudiantes que toman la materia B. Encuentre cuantos estudiantes que toman la materia A, también toman la materia B.

26. Un almacén ofrece un producto con descuento según la siguiente tabla:

Kilos comprados	Precio por kilo
Menos de 3	$2.4
[3 a 6)	$2.3
[6 a 10)	$2.1
Más de 10	$1.85

Lea la cantidad comprada y determine cuanto debe pagar.

27. Modifique el siguiente algoritmo de tal manera que los conectores lógicos (**$\wedge$, $\vee$, $\neg$)** sean eliminados con instrucciones que contengan solamente condiciones simples

1 Leer a, b, c
2 x $\leftarrow$ 0
3 y $\leftarrow$ 0
4 Si $\neg$(a<2 $\wedge$ b>1) $\vee$ (c>3)
5 x $\leftarrow$ a + b
6 Sino
7 x $\leftarrow$ b-c
8 Fin
9 Mostrar x, y

28. Considere el siguiente algoritmo

1 Leer a, b, c
2 Si a<b salte a la línea 4
3 Salte a la línea 5
4 Si b>c salte a la línea 6
5 Salte a la línea 7
6 Mostrar a
7 Mostrar b
8 Mostrar c

a) Construya un diagrama de flujo ordenado y simplificado que sea equivalente al algoritmo propuesto.

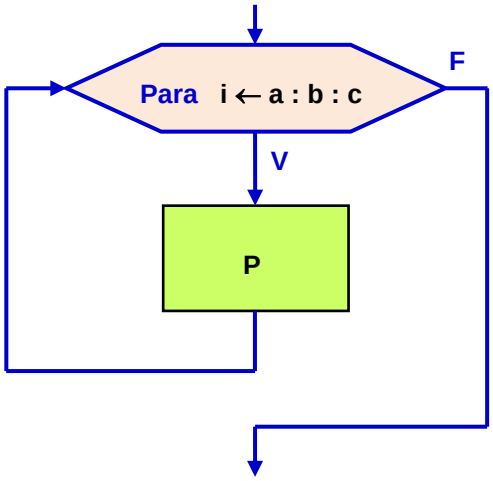
b) Interprete el diagrama de flujo y codifíquelo en notación MATLAB

4.4 Ciclos

Estas estructuras de control se usan para describir la ejecución repetida de un bloque de instrucciones..

Hay dos formas comunes que se usan en la construcción de algoritmos.

4.4.1 Ejecución repetida de un bloque mediante un conteo de ciclos



Para usar esta estructura de control es necesario especificar el conteo de ciclos indicando el valor inicial del conteo, el incremento y el valor final. Al ejecutarse la estructura el valor de la variable del conteo cambiará siguiendo la especificacion establecida.

Al entrar a esta estructura, se inicia el conteo, si la variable del conteo no excede al valor final, se ejecuta el bloque, regresa nuevamente al inicio del ciclo y se incrementa la variable del conteo. Mientras el valor del conteo no exceda al valor final, el bloque será nuevamente ejecutado. Cuando el conteo exceda al valor final, la ejecución continuará después del bloque.

El conteo de ciclos se lo puede definir especificando el valor inicial, el valor final y el incremento, con la siguiente sintaxis:

i ← a : b: c

En donde

i: Variable que toma los valores del conteo
a: Valor inicial del conteo
b: Valor con el que se incrementa el conteo
c: Valor final del conteo

Si no se escribe el valor del incremento, se supondrá que es **1**.

i ← a : c

Repetición de un bloque mediante un conteo de ciclos en MATLAB

```
for   i = a : b: c
      Bloque de Instrucciones que se repiten
end
```

Igualmente, si el incremento es 1 se puede abreviar el conteo a **i = a : c**

Ejemplo. Describa en notación algorítmica una solución al siguiente problema.

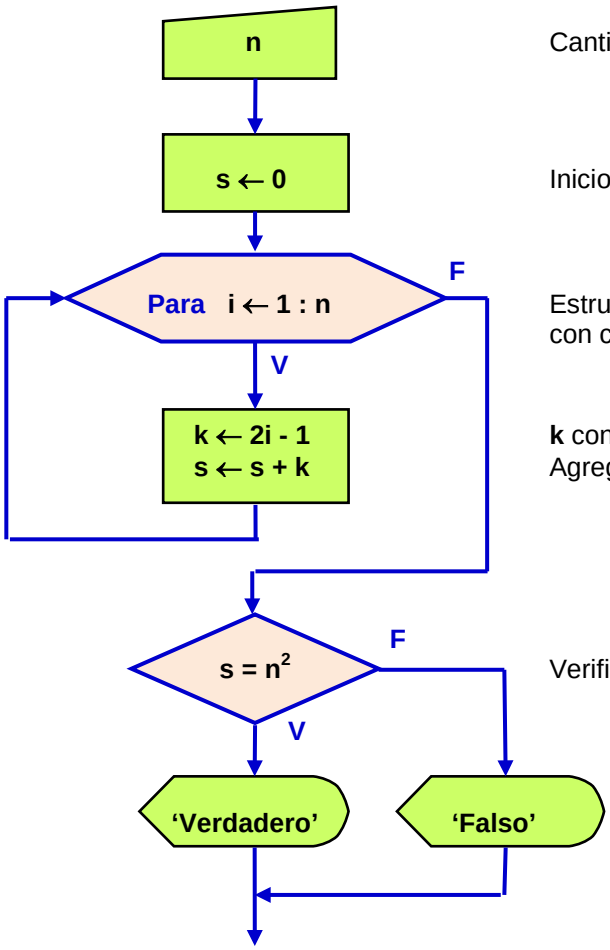
Dado un entero positivo **n**, se desea verificar que la suma de los primeros **n** números impares es igual a $\mathbf{n^2}$

Ej. n = 5: $\mathbf{1 + 3 + 5 + 7 + 9 = 5^2}$

Algoritmo: Suma de impares

Variables

n: Cantidad de números impares
k: Cada número impar
s: Suma de impares
i: Conteo de ciclos



Prueba. Realice una prueba del algoritmo anterior. Ingrese un dato desde fuera del bloque y registre los cambios en el contenido de las variables.

Prueba	n	Ciclo i	Impar k	s	Salida
	5			0	
		1	1	1	
		2	3	4	
		3	5	9	
		4	7	16	
		5	9	25	'Verdadero'

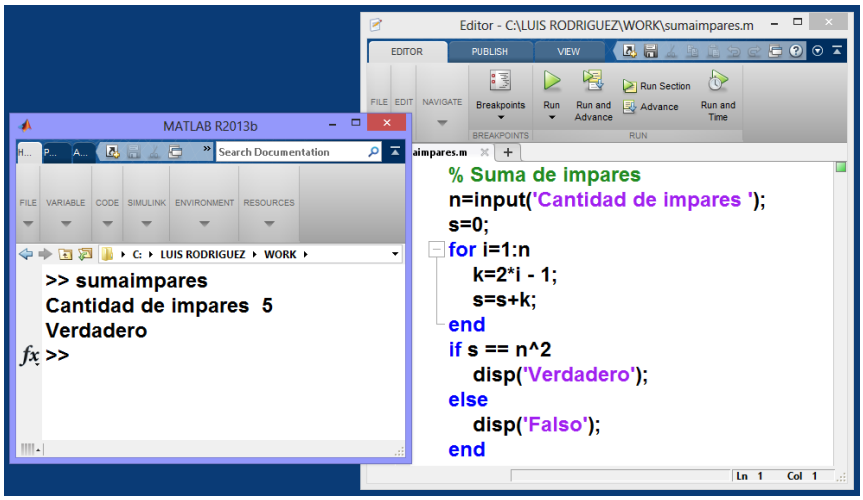
Verifica que el resultado es 'Verdadero'.

Note que esto no constituye una demostración matemática.

Lenguaje MATLAB

```
% Suma de impares
n=input('Ingrese la cantidad de impares ');
s=0;
for i=1:n
   k=2*i - 1;
   s=s+k;
end
if s == n^2
   disp('Verdadero');
else
   disp('Falso');
end
```

Interacción con MATLAB



Práctica en MATLAB con repeticiones con conteo de ciclos

Ejemplo. Escriba un programa que muestre 4 veces un nombre

Lenguaje MATLAB

```
for  i = 1: 4
   disp('Simón Bolívar');
end
```

Interacción

Simón Bolívar
Simón Bolívar
Simón Bolívar
Simón Bolívar

Ejemplo. Liste los números naturales entre **1** y **n**, siendo **n** un dato

Lenguaje MATLAB

```
n=input('Ingrese el valor final ');
for  i = 1: n
   disp(i);
end
```

Interacción

Ingrese el valor final 5
1
2
3
4
5

Se puede especificar otro valor para el incremento en el conteo:

Ejemplo. Listar los números naturales impares entre 1 y n

Lenguaje MATLAB

```
n=input('Ingrese el valor final ');
for  i= 1: 2: n
   disp(i);
end
```

Interacción

Ingrese el valor final 5
1
3
5

La secuencia de la variable de control de repeticiones puede incluir valores decimales.

Ejemplo: 0, 0.1, 0.2, 0.3, …, 2.0

```
for  i = 0: 0.1: 2

end
```

Puede ser decreciente.

Ejemplo: 5, 4, 3, 2, 1

```
for  i = 5: -1: 1

end
```

Puede ser una lista arbitraria.

Ejemplo: 7, 4, 5, 6, 8, 2

```
for  i = [7, 4, 5, 6, 8, 2]

end
```

Ejemplo. Dado un número entero, liste las raíces cuadradas de los enteros impares entre 1 y el dato dado.

```
n=input('Ingrese el valor final ');
for  x = 1: 2: n
   r=sqrt(x);
   disp([x,r]);
end
```

```
Ingrese el valor final  9
    1    1
    3    1.7321
    5    2.2361
    7    2.6458
    9    3
```

Ejemplo. Suma de los cuadrados de los primeros **n** números naturales

```
n=input('Ingrese el valor final ');
s=0;
for  i = 1: n
   c=i^2;
   s=s+c:
end
disp('La suma es');
disp(s);
```

```
Ingrese el valor final  5

La suma es
    55
```

Ejemplo. Muestre el promedio de **n** datos ingresados desde el teclado

```
n=input('Cantidad de datos ');
s=0;
for  i = 1: n
   x=input('Ingrese un dato ');
   s=s+x:
end
p=s/n;
disp('El promedio es');
disp(p);
```

```
Cantidad de datos  5
 Ingrese un dato  20
 Ingrese un dato  24
 Ingrese un dato  32
 Ingrese un dato  41
 Ingrese un dato  26

 El promedio es
   28.6
```

Ejemplo. Elija y muestre la mayor calificación de **n** datos ingresados desde el teclado

```
n=input('Cantidad de datos ');
m=0;
for  i = 1: n
   x=input('Ingrese un dato ');
   if x>m
     m=x:
   end
end
disp('El mayor valor es');
disp(m);
```

```
Cantidad de datos  5
Ingrese un dato  70
Ingrese un dato  60
Ingrese un dato  80
Ingrese un dato  65
Ingrese un dato  72

El mayor valor es
  80
```

Ejemplo. Elija y muestre la menor calificación de **n** datos ingresados desde el teclado

```
n=input('Cantidad de datos ');
m=100;
for  i = 1: n
   x=input('Ingrese un dato ');
   if x<m
     m=x:
   end
end
disp('El menor valor es');
disp(m);
```

Cantidad de datos 5
Ingrese un dato 70
Ingrese un dato 60
Ingrese un dato 80
Ingrese un dato 65
Ingrese un dato 72

El menor valor es
60

Ejemplo. Dado un número entero positivo **n**, sume los cuadrados de los enteros entre **1** y **n**. Verifique si esta suma es múltiplo del dato dado **n**.

Variables

n: número dado
x: cada número entero
s: suma de los cuadrados

```
%Prueba de múltiplos
n=input('Ingrese el valor final ');
s=0;
for x = 1: n
   s=s+x^2;
end
if mod(s, n) == 0
   disp('La suma es múltiplo');
else
   disp('La suma no es múltiplo ');
end
```

Ejemplo. Dado un número entero positivo, encuentre todos sus divisores enteros exactos.

n: dato
x: cada número entero

```
%Encontrar los divisores de un número
n=input('Ingrese el dato ');
for x = 1: n
   if mod(n,x) == 0
      disp(x);
   end
end
```

Ingrese el dato 20
1
2
4
5
10
20

Ejemplo. Dado un número entero, cuente sus divisores enteros exactos.
n: dato
x: cada número entero
c: cantidad de divisores

```
%Conteo de los divisores de un número
n=input('Ingrese el dato ');
c=0;
for x = 1: n
   if mod(n,x) == 0
      c=c+1;
   end
end
disp(c);
```

Ejemplo. Dado un número entero, encuentre cuantos divisores enteros exactos tiene y luego determine si es primo

```
%Determinar si un número es primo
n=input('Ingrese el dato ');
c=0;
for x = 1: n
   if mod(n,x) == 0
      c=c+1;
   end
end
if c > 2
   disp('No es primo ');
else
   disp('Si es primo ');
end
```

Ingrese el dato 1307
Si es primo

4.4.2 Números aleatorios en MATLAB

Los números aleatorios son números cuyo valor no se puede predecir. Por ejemplo, al lanzar un dado, no podemos afirmar cual resultado se obtendra entre los posibles seis números

Los lenguajes computacionales tiene instrucciones especiales para generar números aleatorios que se usan en muchas aplicaciones de interés

La función **RAND** de MATLAB genera un número aleatorio real en el rango **[0, 1)**

Ejemplo.

```
>> rand
```

Se obtendrá un resultado entre **0** y **0.9999…**

Se puede modificar el rango de la función **RAND** para llevarlo a otro rango de interés

Ejemplo. Obtener un entero entre 0 y 9

```
>> fix(10*rand)
```

Ejemplo. Obtener un entero entre 1 y 10

```
>> fix(10*rand)+1
```

Ejemplo. Obtener un número aleatorio entero entre 1 y 6

```
>> fix(6*rand)+1
```

6*rand produce un valor entre **0 y 5.9999…**
fix(6*rand) produce un resultado entero entre **0 y 5**
fix(6*rand)+1 produce un resultado entero entre **1 y 6**

Ejemplo. Lista de números aleatorios de una cifra

n: cantidad de números
x: número aleatorio de una cifra

```
%Lista de números aleatorios
n=input('Cuantos números ');
for i=1:n
   x=fix(10*rand);
   disp(x);
end
```

Ejemplo. Simule **n** lanzamientos de un dado. Muestre cuantas veces se obtuvo el 3.

n: dato
i: conteo de lanzamientos
x: cada número aleatorio
c: cantidad de veces que se obtuvo el 3

```
%Simular lanzamientos de un dado
n=input('Ingrese el dato ');
c=0;
for i = 1: n
   x=fix(6*rand)+1;
   if x==3
      c=c+1;
   end
end
disp(c);
```

NOTA: pruebe con un valor grande de **n** que la respuesta es aproximadamente **1/6** de **n**
Este resultado confirma lo que indica la Teoría de la Probabilidad.

Ejemplo. Simule **n** lanzamientos de dos dados. Muestre cuantas veces los dos dados tuvieron el mismo resultado.

n: dato
i: conteo de lanzamientos
a,b: números aleatorios enteros entre 1 y 6
c: cantidad de veces en las que **a** fue igual a **b**

```
%Lanzamientos de dos dados
n=input('Cantidad de lanzamientos ');
c=0;
for i=1:n
   a=fix(6*rand)+1;
   b=fix(6*rand)+1;
   if a==b
      c=c+1;
   end
end
disp('Cantidad de repetidos');
disp(c);
```

Interacción

```
Cantidad de lanzamientos 400
Cantidad de repetidos
75
```

Ejemplo. Simular n intentos de un juego con un dado, con la siguientes reglas:

6: gana 4 dólares
3: gana 1 dólar
1: siga jugando
2,4 o 5: pierde 2 dólares

n: dato
i: conteo de lanzamientos
x: cada número aleatorio
s: cantidad acumulada de dinero

```
%Simulación de un juego
n=input('cuantos intentos ');
s=0;
for i=1:n
   x=fix(6*rand)+1;
   switch x
      case 6
         s=s+4;
      case 3
         s=s+1;
      case {2, 4, 5}
         s=s-2;
   end
end
disp(s);
```

Observe que si el número de intentos es grande, normalmente se obtendrá un valor negativo. Esto significa que no es una buena idea jugar.

Ejemplo. Simule **n** lanzamientos de un dardo en un cuadrado de 1 m de lado. Determine cuantos intentos cayeron dentro de un círculo inscrito en el cuadrado.

n: dato
i: conteo de lanzamientos
x,y: coordenadas para cada lanzamiento (números aleatorios)
c: cantidad de veces que están dentro del círculo de radio 1

```
%Puntos aleatorios dentro de un círculo
n=input('¿Cuántos intentos? ');
c=0;
for i=1:n
   x=rand;
   y=rand;
   if x^2 + y^2 <= 1
      c=c+1;
   end
end
disp('Dentro del círculo');
disp(c);
```

Interacción

```
¿Cuántos intentos? 100
Dentro del círculo
78
```

Ejemplo. Dado un conjunto de enteros numerados **1** a **n**, elegir una muestra aleatoria de **m** números, **m≤n**

```
%Muestra aleatoria (con repeticiones)
n = input('Ingrese dato de la población ');
m = input('Ingrese tamaño de la muestra ');
for i=1: m
   x=fix(n*rand)+1;
   disp(x);
end
```

Interacción

```
Ingrese dato de la población  15
Ingrese tamaño de la muestra  5
10
12
14
12
 3
```

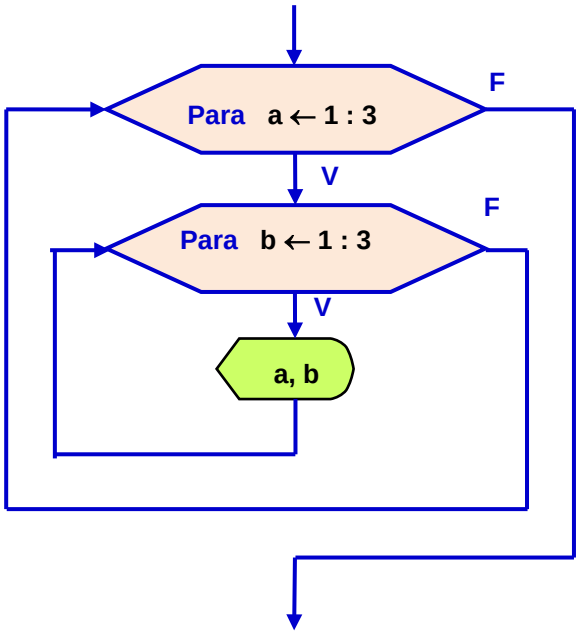
Note que pueden salir valores repetidos. Posteriormente se describirá una técnica para que no se incluyan elementos repetidos.

4.4.3 Ciclos anidados

Algunos algoritmos requieren ciclos dentro de otros ciclos. La regla establece que para cada valor que toma la variable del ciclo externo, el ciclo interior se realiza completamente.

Ejemplo. Liste todas las parejas de números con valores enteros del 1 al 3

a,b: números



Ciclo externo para a = 1, 2, 3

Ciclo interno para b = 1, 2, 3

Prueba. Realice una prueba del algoritmo anterior. Ingrese un dato desde fuera del bloque y registre los cambios en el contenido de las variables.

Prueba	a	b	Salida
	1	1	1, 1
		2	1, 2
		3	1, 3
	2	1	2, 1
		2	2, 2
		3	2, 3
	3	1	3, 1
		2	3, 2
		3	3, 3

Lenguaje MATLAB

```
%Parejas de números
for a = 1: 3
   for b = 1: 3
      disp([a, b]);
   end
end
```

Interacción

```
1    1
1    2
```

```
1    3
2    1
2    2
2    3
3    1
3    2
3    3
```

El ciclo interno se realiza completamente para cada valor de la variable del ciclo externo. En el ejemplo anterior en total se realizan 3x3 = 9 ciclos

Ejemplo. Liste todas las parejas de números con valores enteros del 1 al 3 pero sin que hayan parejas repetidas

a,b: números

```
%Parejas de números sin repeticiones
for a = 1: 3
   for b = a: 3
       disp([a, b]);
   end
end
```

Interacción

```
1    1
1    2
1    3
2    2
2    3
3    3
```

Ejemplo. Liste todos los números de dos cifras que se pueden hacer con los dígitos del 1 al 3

a, b: dígitos entre 1 y 3
n: número

```
for a = 1: 3
   for b = 1: 3
      n = 10*a + b;
      disp(n);
   end
end
```

Interacción

```
11
12
13
21
22
23
31
32
33
```

Ejemplo. Liste todas las ternas (a, b, c) de números enteros entre 1 y 20 que cumplen la propiedad pitagórica: $a^2 + b^2 = c^2$

a, b, c: números entre 1 y 20

```
%Ternas pitagóricas
for a = 1: 20
   for b = 1: 20
      for c = 1: 20
         if a^2 + b^2 == c^2
               disp([a, b, c]);
         end
       end
   end
end
```

Interacción

```
 3    4    5
 4    3    5
 5   12   13
 6    8   10
 8    6   10
 8   15   17
 9   12   15
12    5   13
12    9   15
12   16   20
15    8   17
16   12   20
```

Ejemplo. Liste todas las ternas (a, b, c) de números enteros entre 1 y 20 que cumplen la propiedad pitagórica: $a^2 + b^2 = c^2$ pero sin ternas repetidas

a, b, c: números entre 1 y 20

```
%Ternas pitagóricas sin repeticiones
for a = 1: 20
   for b = a: 20
    for c = b: 20
       if a^2 + b^2 == c^2
             disp([a, b, c]);
       end
       end
   end
end
```

Interacción

```
 3    4    5
 5   12   13
 6    8   10
 8   15   17
 9   12   15
12   16   20
```

Pregunta: ¿Cuantas repeticiones se realizan en total en cada uno de los dos últimos algoritmos?

Nota. Entre dos algoritmos que resuelven el mismo problema, aquel con menor número de ciclos es más eficiente. Este es un criterio para elegir algoritmos

Ejemplo. Determine si la proposición **$(a \wedge b) \Rightarrow (a \vee c)$** es una tautología

Se puede usar una expresión equivalente: $\neg((a \wedge b)) \vee (a \vee c)$

Los valores que se asignan a las variables serán **0** o **1** que se interpretan como valores lógicos.

```
for a=0:1
   for b=0:1
      for c=0:1
         p=~((a&b))|(a|c);
         disp([a,b,c,p]);
      end
   end
end
```

Interacción

```
0   0   0   1
0   0   1   1
0   1   0   1
0   1   1   1
1   0   0   1
1   0   1   1
1   1   0   1
1   1   1   1
```

El resultado muestra que es una **tautología** pues la cuarta columna (p) contiene solamente **1's**
Se puede incluir un conteo dentro de los ciclos para que el programa entregue el mensaje.

4.4.4 La instrucción BREAK

Se utiliza para interrumpir una repetición y salir sin completar la cantidad de ciclos especificada

Ejemplo. Simular lanzamientos de un dado. Determinar la cantidad de lanzamientos hasta que salga el 5.

```
n=input('Cantidad máxima de lanzamientos ');
for i=1:n
   x=fix(rand*6)+1;
   disp(x)
   if x==5
      disp('Lanzamiento en el cual salió el 5');
      disp(i);
      break;
   end
end
```

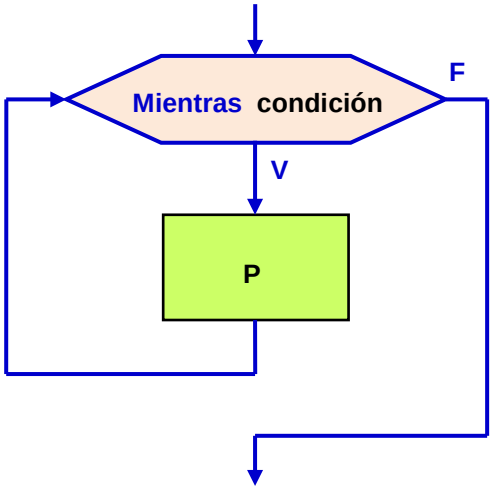
Interacción

```
Cantidad máxima de lanzamientos  200
   4
   3
   6
   4
   2
   1
   5

Lanzamiento en el cual salió el 5
   7
```

Esta solución no luce muy natural pues la salida normalmente debería realizarse cuando se agotan los valores de la lista **for**. La instrucción **break** permite salir del ciclo antes de finalizar el conteo de repeticiones. Adicionalmente, no está garantizado que se obtenga el número **5** dentro de la cantidad máxima de lanzamientos especificada.

Hay una manera más natural de realizar estos algoritmos mediante una instrucción que condiciona la salida del ciclo sin usar un conteo, es la instrucción de repetición **while**

4.4.5 Ejecución repetida de un bloque mediante una condición



Al entrar a esta estructura, se evalúa la condición. Si el resultado es verdadero (**V**) se ejecutará el bloque y regresará nuevamente a evaluar la condición. Mientras la condición mantenga el valor verdadero, el bloque de instrucciones seguirá ejecutándose. Esto significa que es necesario que en algún ciclo la condición tenga el resultado falso (**F**) para salir de la estructura y continuar la ejecución después del bloque.

La condición es cualquier expresión cuyo resultado puede ser únicamente verdadero (**V**) o falso (**F**). Puede incluir operadores para comparar el contenido de variables y también se pueden usar los conectores de la lógica matemática.

Lenguaje MATLAB

```
while   condición

        Bloque de instrucciones que se repiten

end
```

Ejemplo. Describa en notación MATLAB la repetición de un bloque de instrucciones mientras la variable **x** tenga un valor menor a **10**

```
while   x<10

        Bloque de instrucciones que se repiten

end
```

Nota: antes de la instrucción **while,** la variable **x** debe haber sido asignada. Además, dentro del bloque debe cambiar su valor de tal manera que en algún ciclo **x** no sea menor a **10** y la repetición pueda terminar.

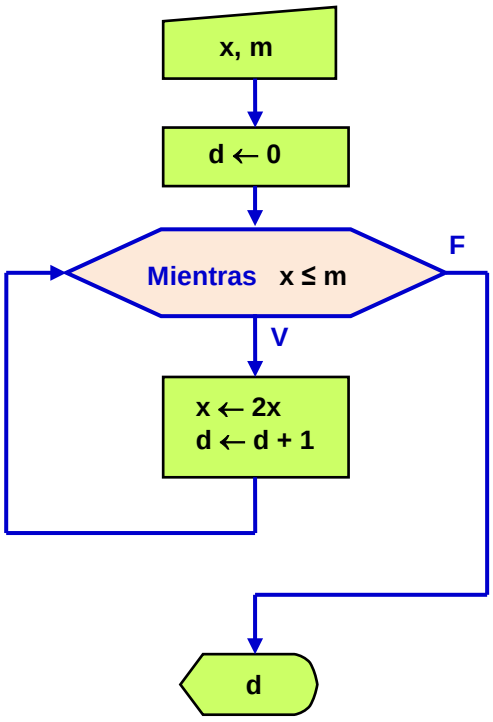
Ejemplo. Describa en notación algorítmica una solución para el siguiente problema usando la estructura de ciclos condicionada.

Dada una cantidad inicial de bacterias, determine en que día la cantidad de bacterias excede a un valor máximo sabiendo que cada día se duplica esta cantidad.

Algoritmo: Crecimiento de la cantidad de bacterias

Variables

x: Cantidad inicial de bacterias
m: Cantidad máxima de bacterias
d: Día



Ingreso de los datos

Iniciar el día en cero

El ciclo se repetirá mientras la cantidad de bacterias no exceda al máximo.

Dentro del bloque se modifica la cantidad de bacterias y el día.

Mostrar el valor final del día

Nota. Se debe usar el ciclo condicionado pues supondremos que no se puede anticipar el número de repeticiones necesarias para que la cantidad de bacterias exceda al valor máximo. (Con las matemáticas si se pudiera calcular directamente este número)

Prueba. Realice una prueba del algoritmo anterior. Ingrese los datos desde fuera del bloque y registre los cambios en el contenido de las variables.

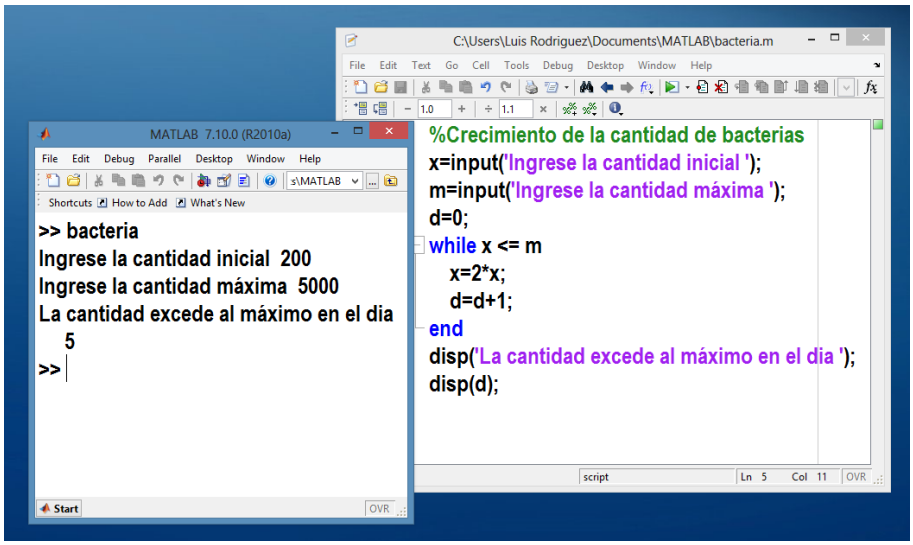
Prueba	x	m	d	Salida
	200	5000	0	
	400		1	
	800		2	
	1600		3	
	3200		4	
	6400		5	5

Se puede verificar que es el resultado esperado

Lenguaje MATLAB

```
%Crecimiento de la cantidad de bacterias
x=input('Ingrese la cantidad inicial ');
m=input('Ingrese la cantidad máxima ');
d=0;
while x <= m
   x=2*x;
   d=d+1;
end
disp('La cantidad excede al máximo en el dia ');
disp(d);
```

Interacción con MATLAB



Ejemplo. Simular lanzamientos de un dado. Muestre el resultado en cada intento hasta que salga el 5.

x: resultado del dado en cada lanzamiento. Inicialmente un valor nulo para entrar al ciclo

```
%Simular el lanzamiento de un dado hasta que sale un número especificado
x=0;
while x~=5
   x=fix(rand*6)+1;
   disp(x)
end
```

Interacción

1
3
6
4
2
5

En el ejemplo se muestra que el uso de la estructura **while** es natural cuando no se conoce la cantidad de repeticiones que deben realizarse hasta salir del ciclo.

Si es de interés conocer el número de repeticiones realizadas se debe incluir una variable para efectuar el conteo como se muestra en el siguiente ejemplo.

Ejemplo. Simular lanzamientos de un dado. Determinar la cantidad de lanzamientos que se realizaron hasta que se obtuvo el número 5.

x: resultado del dado en cada lanzamiento. Inicialmente valor nulo para entrar al ciclo
c: conteo de repeticiones

```
%Conteo de lanzamientos de un dado
c=0;
x=0;
while x~=5
   x=fix(rand*6)+1;
   disp(x)
   c=c+1
end
disp('Cantidad de lanzamientos hasta que salió el 5');
disp(c);
```

Interacción

2
4
2
6
1
4
5
Cantidad de lanzamientos hasta que salió el 5
7

Compare esta solución con la propuesta usando **for** y **break**. Esta solución es más natural

Todos los algoritmos de repetición pueden ser descritos con la instrucción **while**, pero si la secuencia de valores es un conteo simple, normalmente es más claro usar la instrucción **for**

Ejemplo. Suma de los cuadrados de los primero n números naturales

Con la instrucción for:

```
%Suma de cuadrados con FOR
n=input('Ingrese el valor final ');
s=0;
for  i = 1: n
   c=i^2;
   s=s+c:
end
disp('La suma es');
disp(s);
```

Interacción

```
Ingrese el valor final  5
La suma es
  55
```

Ejemplo. Suma de los cuadrados de los primero n números naturales

Con la instrucción while

```
%Suma de cuadrados con WHILE
n=input('Ingrese el valor final ');
s=0;
i=1;
while i <= n
   c=i^2;
   s=s+c;
   i=i+1;
end
disp('La suma es');
disp(s);
```

Interacción

```
Ingrese el valor final  5

La suma es
  55
```

La instrucción **for** luce más adecuada para este ejemplo, pero en los siguientes ejemplos, es mucho más natural usar la instrucción **while**

Ejemplo. Dado un número entero, genere la secuencia con la siguiente regla. Esta secuencia se denomina de Ulam. Finalmente se llega al número 1

$$x = \begin{cases} x/2, & x \text{ par} \\ 3x+1, & x \text{ impar} \end{cases}$$

x: número dado

```
%Secuencia de Ulam
x=input('Ingrese el dato ');
while x ~= 1
   if mod(x,2) == 0
      x=x/2;
   else
      x=3*x+1;
   end
   disp(x);
end
```

Interacción

```
Ingrese el dato  20
10
5
16
8
4
2
1
```

Ejemplo. Dado un entero positivo encuentre sus factores primos

Para probar: Si n es 72, sus factores primos son: 2, 2, 2, 3, 3 pues su producto es 72

n: dato
p: cada factor primo

```
%Factores primos de un número entero
n = input('Ingrese el dato ');
p = 2;
while p<=n
   while mod(n, p) == 0                    %Encontar un divisor
         disp(p);
         n=fix(n/p);                       %Eliminar el divisor
   end
   p=p+1;
end
```

Ejemplo. Se dice que dos números son “amigables” si el primero es la suma de los divisores del segundo y viceversa.

Los números 220 y 284 son amigables pues se tiene que

Los divisores de 220 son: {1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110} y la suma es es 284.
Los divisores de 284 son: {1, 2, 4, 71, 142} y la suma es 220.

Escriba un programa que busque los números amigables entre 1 y 1000.

```
for a=1:1000
    s=0;
    for d=1:a-1                      % suma de los divisores menores de  a
        if mod(a,d)==0
            s=s+d;
        end
    end
    for b=1:1000
        t=0;
        for d=1:b-1                  % suma de los divisores menores de  b
            if mod(b,d)==0           % para cada suma de divisores de  a
               t=t+d;
            end
        end
        if s==b & t==a & a~=b        % chequeo de la condición
            disp([a,b]);
        end
     end
 end
```

Interacción

```
  220   284
  284   220
```

4.4.6 Introducción a validación de datos

En todos los ejemplos anteriores se ha supuesto que los datos que ingresan no tienen algún error. En la realidad al usar un programa es frecuente que se introduzcan datos incorrectos, por lo que el programa debe realizar alguna validación para evitar que se produzca una interrupción brusca de la ejecución o que los resultados sean incorrectos.

Ejemplo. Dado un número entero, determine cuantas cifras tiene.

n: dato
c: cantidad de cifras

```
%Conteo de las cifras de un número
n=input('Ingrese un entero positivo ');
c=0;
while n>0
   n=fix(n/10);
   c=c+1;
end
disp(c);
```

Interacción

```
Ingrese un entero positivo  345721
Cantidad de cifras
  6

Ingrese un entero positivo  -345
Cantidad de cifras
  0
```

Si el dato ingresado es negativo o tiene decimales el resultado será incorrecto como se observa en el ejemplo anterior. Esta situación se puede prevenir con un chequeo en el ingreso:

Ejemplo. Dado un número entero, determine cuantas cifras tiene. Valide que el dato de entrada sea positivo y entero

n: dato
c: cantidad de cifras

```
%Conteo de las cifras de un número
n=input('Ingrese un entero positivo ');
if n>0 & n==fix(n)                    % Verificar que el dato sea positivo y entero
   c=0;
   while n>0
      n=fix(n/10);
      c=c+1;
   end
   disp('Cantidad de cifras');
   disp(c);
else
   disp('Error');
end
```

Interacción

```
Ingrese un entero positivo  -345
  Error
```

Una mejor opción consiste en validar el dato y si tiene algún error permitir que se pueda corregirlo sin terminar el programa, como se muestra en el ejemplo anterior modificado:

Ejemplo. Dado un número entero, determine cuantas cifras tiene. Valide que el dato de entrada sea positivo y entero, con la posibilidad de corregir el error:

n: dato
c: cantidad de cifras

```
%Conteo de las cifras de un número
n=-1;
while n<0 | n~=fix(n)
    n=input('Ingrese un entero positivo ');
end
c=0;
while n>0
    n=fix(n/10);
    c=c+1;
end
disp('Cantidad de cifras');
disp(c);
```

Interacción

```
Ingrese un entero positivo -234
Ingrese un entero positivo 76.35
Ingrese un entero positivo 43216
Cantidad de cifras
    5
```

Note el valor inicial asignado a la variable **n**, para comenzar el ciclo **while**

4.4.7 Ejercicios de programación con ciclos

1.- Calcule el promedio, el menor valor y el mayor valor de los pesos de **n** paquetes en una bodega. Estos datos ingresan uno a la vez dentro de un ciclo. **n** es un dato ingresado al inicio.

2.- Clasifique los pesos de los **n** objetos de una bodega en tres grupos: menor a 10 Kg., entre 10 y 20 Kg., mas de 20 Kg. Los datos ingresan uno a la vez en un ciclo.

3. Determine la cantidad de términos que deben sumarse de la serie $\mathbf{1^2 + 2^2 + 3^3 + 4^4 + ...}$ para que el valor de la suma sea mayor a un número **x** ingresado al inicio.

4.- Dado dos números enteros **a, b**, determine su máximo común divisor **m**.
Ejemplo: **a** = 36, **b** = 45 entonces **m** = 9

5. Calcule un valor aproximado para la constante π usando la siguiente expresión:

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 ...$$

La cantidad de términos es un dato que debe ser ingresado al inicio del algoritmo.

6.- Lea los votos de **n** personas. Cada voto es un número **1, 2, o 3** correspondiente a tres candidatos. Si el dato es 0 es un voto en blanco. Si es otro número es un voto nulo. Determine el total de votos de cada candidato y el total de votos blancos y nulos.

7.- Lea las coordenadas de **u, v** de la ubicación de una fábrica y las coordenada **x, y** de **n** sitios de distribución. Encuentre cual es la distancia del sitio más alejado de la fábrica

8.- Encuentre el mayor valor de la función **f(x)=sen(x)+ln(x)**, para los valores:
x = 1.0, 1.1, 1.2, 1.3, ..., 4

9.- Se tienen una lista de las coordenadas **x, y** de **n** puntos en un plano. Lea sucesivamente las coordenadas de cada punto y acumule las distancias del punto al origen. Muestre la distancia total acumulada.

10.- Determine la suma de los términos de la serie $\mathbf{1^3 + 2^3 + 3^3 + ... + n^3}$ en donde **n** es un número natural

11.- Determine la suma de los n primeros números de la serie:
1, 1, 2, 3, 5, 8, 13, 21, ...
en la cual cada término, a partir del tercero, se obtiene sumando los dos términos anteriores

12.- El inventor del juego del ajedréz pidió a su rey que como recompensa le diera por la primera casilla 2 granos de trigo, por la segunda, 4 granos, por la tercera 8, por la cuarta 16, y así sucesivamente hasta llegar a la casilla 64. El rey aceptó. Suponga que cada Kg. de trigo consta de 20000 granos de trigo. Si cada tonelada tiene 1000 Kg. describa un algoritmo para calcular la cantidad de toneladas de trigo que se hubiesen necesitado.

En el ciclo describa la suma $\mathbf{2^1 + 2^2 + 2^3 + 2^4 + ... + 2^{64}}$

13.- Una empresa compra una máquina en $20000 pagando cuotas anuales durante cinco años. La siguiente fórmula relaciona el costo de la máquina **P**, el pago anual **A**, el número de años **n** y el interés anual **r**:

$$\mathbf{A = P\frac{r(1+r)^n}{(1+r)^n - 1}}$$

Escriba un programa que permita calcular el valor de **P** para valores de **r = 0.01,0.02, …, 0.1**

14.- Una persona tiene una lista con los precios de n artículos y dispone de una cierta cantidad de dinero. Los artículos son identificados con la numeración natural. Escriba un programa para leer estos datos y obtener los siguientes resultados
a) Muestre la identificación de los artículos que puede comprar
b) Para cada artículo cuyo precio es menor que la cantidad de dinero disponible, determine la cantidad que puede comprar.

15.- La plataforma de un transporte tiene capacidad para llevar hasta m kilos. Se tiene una lista ordenada en forma creciente con el peso de n paquetes. Determine cuantos paquetes pueden ser transportados. La elección debe hacerse comenzando con los paquetes de menor peso.

16.- En un supermercado se hace una promoción, mediante la cual el cliente obtiene un descuento dependiendo de un número de una cifra que se escoge al azar. Si el numero escogido es menor que 7 el descuento es del 5% sobre el total de la compra, si es mayor o igual a 7 el descuento es del 10%. Lea la cantidad de dinero. genere el número aleatorio y muestre cuanto dinero se le descuenta.

17.- Escriba un programa para simular la extracción de n bolas de una caja que contiene m bolas numeradas con los números naturales del 1 al m. Cada vez que se saca la bola se muestra el número y se la devuelve a la caja, por lo tanto pueden salir bolas repetidas.

18.- Escriba un programa que genere un número aleatorio con un valor entre 1 y 100 y que sea un número primo.

19.- Escriba un programa que muestre dos números aleatorios con valores enteros entre 1 y 100 tales que la suma sea un número primo.

20.- Lea un número par. Encuentre dos números al azar tales que la suma sea igual al dato dado.

21.- Lea un número par. Encuentre dos números al azar tales que sean primos y la suma sea igual al dato dado.

22.- Simule el siguiente juego entre tres ranas. Las ranas están al inicio de una pista de 20 m. En turnos cada rana realiza un salto. El salto es aleatorio y puede ser: a) Brinca y cae en el mismo lugar, b) Salta 0.5 m en la dirección correcta, c) Salta 1 m en la direccion correcta, d) Salta 0.5 m retrocediendo. Determine cual de las tres ranas llega primero a la meta.

23.- Realice la simulación de n intentos de lanzamientos de un dado con las siguientes reglas: si sale 6 gana $5. Si sale 1 gana $1. Si sale 2, 3, 4 o 5 pierde $2. Determine la cantidad acumulada al final del juego

24.- Dado un valor entero positivo **n** verifique que $\mathbf{1^3+2^3+3^3+\ldots+n^3 = (1+2+3+\ldots n)^2}$

25.- Escriba un programa que genere n parejas de número primos gemelos. Estos números primos tienen la propiedad que además de ser primos, la distancia en tre ellos es 2.
Ejemplo. 3 y 5, 5 y 7, 11 y 13, 17 y 19, etc

26.- En un juego se debe asignar a cada persona un número mágico que se obtiene con la siguiente regla: Se suman los dígitos de la fecha de nacimiento y se suman nuevamente los dígitos del resultado hasta obtener obtener un solo dígito, como en el siguiente ejemplo:

Fecha de Nacimiento: **28/11/1989**

28 + 11 + 1989 = 2028 $\Rightarrow$ 2 + 0 + 2 + 8 = 12 $\Rightarrow$ 1 + 2 = 3

Entonces el número buscado es **3**

Lea tres números: dia, mes, año y muestre el número mágico correspondiente

27. Analice el siguiente algoritmo. Escriba los resultados que se obtendrían si el dato que ingresa para **n** es **25**

```
n = input('Ingrese un dato ');
r = 0;
while n>0
   d = mod(n, 2);
   n = fix(n/2);
   r = 10*r + d;
   disp([d, n, r]);
end
```

28. Analice el siguiente algoritmo

1 Leer a, b
2 Salte a la línea 5
3 Mostrar x
4 Salte a la línea 12
5 x ← 0
6 Si a<5 salte a la línea 10
7 x ← x + a
8 Si x>b salte a la línea 11
9 Salte a la línea 7
10 x ← x + a – b
11 Salte a la línea 3
12 Fin

a) Construya un diagrama de flujo ordenado que sea equivalente al algoritmo propuesto.
b) Interprete el diagrama de flujo y codifíquelo en notación MATLAB

29. Analice el siguiente algoritmo

1 Leer a, b, c
2 r ← 0
3 Si a<b ∨ c<0 salte a la línea 8
4 Si b es par salte a la línea 6
5 r ← r + c
6 b ← b - 1
7 Salte a la línea 3
8 Mostrar r

a) Construya un diagrama de flujo ordenado que sea equivalente al algoritmo propuesto.
b) Interprete el diagrama de flujo y codifíquelo en notación MATLAB

30. Dibuje un diagrama de flujo para describir un algoritmo que resuelva el siguiente problema:

En la asamblea de un partido político hay dos posibles candidatos para inscribirlo en las elecciones de alcalde. Para elegir al candidato del partido, cada una de las **n** personas asistentes a la reunión entregan un voto. Se deben leer uno por uno los votos y determinar si alguno de los dos candidatos obtuvo más de la mitad de los votos. Este será el candidato.

31. Analice el siguiente programa que usa un ciclo **FOR**. Escriba un programa equivalente que produzca el mismo resultado, pero sustituyendo el ciclo **FOR** por un ciclo **WHILE** Debe definir una variable para conteo de repeticiones y la condición para salir del ciclo.

```
n = input('Ingrese un dato ');
s = 0;
for i = 1: n
   s = s + i^2;
end
disp(s);
```

32. Escriba un programa con un ciclo. Dentro del ciclo se generarán tres números aleatorios con valores enteros del 1 al 10. El programa deberá terminar cuando en alguna repetición, uno de los tres números sea igual al producto de los otros dos números. Muestre los números resultantes. Muestre también la cantidad de repeticiones que se realizaron.

33. El cuadrado de cualquier número terminado en 5 se lo puede formar como el producto: (decenas)(decenas+1) + 25.

Ej. $85^2 = 10(8)10(9) + 25 = 7225$
$475^2 = 10(47)10(48) + 25 = 225625$

Elabore un programa que verifique si se cumple esta regla con los números 5, 10, 15, 20, ..., **m**. Si no es verdad, muestre el primer número que no cumple esta regla.

34. El número EAN (European Article Number) usado en la identificación de productos consta de 13 dígitos: tres para el país, cuatro para la empresa, cinco para el producto y el dígito de control para detectar errores de digitación con la siguiente regla:

Comenzado por la izquierda hasta el decimo segundo dígito, multiplique el dígito por 1 si la posición es impar y por 3 si la posición es par. Sumar los resultados de los productos y restar de la decena superior. Este último resultado debe coincidir con el dígito de control

Escriba un programa que lea un número EAN, valide que tenga trece dígitos, calcule el dífito de control e informe el resultado.

Ej. EAN: 7 7 0 2 0 0 4 0 0 3 5 0 **8**
7x1 + 7x3 + 0x1 + 2x3 + 0x1 + 0x3 + 4x1 + 0x3 + 0x1 + 3x3 + 5x1 + 0x3 = 52

Decena superior: 60

60 – 52 = **8** Coincide con el dígito de control. El número EAN es correcto.

4.5 Programas que interactúan con un menú

Estructura básica para interactuar con un menú

1) El programa muestra las opciones disponibles para el usuario
2) El usuario elige una opción
3) El programa realiza la acción solicitada

Si la interacción está en un ciclo, el programa debe incluir una opción para salir. Para programar la realización de las acciones, es adecuado el uso de la instrucción **switch**.

Ejemplo. Conversión de Temperatura entre ºF y ºC. La siguiente fórmula permite convertir un valor de temperatura entre grados **farenheit** y grados **celcius**:

$$c = \frac{5}{9}(f - 32)$$

Escriba un programa con un **menú** para realizar la conversión en ambos casos:

```
x=0;
while x~=3
   disp('1) Convertir F a C');
   disp('2) Convertir C a F');
   disp('3) Salir');
   x=input('elija una opción ');
   switch x
      case 1
         f=input('ingrese grados F ');
         c=5/9*(f-32);
         disp(c);
      case 2
         c=input('ingrese grados C ');
         f=9/5*c+32;
         disp(f);
   end
end
```

Ejemplo. Un programa con un **menú** y una **repetición** para mostrar el valor de la compra de una pizza. Elegir el tamaño de la pizza y luego ingresar el número de ingredientes adicionales que se desean. Calcular el valor a pagar sabiendo que la pizza pequeña cuesta $**6,** la grande cuesta $**8** y la familiar **$12.** Cada ingrediente adicional cuesta $**1.20.** Incluya el IVA.

t: tamaño de la pizza
n: número de ingredientes
p: valor a pagar

```
op=0;
while op ~= 4
    disp('1) Pizza pequeña ');
    disp('2) Pizza grande ');
    disp('3) Pizza familiar ');
    disp('4) Salir ');
    op=input('Elija una opción ');
    switch op
      case 1,
          n=input('Número de ingredientes ');
          p= 1.12*(6+n*1.2);
          disp(p);
      case 2,
          n=input('Número de ingredientes ');
          p= 1.12*(8+n*1.2);
          disp(p);
      case 3,
          n=input('Número de ingredientes ');
          p= 1.12*(12+n*1.2);
          disp(p);
      otherwise,
          disp('Error');
    end
end
```

Agregue comandos **clc** y **pause** para mejorar la interacción

Ejemplo: Uso interactivo de la fórmula del interés compuesto con un menú.

$$A = P\left[\frac{(1+x)^n - 1}{x}\right],$$ en donde

A: Valor acumulado,
P: Valor de cada depósito **mensual**
n: Cantidad de depósitos **mensuales**
x: Tasa de interés **mensual**

Diseñar un algoritmo que permita interactuar con un **menú** para obtener; **A, P, n**

```
format bank;
t=0;
while t~=4
   disp('1) Valor acumulado');
   disp('2) Depósito mensual');
   disp('3) Número de depósitos');
   disp('4) Salir');
   t=input('Elija una opción ');
   switch t
      case 1,
         p=input('Valor del depósito mensual ');
         n=input('Número de depósitos mensuales ');
         x=input('Interés anual en porcentaje ');
         m=0.01*x/12;
         a=p*((1+m)^n-1)/m;
         disp('Valor acumulado ');
         disp(a)
      case 2,
         a=input('Valor acumulado ');
         n=input('Número de depósitos mensuales ');
         x=input('Interés anual en porcentaje ');
         m=0.01*x/12;
         p=a*m/((1+m)^n-1);
         disp('Cuota mensual ');
         disp(p)
      case 3,
         a=input('Valor acumulado ');
         p=input('Valor del depósito mensual ');
         x=input('Interés anual en % ');
         m=0.01*x/12;
         n=log(a*m/p+1)/log(1+m);
         disp('Numero de depósitos ');
         disp(n)
   end
end
```

Interacción con MATLAB.

1) Valor acumulado
2) Depósito mensual
3) Número de depósitos
Elija una opción 1

Valor del depósito mensual 120
Número de depósitos mensuales 60
Interés anual en % 4
Valor acumulado
7955.88

4.5.1 Ejercicios de programación con menú

1) La suma de los términos de una sucesión aritmética está dada por

$$\mathbf{s = \frac{n}{2}(a + u)}$$, en donde

s: Suma de los términos,
n: Cantidad de términos
a: Primer término de la sucesión,
u: Último término de la sucesión

Escriba un programa con un menú que permita calcular: **s, n, a, u** dados los otros 3 datos.

Menú

1) Suma de términos
2) Cantidad de términos
3) Primer término
4) Último término

5 Vectores en MATLAB

Los vectores son dispositivos para representar y manejar variables que pueden tener varios componentes de un mismo tipo.

5.1 Definición de un vector

Un vector se puede crear con la notación de corchetes **[ ]** para asignar su contenido

Nombre del vector = [valores de sus componentes]

Notación para referirse a los componentes de un vector

Notación Matemática: $\mathbf{x_i}$
Notación MATLAB: **x(i)**

x es el nombre del vector
i es el número de la celda (numeradas en forma natural desde **1**)

Para crear un vector fila los componentes deben separarse con espacios o con comas.

Crear un vector fila de tres componentes

```
>> x=[5, 7, 4];
>> x
x =
  5    7    4
```

Si se desea crear un vector columna, los elementos deben separarse con punto y coma.

Crear un vector columna de tres componentes

```
>> t=[6; 9; 2];
>> t
 t  =
  6
  9
  2
```

Los vectores pueden manejarse dinámicamente. Esta posibilidad es muy útil:

Agregar un nuevo elemento al final del vector **x** anterior:

```
>> x=[x, 8];
>> x
  x =
    5    7    4    8
```

Agregar un nuevo elemento al inicio del vector **x** anterior:

```
>> x=[6, x];
>> x
  x =
    6    5    7    4    8
```

También se puede crear un vector asignando valores a sus componentes con el índice:

Crear un vector fila de tres componentes

```
>> x(1)=5;
>> x(2)=7;
>> x(3)=4;
>> x
x =
    5    7    4
```

El manejo individual de los componentes requiere el uso de un índice

Crear un vector fila

```
>> x=[6, 7, 4, 8, 3];
>> x
x =
  6    7    4    8    3
```

Mostrar el tercer componente

```
>> x(3)
ans =
  4
```

Sustituir el cuarto componente por el 9

```
>> x(4)=9
x =
    6    7    4    9    3
```

Mostrar los componentes 2, 3 y 4

```
>> x(2:4)
ans =
    7    4    9
```

Insertar en la posición 3 el valor 8

```
>> x=[x(1:2),8,x(3:5)]
x =
    6    7    8    4    9    3
```

Para eliminar un elemento, se lo sustituye con un elemento nulo

Eliminar el tercer elemento del vector **x.**

```
>> x(3)= [ ];
>> x
 x =
    6    5    4    8
```

Eliminar el vector completo

```
>> x= [ ];
>> x
   x = [ ]
```

5.2 Algunas funciones de MATLAB para manejo de vectores

Longitud de un vector

```
>> v=[2 4 7 3 5 7 8 6];
>> n=length(v)
n =
  8
```

Suma de los componentes de un vector

```
>> s=sum(v)
s =
42
```

Máximo valor de los componentes

```
>> m=max(v)
m =
    8
```

Máximo valor de los componentes y su posición

```
>> [m,p]=max(v)
m =
    8
p =
    7
```

Media o promedio aritmético

```
>> p=mean(v)
p =
   5.2500
```

Petenencia de un elemento en un vector

```
>> v=[2 4 7 3 5 7 8 6];
>> e=ismember(8,v)
e =
    1
>> e=ismember(9,v)
e =
    0
```

Petenencia de un elemento en un vector y determinar su posición en el vector

```
>> [e,p]=ismember(8,v)
e =
    1
p =
    7
```

Determinar **cuales** elementos de un vector son iguales a un elemento dado

```
>> v=[2 4 7 3 5 7 8 6];
>> t=ismember(v,7)
t =
    0    0    1    0    0    1    0    0
```

El resultado es un vector de coincidencias

Determinar **cuantos** elementos de un vector son iguales a un elemento dado

```
>> v=[2 4 7 3 5 7 8 6];
>> t=ismember(v,7)
t =
    0    0    1    0    0    1    0    0
>> n=sum(t)
n =
    2
```

OBSERVACIÓN: Para facilitar el aprendizaje de los algoritmos y concentrarnos en la lógica de la resolución de los problemas, conviene usar las funciones definidas en el lenguaje MATLAB.

En una etapa posterior se desribe la programación de estas funciones para conocerlas, fundamentar su uso y facilitar el desarrollo de nuevas funciones.

5.3 Algoritmos con vectores

Nota para pruebas con vectores. Al realizar pruebas con vectores de diferente longitud conviene borrar de la memoria el vector de la ejecución anterior pues el vector aún mantiene los elementos de la ejecución anterior. Se sugiere incorporar en el programa o en la ventana de comandos una instrucción para iniciar el vector en cada prueba insertando el comando **clear** con el nombre del vector, o iniciándolo como un vector vacío definiéndolo con la notación **[ ]**

Ejemplo. clear x; o **x=[ ];**

Todas las variables de los programas son visibles desde fuera del programa, es decir que están disponibles y pueden usarse desde la ventana de comandos con el nombre con que fueron creadas.

5.3.1 Ingreso de datos de un vector a un programa

Se pueden ingresar los datos individualmente y agregarlos al vector (agregar cada dato a la derecha). (Previamente requiere conocer cuantos datos se leerán)

```
n=input('cuantos datos? ');
v=[ ];
for i=1:n
   x=input('ingrese dato ');
   v=[v, x];
end
```

También se pueden agregar los datos hacia la izquierda

```
n=input('cuantos datos? ');
v=[ ];
for i=1:n
   x=input('ingrese dato ');
   v=[x, v];
end
```

La manera tradicional es el ingreso de cada dato individualmente al vector usando un índice. (Previamente requiere conocer cuantos datos se leerán)

```
n=input('cuantos datos? ');
for i=1:n
   x=input('ingrese dato ');
   v(i)=x;
end
```

La manera más simple es ingresar el vector completo al programa.
La función **length** detecta la cantidad de datos que ingresaron al vector

```
v=input('ingrese el vector ');
n=length(v);
```

5.3.2 Asignación de valores aleatorios a un vector dentro de un programa

Asignación individual

Ejemplo. Crear un vector aleatorio agregando cada dato al vector (números de un dígito)

```
n=input('cuantos números? ');
v=[ ];
for i=1:n
   x=fix(rand*10);
   v=[v, x];
end
```

También se puede crear el vector asignando los valores mediante un índice

```
n=input('cuantos números? ');
for i=1:n
   v(i)=fix(rand*10);
end
```

Se puede asignar el vector aleatorio completo en una instrucción

```
n=input('cuantos números? ');
v=fix(10*rand(1,n));
```

Se ha generado un vector de **1 fila** y **n columnas** con números aleatorios de una cifra

5.3.3 Práctica con vectores

Es una buena práctica inicial emular o construir porgramas equivalentes a funciones existentes en el lenguaje para comparar resultados y adquirir confianza. Posteriormente se puede enfrentar la solución de problemas nuevos o diferentes.

Ejemplo. Dado un vector, emule la función **SUM** de MATLAB para sumar los componentes:

```
>> x=[7  8   9  12  6];
>> s=sum(x)
        42
```

Un programa equivalente
Sean
x: vector
n: número de componentes
s: suma de los componentes

```
x=input('Ingrese vector ');
n=length(x);
s=0;
for i = 1: n
   s=s+x(i);
end
disp('La suma es');
disp(s);
```

Interacción:

```
Ingrese vector  [7  8  9  12  6]
  La suma es
        42
```

Ejemplo. Dado un vector, sume sus componentes con valor impar.

x: vector
n: número de componentes
s: suma de los componentes

```
%Suma de componentes con valor impar de un vector
x=input('Ingrese vector ');
n=length(x);
s=0;
for i = 1: n
   if mod(x(i), 2) ~= 0
      s=s+x(i);
   end
end
disp(s);
```

Interacción

```
Ingrese vector  [7  8  9  12  6]
        16
```

Ejemplo. Dado un vector, muestre sus componentes en orden opuesto.
x: vector dato
y: vector resultado
n: número de componentes

En esta solución se colocan los elementos que ingresan, en otro vector pero en orden opuesto usando un índice:

```
%Invertir un vector fila
x=input('Ingrese vector ');
n=length(x);
j=n;
for i = 1:n
    y(i)=x(j);
    j=j-1;
end
disp(y);
```

En la siguiente solución se usa la notación de MATLAB para agregar los elementos colocándolos directamente a la izquierda de un vector que crece dinámicamente:

```
%Invertir un vector
x=input('Ingrese vector ');
n=length(x);
y=[ ];
for i = 1:n
    y=[x(i),y];
end
disp(y);
```

Ejemplo. Generar un vector con números aleatorios de dos cifras. Mostrar los elementos mayores al promedio

n: número de componentes
x: vector
p: promedio

```
n=input('Cantidad de elementos ');
for i = 1: n
   x(i)=fix(100*rand);
end
disp(x);
p=mean(x);                    %Función de MATLAB para obtener el promedio
for i = 1: n
   if x(i)>p
     disp(x(i));
   end
end
```

Interacción

```
 Cantidad de elementos  8
81   90   12   91   63    9   27   54
81
90
91
63
54
```

vector
elementos mayores al promedio

Ejemplo. Emule la función MAX de MATLAB para encontar el mayor valor de un vector.

```
>> v=[5 3 8 9 7 9 6];
>> r=max(v)
r =
    9
```

Un programa equivalente
Sean

v: vector
n: número de componentes
r: el mayor valor

La estrategia consiste en colocar el la variable de salida r, el primer elemento del vector. Mediante un ciclo se compara cada uno de los otros elementos con el valor de r. En cada comparación si el elemento tiene un valor myor, se actualiza el valor de r. Al terminar el ciclo contendrá el mayor valor

```
v=input('Ingrese vector ');
n=length(v);
r=v(1);
for i = 2: n
   if v(i) > r
      r = v(i);
   end
end
disp(r);
```

Interacción

```
Ingrese vector  [5 3 8 9 7 9 6]
  9
```

Ejemplo. Emule la función **MAX** de MATLAB para encontrar el mayor valor de un vector y su posición en el vector.

```
>> v=[5 3 8 9 7 9 6];
>> [r,p]=max(v)
r =
    9
p =
    4
```

Programa equivalente
Sean

v: vector
n: número de componentes
r: el mayor valor
p: posición del mayor valor en el vector

Al encontrar cada valor más grande, se guarda su posición en una variable:

```
v=input('Ingrese vector ');
n=length(v);
r=v(1);
p=1;
for i = 2: n
   if v(i) > r
      r= v(i);
      p=i;
   end
end
disp(r);
disp(p);
```

Interacción

```
Ingrese vector  [5 3 8 9 7 9 6]

  9
  4
```

Ejemplo. Emule la función **ISMEMBER** de MATLAB para **buscar un elemento en un vector.**

```
>> v=[5 3 8 9 7 9 6];
>> r=ismember(4, v)
r =
    0
>> r=ismember(9, v)
r =
    1
```

Programa equivalente

Esta función es muy importante y conviene entender la estrategia de su programación
Sean

v: vector
e: elemento que se busca
n: número de elementos del vector
r: resultado de la búsqueda

La idea básica es comparar el dato con cada elemento del vector

La siguiente es una solución incorrecta ¿Puede decir porque?

```
v=input('Ingrese vector ');
e=input('Ingrese elemento ');
n=length(v);
for i = 1: n
   if e==v(i)
      r = 1;
   else
      r = 0;
   end
end
disp(r);
```

Interacción

```
Ingrese vector  [5 3 8 9 7 9 6]
Ingrese elemento 9
   0
```

El problema es que el resultado solo dependerá solamente de la comparación con el último elemento aunque se lo hubiese encontrado en una posición anterior.

Una solución correcta

En la solución se usa la siguiente estrategia: La variable **r** se inicia en **0** antes del ciclo, pero si el elemento es encontrado, **r** toma el valor **1** y ya no cambia a **0**.

```
v=input('Ingrese vector ');
e=input('Ingrese elemento ');
n=length(v);
r=0;
for i = 1: n
   if e==v(i)
      r = 1;
   end
end
disp(r);
```

Interacción

```
Ingrese vector  [5 3 8 9 7 9 6]
Ingrese elemento 9
   1
```

Ejemplo. Emule la función **ISMEMBER** de MATLAB para **buscar un elemento en un vector y determinar su posición en el vector.**

```
>> v=[5 3 8 9 7 9 6];
>> [r,p]=ismember(9, v)
r =
    1
p =
    4
```

Programa equivalente

Sean

v: vector
e: elemento que se busca
n: número de elementos del vector
r: resultado de la búsqueda
p: posición del elemento en el vector

La siguiente es una solución "casi" correcta ¿Puede decir porque?

```
v=input('Ingrese vector ');
e=input('Ingrese elemento ');
n=length(v);
r=0;
p=0;
for i = 1: n
   if e==v(i)
      r = 1;
      p = i;
   end
end
disp(r);
disp(p);
```

Interacción

```
Ingrese vector  [5 3 8 9 7 9 6]
Ingrese elemento 9
   1
   6
```

Problema: si un número está repetido en el vector y las comparaciones continuan hasta el final del ciclo, la posición que entrega el programa corresponderá a la última coincidencia del elemento en el vector. Si se desea la posición de la primera coincidencia, el ciclo debe ser interrumpido. En la siguiente solución se usa la instrucción **break** para salir del ciclo al encontrar el elemento

Una solución correcta

Sean

v: vector
e: elemento que se busca
n: número de elementos del vector
r: resultado de la búsqueda
p: posición del elemento en el vector

```
v=input('Ingrese vector ');
e=input('Ingrese elemento ');
n=length(v);
r=0;
p=0;
for i = 1: n
   if e==v(i)
      r = 1;
      p = i;
      break;
   end
end
disp(r);
disp(p);
```

Interacción

```
Ingrese vector  [5 3 8 9 7 9 6]
Ingrese elemento 9
   1
   6
```

Ejemplo. Obtener una muestra de tamaño **m** de un conjunto de tamaño **n** sin repeticiones

La siguiente es una solución incorrecta. ¿Puede decir porque?

n: total de elementos (población)
m: tamaño de la muestra
e: cada número aleatorio entre 1 y n

```
n=input('Total de elementos ');
m=input('Tamaño muestra ');
for i=1:m
   e=fix(n*rand)+1;
   disp(e);
end
```

Interacción

```
Total de elementos 10
Tamaño de la muestra 4
    5   7   9   7
```

Una solución correcta requiere un vector para almacenar los elementos para poder verificar que no estén repetidos. La salida del ciclo se produce cuando el vector con la muestra está lleno.

Con la idea anterior se plantea la siguiente solución, pero no está completamente correcta. ¿Puede decir porque?

n: total de elementos (población)
m: tamaño de la muestra
e: cada número aleatorio entre 1 y n
v: vector con los elementos seleccionados para la muestra

```
n=input('total de elementos ');
m=input('tamaño muestra ');
v=[ ];
for i=1:m
   e=fix(n*rand)+1;
   r = ismember(e, v);
   if r == 0
       v=[v, e];
   end
end
disp(v)
```

Interacción

```
Total de elementos 10
Tamaño de la muestra 4
    5   7   9
```

Problema: la cantidad de ciclos **m** controlada con la instrucción **for** es fija. Por lo tanto, se pierden los ciclos en los que los hay elementos repetidos y no se completa el vector de salida.

Una solución correcta

La estrategia para esta solución es continuar agregando elementos aleatorios diferentes al vector mientras la cantidad de elementos sea menor al tamaño de la muestra. Por lo tanto, la cantidad de ciclos continuará hasta llenar el vector.

n: total de elementos (población)
m: tamaño de la muestra
e: cada número aleatorio entre 1 y n
v: vector con los elementos seleccionados para la muestra

```
n=input('total de elementos ');
m=input('tamaño muestra ');
v=[ ];
while length(v) < m
   e=fix(n*rand)+1;
   r = ismember(e, v);              % si e no está, se lo agrega al vector de salida
   if r == 0
       v=[v, e];
   end
end
disp(v)
```

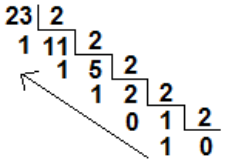
Interacción

```
Total de elementos 10
Tamaño de la muestra 4
    5   7   9   3
```

Ejemplo. Dado un número entero encuentre los dígitos de su equivalente en el sistema binario.

El algoritmo para obtener los dígitos binarios de un número entero decimal consiste en dividirlo sucesivamente para 2. Los resíduos de la división entera tomados desde el final hacia arriba son los dígitos buscados.

Ejemplo. Obtener los dígitos binarios del número **23**



Entonces, **23** es equivalente a **1 0 1 1 1** en el sistema binario

Sean

n: número entero positivo decimal
v: vector que contiene los dígitos de n en el sistema binario.

```
n=input('Ingrese un número ');
v=[ ];
while n>0                    % repetir mientras el número n tenga dígitos
   d=mod(x,2);
   n=fix(n/2);
   v = [d, v];               % el dígito d se agrega a la izquierda en el vector v
end
disp(v);
```

Interacción

Ingrese un numero 23
1 0 1 1 1

Ejemplo. Dado un vector conteniendo los dígitos binarios de un número, encontrar el número

El algoritmo consiste en usar la definición aritmética mediante la suma de las cifras binarias multiplicadas por la base 2 elevada a la potencia que corresponde al dígito binario.

Ejemplo. Dadas las cifras en el sistema binario: **1 0 1 1 1** encontrar su equivalente en el sistema decimal

$$n = 1x2^4 + 0x2^3 + 1x2^2 + 1x2^1 + 1x2^0 = 16 + 0 + 4 + 2 + 1 = 23$$

Sean

v: vector que contiene los dígitos de n en el sistema binario.
c: cantidad de elementos del vector (dígitos binarios)
n: número decimal equivalente

```
v=input('Ingrese el vector con los dígitos binarios ');
c = length(v);
n=0;
for i=1:c
   e=v(i)*2^(c-i);
   n=n + e;
end
disp(n);
```

Interacción

Ingrese el vector con los dígitos binarios [1 0 1 1 1]

23

Ejemplo. Almacene en un vector los números de la secuencia de Ulam:

$$x = \begin{cases} x/2, & x \text{ par} \\ 3x+1, & x \text{ impar} \end{cases}$$, x es un número natural

Muestre la secuencia comenzando desde el final

Sean

x: valor inicial para la secuencia
u: vector con los números de la secuencia
n: cantidad de números de la secuencia
i: indice

```
x=input('Valor inicial para la secuencia ');
n=0;
while x>1
    if mod(x, 2) == 0
        x=x/2;
    else
        x=3*x+1;
    end
 n = n + 1;
 u(n) = x;
end
for i=n:-1:1
  disp(u(i));
end
```

Interacción

Valor inicial para la secuencia 10

1
2
4
8
16
5

Otra solución para el ejemplo anterior (vector con números de la secuencia de Ulam). En esta solución el vector se construye dinámicamente agregado directamente cada número:

x: valor inicial para la secuencia
u: vector con los números de la secuencia

```
x=input('Valor inicial para la secuencia ');
u=[ ];
while x>1
    if mod(x, 2) == 0
        x=x/2;
    else
        x=3*x+1;
    end
 u=[u, x];
end
disp(u);
```

Interacción

Valor inicial para la secuencia 10
5 16 8 4 2 1

NOTA: Los vectores que se construyen con la notación anterior son filas y se muestran horizontalmente. Si se desea que el vector resultante sea un vector columna y se muestre verticalmente hay varias opciones:

a) Mostrar en la instrucción de salida el vector transpuesto (se usa la tilde para obtenerlo)

disp(u'); (Vector u transpuesto)

b) Construir el vector columna usando punto y coma en luga de coma:

u = [u; x]; (separar los elementos con punto y coma)

c) Usar un ciclo **for**: Cada elemento se muestra en una nueva línea en la salida:

```
for i=1:n
   disp(u(i));
end
```

Ejemplo. Construya un vector con una cantidad de números primos especificada

k: cantidad de números primos
c: conteo de divisores exactos de **n**
n: números naturales
p: vector con los números primos

En la solución se prueba cada número natural **n = 1, 2, 3, 4, ...**

Mediante el conteo de sus divisores se detecta si **n** es primo. En este caso se lo agrega al vector p

```
k=input('¿Cuantos números primos? ');
n=1;
p = [ ];
while length(p) < k                    %La salida se produce si el vector p está lleno
   c=0;
   for d=1: n
      if mod(n, d) == 0
         c=c+1;
      end
   end
   if c<=2
      p = [p, n];
   end
   n=n+1;
end
disp(p);
```

Interacción

¿Cuantos números primos? 8

1 2 3 5 7 11 13 17 19

Ejemplo. Desarrolle un un algoritmo para asignar las parejas para el juego del "amigo secreto". Existen **n** varones numerados del **1** a **n**, y **n** mujeres numeradas también de **1** hasta **n.** La asignación de parejas debe ser aleatoria de tal manera que cada persona esté asignada solamente una vez.

Sean

h: Vector con los números de los varones asignados en orden consecutivo
d: Vector con los números de las mujeres, asignados aleatoriamente.
x: Cada número aleatorio entre 1 y n

```
%Parejas aleatorias
n=input('Cuantas parejas ');
for i=1:n
   h(i)=i;                    % La asignación de hombres es secuencial
end
d=[ ];
while length(d)<n             % La asignación de damas es aleatoria
   x=fix(n*rand)+1;
   r=ismember(x,d);
   if r==0
      d=[d, x];               % el vector d debe contener todos los números
   end                        % y en orden aleatorio
end
for i=1:n
   disp([h(i),d(i)]);         %Salida de parejas
end
```

Interacción

```
Cuantas parejas 8
   1    7
   2    8
   3    2
   4    6
   5    1
   6    3
   7    5
   8    4
```

Ejemplo. Lea un vector y muestre otro vector que contenga los mismos elementos pero sin elementos repetidos

Sean

v: Vector con los datos
u: Vector con los datos de **v** pero sin incluir repetidos

En la solución, se toma cada elemento del vector original y se lo traslada a otro vector verificando que no haya sido agregado anteriormente.

```
%Eliminar elementos repetidos
v=input('Ingrese el vector ');
n=length(v);
u=[ ];
for i=1:n
   r=ismember(v(i), u);
   if r==0
      u=[u, v(i)];
   end
end
disp(u);
```

Interacción

```
Ingrese el vector [3 7 6 5 3 6 8]
    3     7     6     5     8
```

Ejemplo. Simule **n** lanzamientos de un dado. Muestre la cantidad de veces que sale cada número.

n: cantidad de lanzamientos
d: número obtenido para el dado en cada lanzamiento
c: vector con la cantidad de resultados de cada número del dado

```
%Conteo de resultados de un dado
n=input('Cantidad de pruebas ');
c=[0 0 0 0 0 0];
for i=1:n
    d=fix(rand*6)+1;
    switch d
       case 1, c(1) = c(1)+1;
       case 2, c(2) = c(2)+1;
       case 3, c(3) = c(3)+1;
       case 4, c(4) = c(4)+1;
       case 5, c(5) = c(5)+1;
       case 6, c(6) = c(6)+1;
    end
end
disp(c);
```

Interacción

```
Cantidad de pruebas  6000
      999       988       1016       1007       986       1004
```

La siguiente solución es una compactación de la solución anterior. Se observa que el número del dado se puede usar como el índice para el elemento que hay que incrementar en el vector:

Ejemplo. Simule **n** lanzamientos de un dado. Muestre la cantidad de veces que sale cada número.

Sean
n: cantidad de lanzamientos
d: número obtenido para el dado en cada lanzamiento
c: vector con la cantidad de resultados de cada número del dado

```
%Conteo de resultados de un dado
n=input('¿cuantas pruebas? ');
c=[0 0 0 0 0 0];
for i=1:n
    d=fix(rand*6)+1;
    c(d)=c(d)+1;
end
disp(c);
```

Ejemplo. Generar una tabla de 15 números con valores entre 1 y 25

```
t=[ ];                          % contendrá la tabla de 15 números diferentes
while length(t)<15
   x=fix(25*rand)+1;            % número aleatorio entre 1 y 25
   if ismember(x,t)==0
      t=[t,x];
   end
end
disp(t');
```

Interacción

```
   7   21   24    9    5   16   12   15   14   23    8   19   10    2   20
```

Ejemplo. Escriba un programa para organizar los 32 equipos participantes en un campeonato en 8 grupos de 4 equipos. Cada grupo debe tener un equipo fijo que debe ser un dato inicial

```
c=input('Ingrese los 8 equipos fijos en cada grupo ');
e=c;                        %Contendrá todos los equipos para comparar repetidos
for i=1:8
   g=[c(i)];                %Cada grupo de 4 se inicia con un equipo fijo (dato)
   while length(g)<4
      x=fix(32*rand)+1;
      if ismember(x,e)==0   %Verificar si el equipo ya fue asignado antes
         g=[g,x];
         e=[e,x];
      end
   end
   disp('Grupo');
   disp(i);
   disp(g);
end
```

Interacción

```
Ingrese los 8 equipos fijos en cada grupo [ 2 4 7 8 3 5 9 12]
Grupo
   1
   2   23   24   15
Grupo
   2
   4   30   27   18
Grupo
   3
   7   32   31    1
Grupo
   4
   8   25   28   13
Grupo
   5
   3   26   14    6
Grupo
   6
   5   19   20   17
Grupo
   7
   9   29   16   11
Grupo
   8
  12   21   10   22
```

Ejemplo. Escriba un programa que lea desde el teclado dos vectores y determine la cantidad de elementos comunes entre ambos vectores.

Sean

a, b: vectores con datos
c: conteo de datos comunes

En la solución propuesta se examina cada elemento de un vector y se lo busca en el otro vector usando la función ISMEMBER. Si lo encuentra se hace el conteo.

```
a=input('primer vector ');
b=input('segundo vector ');
c=0;
for i=1: length(a)
   if ismember(a(i),b)==1
      c=c+1;
   end
end
disp('cantidad elementos comunes');
disp(c);
```

Interacción

```
primer vector [3 5 7 8 3 9]
segundo vector [5 3 6 7 1]
cantidad elementos comunes
   4
```

Ejemplo. Escriba un programa para ordenar en forma ascendente los datos de un vector.

Sean

x: vector con los datos
y: vector con los datos ordenados

En esta solución se usará función **MIN** de MATLAB para obtener el menor valor de un vector y su posición. El algoritmo tomará este valor del vector **x** y lo agregará al vector **y**, pero además lo eliminará del vector **x**.

```
x=input('Ingrese el vector ');
y=[ ];
while length(x)>0
   [e,p]=min(x);
   y=[y, e];
   x(p)=[ ];
end
disp(y);
```

Interacción

```
Ingrese el vector  [6  2  7  3  8  9  5  4]

   2    3    4    5    6    7    8    9
```

Ejemplo. Escriba un programa para **ordenar** un vector pero sustituyendo la función **MIN** con instrucciones correspondientes

```
x=input('Ingrese el vector ');
y=[ ];
while length(x)>0
   n=length(x);
   e=x(1);
   p=1;
   for i=2:n
      if x(i)<e
         e=x(i);
         p=i;
      end
   end
   y=[y, e];
   x(p)=[ ];
end
disp(y);
```

Interacción

Ingrese el vector [6 2 7 3 8 9 5 4]

2 3 4 5 6 7 8 9

Ejemplo. Colocar el mayor valor de un vector en la última posición

Sean

x: vector
n: número de componentes

El algoritmo compara cada dato con el último elemento, si es mas grande, los intercambia. Al final, en la última posición quedará el mayor valor.

```
x=input('Ingrese vector ');
n=length(x);
for j = 1: n-1
   if x(j) > x(n)
      t = x(j);
      x(j) = x(n);
      x(n) = t;
   end
end
disp(x);
```

Interacción

Ingrese vector [6 2 7 3 8 9 5 4]

4 2 6 3 7 8 5 9

Ejemplo. Ordenar los datos de un vector con el método de intercambio de elementos del ejemplo anterior.

Sea

x: vector con los datos, su contenido es modificado para ordenarlo

La idea es aplicar en un ciclo el intercambio de elementos, dejando en cada repetición el mayor valor al final del vector. En cada repetición el vector se reduce excluyendo al último elemento.

```
x=input('Ingrese un vector ');
n=length(x);
for k=n:-1:2
    for j=1:k-1
        if x(j) > x(k)
           t=x(j);
           x(j)=x(k);
           x(k)=t;
        end
    end
end
disp(x)
```

Interacción

Ingrese el vector [6 2 7 3 8 9 5 4]

2 3 4 5 6 7 8 9

Ejemplo. Escriba un programa para desordenar un vector

Sean
v: vector inicial
u: vector con los elementos de v en orden aleatorio

En esta solución se toma aleatoriamente cada elemento de **v** y se lo agrega a **u**. Este elemento debe ser eliminado del vector **v.** La longitud del vector **v** se reduce en cada ciclo

```
%Desordenar un vector
v=input('Ingrese el vector ');
u=[ ];
while length(v)>0
   n=length(v);
   p=fix(n*rand)+1;
   u=[u,v(p)];
   v(p)=[ ];
end
disp(u);
```

Interacción

```
Ingrese el vector [ 2 5 6 7 9 10 15 20]
   7   15    9    5   20   10    6    2
```

Otra solución para desordenar un vector

En esta solución se crea un vector **p** con las posiciones desordenadas aleatoriamente. Se usan estas posiciones para tomar cada elemento del vector **v** para agregarlo al vector **u**

Sean
v: vector inicial
u: vector con los elementos de v en orden aleatorio
p: vector con las posiciones de los elementos de v en orden aleatorio

```
%Desordenar un vector
v=input('Ingrese un vector ');
n=length(v);
p=[ ];
u=[ ];
while length(p)<n
   i=fix(n*rand)+1;
   r=ismember(i,p);
   if r==0
      p=[p, i];
      u=[u,v(i)];
   end
end
disp(u);
```

Interacción

```
Ingrese el vector  [2 4 3 6 7 8 2 7]

   6    7    2    4    8    7    2    3
```

Ejemplo. Manejo de un conjunto. Escriba un programa para simular algunas operaciones de manejo de un conjunto mediante un menú que ofrezca las siguientes opciones:

1) Agregar un elemento al conjunto
2) Eliminar un elemento del conjunto
3) Determinar si un elemento pertenece al conjunto
4) Salir

Use un vector para almacenar el conjunto. Al inicio es un vector nulo.

En cada opción el programa debe pedir un elemento al usuario y realizar la operación solicitada.

```
v=[];
x=0;
while x~=4
   disp('1) Agregar elemento');
   disp('2) Eliminar elemento');
   disp('3) Pertenecia de un elemento');
   disp('4) Salir');
   x=input('Elija una opcion ');
   switch x
      case 1,
         e=input('ingrese elemento ');
         if ismember(e,v)==0
            v=[v, e];
         end
      case 2,
         e=input('ingrese elemento ');
         [z,p]=ismember(e,v);
         if z == 1
            v(p)=[];
         end
      case 3;
         e=input('ingrese elemento ');
         z=ismember(e,v);
         disp(z);
   end
end
```

5.4 Ejercicios con vectores

1. Una persona tiene una lista con los precios de n artículos y dispone de una cierta cantidad de dinero. Escriba un programa para leer estos datos y almacenarlos en un vector:

a) Muestre los numeros de los artículos que puede comprar
b) Para cada artículo cuyo precio es menor que la cantidad de dinero disponible, determine cuantas unidades puede comprar

2. Lea una lista de los pesos de las **n** cajas en un contenedor. Determine cuantas cajas tienen el peso mayor al peso promedio del grupo.

3. Lea una lista de los pesos de los **n** objetos en una bodega. Determine cual es el rango de los pesos de estos objetos.

4. Lea los códigos de las **n** cajas de un contenedor. Determine si hay códigos repetidos

5. Una bodega contiene **n** paquetes numerados en forma natural. Para una inspección se debe tomar una muestra aleatoria del **10%** de los paquetes. Escriba un programa para elegir la muestra. La muestra no debe contener elementos repetidos.

6. Para la inspección de los **m** paquetes de una bodega se han elegido a **m** inspectores. Realice aleatoriamente la asignación de tal manera que cada inspector se le asigne la revisión de un solo paquete.

7. Para la inspección de los **m** contenedores de una bodega se han elegido a **m/2** inspectores. Realice aleatoriamente la asignación de tal manera que cada inspector se le asigne la revisión de dos contenedores. No se puede asignar un contenedor más de una vez.

8. Escriba un programa para leer un vector con números enteros de una cifra. Luego genere un número aleatorio de una cifra. Entonces, elimine del vector todos los números cuyo valor sea menor al número aleatorio. Muestre el vector resultante.

Nota. Para eliminar un elemento de un vector, se lo sustituye por un elemento nulo: [ ]

9. Lea la lista de los números de identificación de los estudiantes que están inscritos en la material A, y otra lista con los estudiantes que están inscritos en la materia B.

a) Encuentre cuantos estudiantes están inscritos en la materia A y también en la materia B
b) Encuentre los estudiantes que están inscritos en la materia A pero no en la materia B

10. Se tiene la lista de los códigos de las cajas en la bodega A y una lista de los códigos de las cajas en la bodega B. Además se tiene una lista con los códigos de las cajas que deben ser inspeccionadas. Determine cuantas cajas en cada bodega serán inspeccionadas.

11. Almacene en un vector **X** las abscisas y en un vector **Y** las ordenadas de un conjunto de puntos en un plano. Ambos vectores son ingresados como datos. Determine cual es el punto más alejado del origen. Use la fórmula de la distancia. $\mathbf{d = \sqrt{x_i^2 + y_i^2}}$

12. Almacene en un vector **X** las abscisas y en un vector **Y** las ordenadas de un conjunto de puntos en un plano. Ambos vectores son ingresados como datos. Suponga que estos puntos representan los vértices de un polígono. Determine el perímetro del polígono.

Use la fórmula para calcular la distancia entre cada par de puntos.

$$\mathbf{d = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}$$

13. El cálculo del área de un polígono arbitrario, dadas las coordenadas de sus vértices **$(x_1, y_1)(x_2, y_2), ..., (x_n, y_n)$** se lo puede hacer con la siguiente fórmula:

$$\textbf{Area} = [(x_1+x_2)(y_1-y_2) + (x_2+x_3)(y_2-y_3) + \ldots + (x_{n-1}+x_{n-2})(y_{n-1}-y_{n-2}) + (x_n+x_1)(y_n-y_1)]/2$$

Escriba un programa para leer en vectores los valores de las abscisas y ordenadas de los vértices. Calcule y muestre el valor del área con la fórmula anterior

14. Para simular los saltos de **n** ranas en una pista de longitud **m** metros se usará un vector de **n** elementos que inicialmente contiene ceros. Use un ciclo para agregar a cada rana un número aleatorio que puede ser **0**, **1** o **2** metros. Repita este ciclo hasta que alguna rana llegue al final de la pista. Muestre en cual turno alguna rana llegó al final de la pista.

15. En un proceso electoral se tienen anotados los **n** votos para aprobar una moción. Cada voto tiene el número de identificación del elector (números enteros del **1** al **n**) y un número que representa su decisión: 1 si es a favor, 0 si es en contra, cualquier otro número es nulo. Escriba un programa que lea los **n** datos conteniendo el número del elector (no suponga que están ordenados) y su voto. Coloque los números de identificación en tres listas: votantes a favor, votantes en contra y votantes nulos. Finalmente busque y muestre si hay números de identificación de electores que están en más de una lista.

16.- Desarrolle un programa basado en vectores para instrumentar las operaciones básicas entre dos conjuntos A, B mediante las siguientes opciones de un menú:

1) Agregar elemento
2) Eliminar elemento
3) Pertenencia
4) Consultar
5) Unión
6) Intersección
7) Diferencia
8) Salir

En la opción 1) preguntar cual conjunto: A o B y luego pedir el elemento para agregar
En la opción 2) preguntar cual conjunto: A o B y luego pedir el elemento para eliminar
En la opción 3) pedir el elemento y determinar en cual conjunto se encuentra (pueden ser ambos)
En la opción 4) preguntar cual conjunto: A o B y luego mostrar sus elementos

17.- Se tiene como dato la cantidad de boletos disponibles para un concierto. Escriba un programa para organizar la venta en línea. Cada fan debe presentar su cédula de identidad. El programa debe leer y agregar a un vector el número de cédula. En caso de que el número de cédula ya esté en el vector, muestre un mensaje rechazando la venta de boletos. Si la venta se realiza, lea la cantidad de boletos que se compra (no debe ser mayor a 4) y reste de la cantidad disponible. Cuando esta cantidad llegue a cero, muestre un mensaje y finalice el programa.

18. En una escuela de fútbol se inscriben **n** jugadores identificados con su número en la lista de asistencia y un código que identifica su habilidad **1**: portero, **2**: defensa, **3**: mediocampista, **4**: delantero. Este dato debe ser ingresado y validado.

Escriba un algoritmo para ayudar al entrenador a formar dos equipos de **11** jugadores elegidos aleatoriamente. Cada equipo debe contener **1** portero, **4** defensas, **2** mediocampistas y **4** delanteros. Cada jugador no puede pertenecer a más de un equipo. Suponga que en la lista hay suficientes jugadores para elegir.

6 Cadenas de caracteres

Las aplicaciones computacionales del manejo de cadenas de caracteres son de interés general y también pueden ser parte de la solución de problemas de ingeniería.

Una cadena de caracteres es una secuencia de letras, números y símbolos especiales. La representación de estos objetos en la memoria es diferente a la representación de los números.

6.1 Algunos comandos para cadenas de caracteres

Una cadena de caracteres se las expresa con la notación de apóstrofes:

Asignación directa de una cadena de caracteres. Se la debe escribir entre apóstrofes

```
>> x='problema';
```

Una subcadena es una porción de una cadena de caracteres

```
>> t = x(2:6)
t =
   roble
```

La notación dinámica de MATLAB permite operar con facilidad cadenas de caracteres

Agregar caracteres

```
>> x=[x,'s']
x =
problemas
```

Insertar caracteres

```
>> x=[x(1:3),'x',x(4:9)]
x =
  proxblems
```

Cadena nula. Puede usar la notación **[ ]** o simplemente dos comillas seguidas:

```
>> x='';
```

Eliminar caracteres. Se sustituye por un carácter nulo

```
>> x='problema';
>> x(3)=''
x =
prblema
```

Determinar si un carácter se encuentra en una cadena

```
>> x='programa';
>> e=ismember('r',x)
e =
   1
```

También se puede conocer su posición

```
>> [e,p]=ismember('r',x)
e =
    1
p =
   2
```

Entrega la primera posición si hay más de una coincidencia

```
>> x='programa';
>> [e,p]=ismember('a',x)
e =
   1
p =
   6
```

Examina la coincidencia y posición de cada carácter si hay más de uno

```
>> [e,p]=ismember('rta',x)
e =
   1    0    1
p =
   2    0    6
```

Determinar si una cadena se encuentra en otra cadena. El resultado es un vector con las posiciones de coincidencia

```
>> x='En esta prueba para esta materia';
>> t='esta';
>> e=strfind(x,t)
e =
   4   21
>> length(e)
ans =
   2                                   Es la cantidad de coincidencias
```

Determinar si dos cadenas son idénticas: el resultado es un valor lógico

```
>> x='abc';
>> y='abc';
>> t=strcmp(x,y)
t =
   1
```

Funciones para convertir entre numérico y cadena de caracteres

```
>> x=234;
>> c=num2str(x)
c
   = 234

>> c='345';
>> n=str2num(c);
n
   = 345
```

6.2 Listas de cadenas de caracteres

```
>> a=['abc';'rst';'xyz']          Las cadenas deben tener igual longitud
a =
abc
rst
xyz
>> a(2,:)
ans =
rst

>> a=char('abcd','rs','uwxyz','tx')      La function char forma listas con cadenas
a =                                      que pueden tener diferente longitud
abcd
rs
uwxyz
tx
>> [n,k]=size(a)
n =
     4                    Primer componente de size: cantidad de cadenas
k =
     5                    Segundo componente de size: la mayor longitud
```

NOTA: Para manejo de vectores de cadenas de caracteres de diferente longitud, es más conveniente el uso de **arreglos de celdas**. Este tipo especial de datos se revisará en una sección posterior.

Ejemplo. Salida de un título junto a un número. También puede usarse en input

```
for i=1:5
        disp(['dato ', num2str(i)])
end
```

```
dato 1
dato 2
dato 3
dato 4
dato 5
```

Lectura de una cadena

```
x = input('Ingrese una cadena ');
```

El dato debe ser ingresado encerrado entre comillas simples o mediante una variable previamente asignada con una cadena

Se pueden omitir las comillas al ingresar el dato si se incluye la especificando **'s'** en input

```
x = input('Ingrese una cadena ', 's');
```

6.3 Algoritmos con cadenas de caracteres

Ejemplo. Leer una cadena de caracteres y mostrarla con los caracteres puestos en orden inverso

```
x=input('ingrese una frase ');
n=length(x);
y= '';                          % también puede iniciarla con  [ ]
for i=1:n
   y=[x(i),y];                  % insertar carácter a la izquierda de la cadena
end
disp(y)
```

Interacción

```
ingrese una frase esta es una prueba
abeurp anu se atse
```

Con el mismo programa se puede restaurar la frase original

```
>> invertir
ingrese una frase abeurp anu se atse
esta es una prueba
```

Ejemplo. Leer una cadena de texto y eliminar los espacios en blanco

Solución 1. El resultado se coloca en otra variable

```
x=input('ingrese frase ',’s’);
n=length(x);
y=[ ];
for i=1:n
   if x(i)~=' '
      y=[y,x(i)];
   end
end
disp(y);
```

Solución 2. La misma cadena de entrada es modificada

```
x=input('ingrese frase ',’s’);
n=length(x);
i=0;
while i<length(x)              % No se puede usar FOR debido a que la longitud
   i=i+1;                      % cambia dinámicamente
   if x(i)==' '
      x(i)=[ ];                % Elimina el carácter. También se puede: x(i) = '';
      i=i-1;                   % Reducir en 1 la longitud de la cadena
   end
end
disp(x);
```

Ejemplo. Escribir un programa para determinar si una frase se puede leer de izquierda a derecha o de derecha a izquierda. Estas frases se denominan “palíndromes”. El programa es una combinación del algoritmo que elimina espacios en blanco y el algoritmo que invierte una frase.

```
x=input('ingrese una frase ');
n=length(x);
y=[ ];
for i=1:n
   if x(i)~=' '
      y=[x(i),y];
   end
end
e=strcmp(x,y);
if e==1
  disp('si');
else
   disp('no');
end
```

Ejemplos: radar, reconocer, la ruta nos aportó otro paso natural, damas oid a dios amad

Ejemplo. Leer una frase y determinar cuantas palabras contiene

```
x=input('ingrese frase ',’s’);
n=length(x);
c=1;
for i=1:n
   if x(i)==' '
      c=c+1;
   end
end
disp([‘cantidad de palabras ’, num2str(c)]);
```

Ejemplo. Leer una frase y determinar cuántas veces contiene a una palabra

```
x=input('ingrese frase ',’s’);
p=input(‘ingrese palabra’,’s’);
e=strfind(x,p);                    %Vector conteniendo los inicios de coincidencias
n=length(e);
disp(n);
```

Ejemplo. Leer una frase y mostrarla enmascarada intercambiando parejas consecutivas de caracteres

```
x=input('Ingrese mensaje ',’s’);
y='';
n=length(x);
for i=1:2:n-1
   a=x(i);
   b=x(i+1);
   y=[y,b,a];
end
if mod(n,2)==1
   y=[y,x(n)];                     %si n es impar, agregar el último carácter
end
disp(y);
```

Interacción

Ingrese mensaje esta es una prueba
seate snu arpeuab

Con el mismo programa se puede restaurar la frase original

Interacción

Ingrese mensaje seate snu arpeuab
esta es una prueba

Ejemplo. Diseñe un esquema para enmascarar una frase colocando los caracteres alternadamente alrededor del centro.

Codificar el mensaje

```
%Enmascara colocando caracteres alrededor del centro
x=input('Ingrese mensaje ','s');
y=[ ];
if mod(length(x),2)==1
   x=[' ',x];
end
n=length(x);
for i=1:2:n
   a=x(i);
   b=x(i+1);
   y=[b,y,a];
end
disp(y);
```

Decodificar el mensaje

```
%Decodifica el mensaje del algoritmo anterior
x=input('Ingrese mensaje ','s');
y=[ ];
n=length(x);
for i=1:n/2
   a=x(n/2+i);
   b=x(n/2-i+1);
   y=[y,a,b];
end
disp(y);
```

Ejemplo. Lea nombres que pueden tener diferente longitud y forme una lista. Muestre por cada nombre, la cantidad de veces que contiene la letra ‘a’

Note el uso de **char** para crear la lista de nombres de diferente longitud

```
%Lista vertical de nombres
n=input('cantidad de nombres ');
t=input('ingrese primer nombre ','s');
x=char(t);                              %inicia lista vertical
for i=1:n-1
   t=input('ingrese siguiente nombre ','s');
   x=char(x, t);                  % char crea una lista vertical
end                               % con los siguientes n-1 nombres
for i=1:n
   v=strfind(x(i,:),'a');         % vector de posiciones de coincidencias de 'a' en x
   c=length(v);
disp([i, c]);
end
```

Ejemplo. Lea una lista de nombre, elimine los nombres con 'r'

```
t=input('ingrese lista ');
i=1;
while i<size(t,1)                    %El primer componente de size es la longitud de la lista
    if ismember('r',t(i,:))==1
        t(i,:)='';
    end
    i=i+1;
end
disp(t)
```

Interacción

```
>> t=char('maria','juan','pedro','luis')
t =
maria
juan
pedro
luis

>> prueba
ingrese lista t
juan
luis
```

6.4 Ejercicios con cadenas de caracteres

1. Escriba un programa que realice lo siguiente

a) Lea una frase.
b) Cuente y muestre cuantas vocales tiene la frase
c) Lea una palabra, cuente y muestre cuantas veces la frase contiene a la palabra
d) Elimine todas las vocales que contiene la frase. Muestre la frase final

2. Escriba un programa que lea una dirección de correo electrónico con formato **usuario@dominio.tipo** y muestre separadamente cual es el usuario y el tipo

3. Escriba un programa que lea una frase y enmascárela sustituyendo las vocales con símbolos: 'a' sustituya con '*', 'e' con '-', 'i' con '?', 'o' con '&', 'u' con '#'
Escriba otro programa que haga la sustitución inversa y restaure la frase original.

4. Una cadena ADN es una linea de texto conteniendo una lista de los caracteres A, C, G, T en cualquier secuencia. Ejemplo. CCGAATCGTA

Se considera que cada par de caracteres consecutivos está ordenado si el carácter a la izquierda es alfabéticamente menor o igual que el carácter a la derecha.

Escriba un programa que reciba una cadena ADN y muestre cuantos pares de la cadena están ordenados. Verifique que la cadena tenga caracteres válidos, caso contrario, muestre un mensaje.

5. Rescriba un programa que reciba una frase y desordena las letras en forma aleatoria.

Ejemplo: Recibe 'martes', entrega 'remsta'

Sugerencia: En un ciclo elija aleaoriamente la posición de cada letra de la frase, agrégue la letra de esa posición a otra variable pero elimínela de la frase original para que no sea elegida otra vez. Note que la longitud de la cadena original cambia, por lo que no conviene usar for

6. Escriba un programa para jugar el juego del ahorcado entre una persona y el computador. Primero almacene una lista de palabras en un vector. Luego el computador selecciona una palabra aleatoriamente pero no la muestra. La persona trata en intentos sucesivos adivinar la palabra ingresando una letra en cada intento. El computado muestra las letras que coinciden con la palabra seleccionada, pero en cada fallo, muestra un mensaje que acerca a la paersona a ser ahorcado.

7. Un paso importante en la decodificación de mensajes encriptados es encontrar algunas letras utilizadas. Para ello se debe determinar la frecuencia de los símbolos usados y asociarlos a las letras del alfabeto. Por ejemplo, en el español la letra de mayor frecuencia es la letra e. Escriba un programa que lea un mensaje y determine la frecuencia de cada símbolo utilizado.

7 Matrices en MATLAB

Es el tipo fundamental de datos en MATLAB. Se puede asociar una matriz a un cuadro con datos organizados en filas y columnas.

Por simplicidad supondremos que las filas son horizontales numeradas con la numeración natural de arriba hacia abajo, y las columnas son verticales, numeradas con la numeración natural de izquierda a derecha.

Notación:

Nombre de la matriz = **[**elementos de fila 1**;** elementos de fila 2**;** elementos de fila 3**;...]**

7.1 Matrices en la ventana de comandos

Definición de una matriz

```
>> a=[2 5 7; 4 6 2; 8 9 3]
a =
    2    5    7
    4    6    2
    8    9    3
```

El manejo de sus elementos requiere especificar el número de fila y de columna (índices)

```
>> e=a(3,2)
e =
    9
```

Puede manejar una fila completa

```
>> d=a(2, 1:3)
d =
    4  6  2
```

También columnas

```
>> d=a(1:3,2)
d =
    5
    6
    9
```

Notación abreviada

```
>> d=a(2, :)
d =
    4  6  2
```

Puede manejar una submatriz

```
>> t=a(1:2, 2:3)
t =
    5    7
    6    2
```

Agregar una fila a una matriz

```
>> a=[2, 4, 5; 6, 0, 7; -2, 8, 3]
a =
    2    4    5
    6    0    7
   -2    8    3
```

```
>> a(4,:)=[5 7 9]
a =
    2    4    5
    6    0    7
   -2    8    3
    5    7    9
```

Agregar columna

```
>> a(:,4)=[3;5;7;9]
a =
    2    4    5    3
    6    0    7    5
   -2    8    3    7
    5    7    9    9
```

Eliminar columna: se la reeplaza por un elemento nulo

```
>> a(:,4)=[ ]
a =
    2    4    5
    6    0    7
   -2    8    3
    5    7    9
```

Eliminar fila

```
>> a(3,:)=[ ]
a =
    2    4    5
    6    0    7
    5    7    9
```

Insertar columna

```
>> b=[3;9;8]
b =
    3
    9
    8
>> a=[a(:,1),b,a(:,2:3)]
a =
    2    3    4    5
    6    9    0    7
    5    8    7    9
```

Matrices especiales

Matriz de ceros

```
>> a=zeros(3,4)
a =
    0    0    0    0
    0    0    0    0
    0    0    0    0
```

Matriz con 1's

```
>> a=ones(3,4)
a =
    1    1    1    1
    1    1    1    1
    1    1    1    1
```

Matriz identidad

```
>> a=eye(4,4)
a =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Matriz con números aleatorios

```
>> a=rand(4,3)
a =
   0.9218   0.9355   0.0579
   0.7382   0.9169   0.3529
   0.1763   0.4103   0.8132
   0.4057   0.8936   0.0099
```

```
>> a=fix(rand(4,3)*10)
a =
     8     5     7
     5     4     9
     3     6     5
     7     6     8
```

```
>> t=fix(rand(5,3)*25+1)
t =
     6    12    24
     3    10    25
     3    20     5
     2    16     4
    11    20    18
```

Funciones especiales para matrices

```
>> a=[2, 4, 5; 6, 0, 7; -2, 8, 3; 9, 2, 8]
a =
     2     4     5
     6     0     7
    -2     8     3
     9     2     8
```

Obtener las dimensiones de una matriz

```
>> [n,m]=size(a)
n =
     4
m =
     3
```

Suma de columnas

```
>> c=sum(a)
c =
    15    14    23
```

Suma de filas

```
>> f=sum(a')
f =
    11    13     9    19
```

a' es la transpuesta de **a**

Suma de todos los elementos

```
>> s=sum(f)
s =
   52
```

El mayor valor de cada columna

```
>> max(a)
ans =
    9    8    8
```

El mayor valor de cada columna y su posición de fila

```
>> [z,p]=max(a)
z =
    9    8    8
p =
    4    3    4
```

El mayor valor de cada fila

```
>> e=max(a')
e =
    5    7    8    9
```

El mayor valor de cada fila columna y su posición en la columna

```
>> [z,p]=max(a')
z =
    5    7    8    9
p =
    3    3    2    1
```

Determinar si un elemento se encuentra en una matriz

```
>> a=[2, 4, 5; 6, 0, 8; -2, 8, 3; 9, 2, 7]
a =
    2    4    5
    6    0    8
   -2    8    3
    9    2    7
>> e=ismember(8,a)
e =
    1
```

Determinar si un elemento se encuentra en una matriz y su posición

```
>> [e,p]=ismember(8,a)
e =
    1
p =
   10
```

Es la posición de la última aparición del número, contando por columnas desde el inicio

Determinar si los elementos de un vector se encuentran en una matriz

```
>> e=ismember([5, 1, 7, 9],a)
e =
    1    0    1    1
```

El resultado es un vector de coincidencias

Determinar si una fila se encuentra en una matriz

```
>> e=ismember([5, 1, 7],a,'rows')
e =
    0
>> e=ismember([6 0 7],a,'rows')
e =
    1
```

Determinar **cuales** elementos de una matriz son iguales a un elemento dado

```
>> a=[3 7 5; 2 8 9; 8 7 6; 4 4 8]
a =
    3    7    5
    2    8    9
    8    7    6
    4    4    8

>> t=ismember(a,8)
t =
    0    0    0
    0    1    0
    1    0    0
    0    0    1
```

El resultado es una matriz de coincidencias

Determinar cuantos elementos de una matriz son iguales a un elemento dado

```
>> a=[3 7 5; 2 8 9; 8 7 6; 4 4 8]
a =
    3    7    5
    2    8    9
    8    7    6
    4    4    8
>> t=ismember(a,8)
t =
    0    0    0
    0    1    0
    1    0    0
    0    0    1
>> n=sum(sum(t))
n =
     3
```

Elementos de la diagonal de una matriz

```
>> a=[2, 4, 5; 6, 9, 7; -2, 8, 3]
a =
    2    4    5
    6    9    7
   -2    8    3

>> d=diag(a)
d =
    2
    9
    3
```

Construir una matriz diagonal con un vector

```
>> e=diag(d)
e =
    2    0    0
    0    9    0
    0    0    3
```

Matriz triangular superior

```
>> t=triu(a)
t =
    2    4    5
    0    9    7
    0    0    3
```

Matriz triangular inferior

```
>> r=tril(a)
r =
     2     0     0
     6     9     0
    -2     8     3
```

Suma de los elementos de la matriz triangular superior

```
>> s=sum(triu(a))
s =
     2    13    15
```

Ordenamiento de una matriz por columnas

```
>> a=[2, 4, 5; 6, 0, 7; -2, 8, 3; 9, 2, 8]
a =
     2     4     5
     6     0     7
    -2     8     3
     9     2     8

>> b=sort(a)
b =
    -2     0     3
     2     2     5
     6     4     7
     9     8     8
```

Funciones especiales para matrices

Determinante

```
>> b=[5, 3, 2; 0, 8, 1; 2, 5, 4]
b =
     5     3     2
     0     8     1
     2     5     4

>> d=det(b)
d =
   109
```

Inversa

```
>> c=inv(b)
c =
    0.2477   -0.0183   -0.1193
    0.0183    0.1468   -0.0459
   -0.1468   -0.1743    0.3670
```

Producto de una matriz por su inversa

```
>> d=b*c
d =
     1     0     0
     0     1     0
     0     0     1
```

Resolución de un sistema de ecuaciones lineales: $AX = B \Rightarrow X = A^{-1}B$

```
>> a=[2 5 7; 4 6 2; 8 9 3]
a =
     2     5     7
     4     6     2
     8     9     3
```

```
>> b=[4;5;6]
b =
     4
     5
     6

>> x=inv(a)*b
x =
   -0.7500
    1.4063
   -0.2187
```

Reducción a forma escalonada canónica

```
>> a=[2 5 7 4; 4 6 2 5; 8 9 3 6]
a =
     2     5     7     4
     4     6     2     5
     8     9     3     6

>> rref(a)
ans =
    1.0000         0         0   -0.7500
         0    1.0000         0    1.4063
         0         0    1.0000   -0.2188
```

Valores propios de una matriz

```
>> c=eig(a)
c =
  -0.8910 + 2.8862i
  -0.8910 - 2.8862i
  15.7820
```

Operaciones con matrices

```
>> a=[2, 4, 5; 6, 0, 7; -2, 8, 3]
a =
     2     4     5
     6     0     7
    -2     8     3

>> b=[5, 3, 2; 0, 8, 1; 2, 5, 4]
b =
     5     3     2
     0     8     1
     2     5     4

>> c=a+b
c =
     7     7     7
     6     8     8
     0    13     7

>> c=a*b
c =
    20    63    28
    44    53    40
    -4    73    16
```

```
>> d=2*c
d =
   40   126    56
   88   106    80
   -8   146    32

>> e=d+3
e =
   43   129    59
   91   109    83
   -5   149    35
```

Transpuesta de una matriz

```
>> a=[2, 4, 5; 6, 0, 7; -2, 8, 3]
a =
    2    4    5
    6    0    7
   -2    8    3

>> u=a'
u =
    2    6   -2
    4    0    8
    5    7    3
```

Producto punto entre matrices

```
>> a=[2, 4, 5; 6, 0, 7; -2, 8, 3]
a =
    2    4    5
    6    0    7
   -2    8    3

>> b=[5, 3, 2; 0, 8, 1; 2, 5, 4]
b =
    5    3    2
    0    8    1
    2    5    4
>> c=a.*b
c =
   10   12   10
    0    0    7
   -4   40   12
```

7.2 Algoritmos con matrices

Inicialmente de desarrollarán programas para emular algunas funciones existentes en MATLAB. Con este conocimiento se podrá enfrentar el desarrollo de nuevos programas

Ejemplo. Escriba un programa equivalente a la función **sum** de MATLAB. El resultado es un vector con las suma de las columnas

```
>> a=[2, 4, 5; 6, 0, 7; 2, 8, 3]
a =
    2    4    5
    6    0    7
    2    8    3
>> s=sum(a)
s =
    10    12    15
```

```
%Suma de las columnas de una matriz
a=input('Ingrese la matriz ');
[n,m]=size(a);
s=[ ];
for j=1:m                          %Para cada columna se realiza la suma de filas
   t=0;
   for i=1:n
      t=t+a(i,j);
   end
   s=[s, t];
end
disp('Suma de columnas');
disp(s);
```

Interacción

```
Ingrese la matriz  a
Suma de columnas
s =
    10    12    15
```

Ejemplo. Escriba un programa equivalente a la función **max** de MATLAB.
El resultado es un vector con el mayor valor de cada columna

```
>> a=[2, 4, 5; 6, 0, 7; 2, 8, 3]
a =
    2    4    5
    6    0    7
    2    8    3

>> r=max(a)
r =
    6    8    7
```

Algoritmo: Obtención del mayor valor de cada columna de una matriz

Entrar la matriz **a**
Obtenga las dimensiones **n, m** de la matriz **a**
Inicie el vector de resultados **r** como un vector nulo
Para j=1,2,3,…,m Cada columna de la matriz a
  t ← $a_{1,j}$ Coloque el primer elemento de la columna **j** en la variable **t**
  Para i=2,3,4,...,n Compare los siguientes valores de la columna **j,** con el valor **t**
    Si $a_{i,j}$> t Si es mayor a **t**, reemplace al valor de **t**
      t←$a_{i,j}$
    fin
  fin
  Agregue el valor final de **t** al vector **r**
fin
Mostrar r Vector de resultados

Lenguaje MATLAB

```
%Obtención de un vector con los mayores valores de las columnas de una matriz
a=input('Ingrese la matriz ');
[n,m]=size(a);
r=[ ];
for j=1:m
   t=a(1,j);
   for i=2:n
      if a(I,j) > t
         t=a(i,j);
      end
   end
   r=[r, t];
end
disp('El mayor por columnas');
disp(r);
```

Interacción

Ingrese la matriz **a**
El mayor por columnas
r =
    10   12   15

Ejemplo. Suma de los elementos con valor par de una matriz

```
a=input('Ingrese la matriz ');
[n,m]=size(a);
s=0;
for i=1:n
   for j=1:m
      if mod(a(i,j),2) == 0
         s=s + a(i,j);
      end
   end
end
disp('la suma es')
disp(s);
```

Interacción

ingrese la matriz **a**
la suma es 22

Ejemplo. Sume los elementos de una matriz triangular inferior

La respuesta se la puede encontrar directamente sumando los elementos de la matriz triangular inferior obtenida con la función **sum(tril( )).** El siguiente programa producirá el mismo resultado:

```
a=input('ingrese matriz ');
[n,m]=size(a);
s=0;
for i=1:n                          %Sumar cada elemento  a_{i,j}  para el cual  i≤j
   for j=1:n
      if i<=j
         s=s+a(i,j);
      end
   end
end
disp(s)
```

Otra solución

```
a=input('ingrese matriz ');
[n,m]=size(a);
s=0;
for i=1:n
   for j=1: i                  %Mediante los índice se suman los elementos
       s=s+a(i,j);
   end
end
disp(s)
```

Ejemplo. Convertir una matriz a un vector fila

```
a=input('ingresar matriz ');
[n,m]=size(a);
v=[ ];
for i=1:m
   for j=1:n
      v=[v, a(i,j)];
   end
end
disp(v);
```

Ejemplo. Convertir una vector a una matriz compatible.

```
v=input('ingrese vector ');
n=input('cuantas filas ');
m=input('cuantas columnas ');
if n*m~=length(v)
   disp('Incompatible');
else
   a=[ ];                                  %para eliminar datos anteriores
   k=1;
   for i=1:n
      for j=1:m
         a(i,j)=v(k);
         k=k+1;
      end
   end
   disp(a);
end
```

Ejemplo. Leer el cuadro de goles de un campeonato de fútbol y producir el cuadro de resultados en puntos. Los goles están anotados por filas.

Algoritmo

Entrar g (matriz cuadrada con el cuadro de goles)
Detemine la dimensión **n**
Inicie en ceros la matriz de resultados **r**
Para cada elemento **i, j** de la matriz g
si g(i,j) = g(j,i) — Elementos simétricos iguales: empate
r(i,j)=1
r(j,i)=1
sinó
si g(i,j) > g(j,i) — El equipo **i** gana al equipo **j**
r(i,j)=3
r(j,i)=0
sinó — Caso contrario
r(i,j)=0 — el equipo j gana al equipo i
r(j,i)=3
fin

fin
fin
Mostrar el cuadro de resultados **r**

```
%Leer cuadro de goles y mostrar cuadro de resultados
g=input('Ingrese el cuadro de goles ');
[n,n]=size(g);
r=zeros(n,n);
for i=1:n
   for j=1:n
      if i~=j
         if g(i,j)==g(j,i)
            r(i,j)=1;
            r(j,i)=1;
         else
            if g(i,j)>g(j,i)
               r(i,j)=3;
               r(j,i)=0;
            else
               r(i,j)=0;
               r(j,i)=3;
            end
         end
      end
   end
end
disp('Cuadro de resultados');
disp(r);
```

Interacción

```
>> g=fix(5*rand(6,6));
>> g=g-diag(diag(g))
g =
    0    4    0    3    2    2
    1    0    0    3    3    1
    2    2    0    3    2    0
    0    0    0    0    4    3
    4    2    2    0    0    3
    4    1    2    1    4    0
```

Tabla de goles (aleatoria)
Para eliminar la diagonal

```
>> prueba1
Ingrese el cuadro de goles  g
Cuadro de resultados
    0    3    0    3    0    0
    0    0    0    3    3    1
    3    3    0    3    1    0
    0    0    0    0    3    3
    3    0    1    0    0    0
    3    1    3    0    3    0
```

Ejemplo. Leer el cuadro de resultados de un campeonato de fútbol y producir la lista del o los ganadores. Los resultados están tabulados por filas. Este programa es el complemento del programa anterior.

Algoritmo

Entrar **r** (matriz cuadrada con el cuadro de resultados)
Determinar la dimensión **n** de la matriz **r**
Sumar las filas y almacenar los resultados en el vector **s**
Asignar a **t** el máximo puntaje
Almacenar en el vector v todas las filas cuyo puntaje coincida con el valor **t**
Mostrar el contenido del vector **v** (Si hay más de uno, significa que hay empate)

```
%Leer el cuadro de resultados del campeonato de fútbol y mostrar el(o los) ganadores
r=input('Ingrese la tabla de resultados ');
[n,n]=size(r);
s=sum(r');                  %Para sumar filas se usa la transpuesta de  r
t=max(s);                   %Vector con el mayor valor de la suma
v=[ ];
for i=1:n
   if s(i)==t
      v=[v, i];             %Vector con equipos que tienen el mayor puntaje
   end
end
if length(v)==1             %Un solo ganador
  disp('Equipo ganador');
  disp(v);
else                        %Más de un equipo con el mayor puntaje
  disp('Empate entre los equipos');
  disp(v);
end
```

Interacción Resultado de la ejecución del programa anterior

```
r =
    0    3    0    3    0    0
    0    0    0    3    3    1
    3    3    0    3    1    0
    0    0    0    0    3    3
    3    0    1    0    0    0
    3    1    3    0    3    0


Tabla de resultados  r
Empate entre los equipos
    3    6
```

Ejemplo. Lea un cuadro de asignación de los paralelos de cada materia, a profesores. Suponer que cada profesor puede tener solo un paralelo de cada materia. Determine:
a) El total de paralelos asignados a cada profesor
b) Lista para cada materia de profesores y paralelos asignadoss

En el cuadro, las filas representarán profesores y las columnas representarán las materias. En el cuadro están los paralelos asignados.

```
a=input('Ingrese matriz de asignación ');
[n,m]=size(a);
% Cantidad de paralelos por profesor
disp('Prof. No. Paralelos')
for  i=1:n                    %asignación por profesor
   c=0;
   for j=1:m
      if a(i,j)>0
         c=c+1;
      end
   end
   disp([i, c]);
end

%Para cada materia, mostrar el profesor y el paralelo asignado
disp('Mat. Profesor Paralelo')
for j=1:m                     %asignación por materia
   for i=1:n
      if a(i,j)>0
         disp([j, i, a(i,j)]);
      end
   end
end
```

Interacción

```
>> a=[0 2 1 3; 1 1 0 0; 3 3 0 1; 2 0 2 2]
a =
    0    2    1    3
    1    1    0    0
    3    3    0    1
    2    0    2    2

Ingrese matriz de asignación  a
Prof. No. Paralelos
    1    3
    2    2
    3    3
    4    3
Mat. Profesor Paralelo
    1    2    1
    1    3    3
    1    4    2
    2    1    2
    2    2    1
    2    3    3
    3    1    1
    3    4    2
    4    1    3
    4    3    1
    4    4    2
```

Ejemplo. Generar una tabla ordenada de 5 filas y tres columnas con números aleatorios entre 1 y 25.

```
v=[ ];
while length(v)<15
   x=fix(rand*25)+1;
   if ismember(x,v)==0
      v=[v,x];
   end
end
v=sort(v);
k=1;
t=[ ];
for i=1:5
   for j=1:3
      t(i,j)=v(k);
      k=k+1;
   end
end
disp(t)
```

Interacción

```
 5    6    8
 9   10   12
13   14   15
16   17   20
21   22   24
```

Ejemplo. Generar el triángulo de Pascal de n filas

```
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
etc.
```

```
n=input('cuantas filas ');
for i=1:n
   a(i,1)=1;
   a(i,i)=1;
end
for i=3:n
   for j=2:i-1
      a(i,j)=a(i-1,j-1)+a(i-1,j);
   end
end
disp(a)
```

La asignacion de la primera columna y de la diagonal con 1's puede hacerse tambien directamente con las instrucciones:

```
a=eye(n);
a(:,1)=1;
```

Ejemplo. Definir y procesar una matriz con dígitos binarios

```
n=input('tamaño de la matriz ');
a=fix(rand(n,n)*2);
% filas con número par de 1's
for i=1:n
   s=sum(a(i,:));
   if mod(s,2)==0
      disp(i);
   end
end

% fila con más 1's
t=[ ];
for i=1:n
   s=sum(a(i,:));
   t=[t, s];
end
[m,p]=max(t);
disp('fila con más unos');
disp(p);

%convertir a decimal
disp('números en decimal');
for i=1:n
   x=0;
   for j=1:n
      x=x+a(i,j)*2^(n-j);
   end
   disp(x);
end
```

Ejemplo. Escriba un programa que lea las coordenadas con la ubicación de un caballo en un tablero de ajedrez (8x8) casillas. Muestre como resultado una matriz con las celdas marcadas con el valor 1 para indicar las casillas a las que se puede mover el caballo.

El caballo puede moverse entre casillas, dos en una dirección y una en una dirección perpendicular. Considere los límites del tablero.

```
f=input('Número de fila del caballo ');
c=input('Numero de columna del caballo ');
t=zeros(8,8);
x=[-2 -1  1  2  2  1 -1 -2];   %sumar a la fila f
y=[ 1  2  2  1 -1 -2 -2 -1];   %sumar a la columna c
for i=1:8
   pf=f+x(i);
   pc=c+y(i);
   if pf>=1 & pf<=8 & pc>=1 & pc <=8
      t(pf,pc)=1;
   end
end
disp(t);
```

Ejemplo. Escriba un programa para control de venta de las sillas de un evento con las siguientes opciones:

1) Verificar silla disponible
2) Venta de silla
3) Devolución de silla

La ubicación de las sillas será registrada con una matriz

```
n=input('cuantas filas ');
m=input('cuantas columnas ');
s=zeros(n,m);
x=0;
while x~=4
   disp('1) verificar silla');
   disp('2) venta');
   disp('3) devolución');
   disp('4) salir');
   x=input('elija una opción ');
   switch x
      case 1
         f=input('cual fila ');
         c=input('cual columna ');
         if s(f,c)==0
            disp('silla disponible');
         else
            disp('silla ocupada');
         end
      case 2
         f=input('cual fila ');
         c=input('cual columna ');
         if s(f,c)==1
            disp('silla ocupada');
         else
            s(f,c)=1;
         end
      case 3
         f=input('cual fila ');
         c=input('cual columna ');
         s(f,c)=0;
   end
end
```

Ejemplo. Programa para control de los casilleros de un club. Los casilleros están organizados en filas y columnas.

Si el casillero está libre contiene cero. Si está asignado, contiene el código del usuario.

El manejo de los casilleros será realizado en una matriz

```
n=input('cuantas filas ');
m=input('cuantas columnas ');
c=zeros(n,m);
x=0;
while x~=5
  disp('1) Asignar casillero');
  disp('2) Devolver casillero');
  disp('3) Consultar casillero');
  disp('4) Consultar usuario');
  disp('5) Salir');
  x=input('Elija una opción ');
  switch x
     case 1
        i=input('cual fila ');
        j=input('cual columna ');
        if c(i,j)~=0
           disp('casillero ocupado');
        else
           s=input('ingrese el código del usuario ');
           c(i,j)=s;
        end
     case 2




     case 3




   end
end
```

Completar las opciones

7.3 Matrices dispersas (sparse matrices)

Son matrices especiales en las que solamente se asigna memoria a los elementos diferentes de cero. El uso de estas matrices es importante en aplicaciones que requieren matrices muy grandes pero en las cuales la mayoría de los elementos son ceros. En estos casos, la defínción de una matriz de tipo dispersa representa un ahorro considerable de memoria pues solamente se asignan memoria a los elementos que no son ceros.

Cada elemento de una matriz dispersa se define con tres componentes: número de fila, número de columna y contenido.

Ejemplo. Defina una matriz dispersa incialmente con 100 por 100 elementos nulos

```
>> a=sparse(100,100);
```

Asignar algunos valores en las posiciones especificadas de la matriz a:

```
>> a(5,7)=32;
>> a(75,28)=43;
>> a(98,87)=73;
```

Al mostrar la matriz, solamente se muestran los elementos no nulos y su posición

```
>> a
a =
   (5,7)       32
  (75,28)      43
  (98,87)      73
```

Se ha asignado memoria unicamente a estos tres elementos. Los otros elementos son cero y no ha sido asighada memoria para su almacenamiento.

Los elementos de una matriz dispersa pueden operarse con la misma notación de indices

Ejemplo. Determine cuantos elementos no nulos contiene la matriz a anterior:

```
n=0;
for i=1:100
   for j=1:100
      if a(i,j)~=0
         n=n+1;
      end
   end
end
disp(n)
```

Si la matriz anterior fuese definida de la manera convencional, la cantidad de memoria asignada sería significativamente mayor.

Ejemplo. Realizar las mismas operaciones con una matriz común:

```
>> b=zeros(100,100);
>> b(5,7)=32;
>> b(75,28)=43;
>> b(98,87)=73;
```

Para comprobar la diferencia en la cantidad de memoria usada en los dos casos, se puede abrir la ventana workspace o usar el commando **whos**:

```
>> whos
  Name        Size              Bytes  Class     Attributes

    a         100x100             984  double    sparse
    b         100x100           80000  double
```

Se puede observar la gran diferencia en la cantidad de memoria usada.

La programación con matrices dispersas es similar a la programación común, pero existen algunas funciones específicas para manejo de estas matrices las cuales se pueden consultar en el sistema de ayuda del lenguaje.

7.4 Arreglos de celdas (cell arrays)

Este es un tipo especial de arreglos (vectores o matrices), cuyos elementos son dispositivos denominados celdas y que pueden contener vectores o matrices de diferente tipo.

Este tipo de datos permite coleccionar información de diferente tipo en un solo arreglo y manejarlo con un nombre.

Un arreglo de celdas se puede construir directamente asignando el contenido a sus componentes con la notación de llaves { } en lugar de los corchetes [ ] que se usan para definir vectores y matrices.

Ejemplo. Defina una matriz de celdas asignando a sus components datos de varios tipos:

```
>> a{1,1}=[5 7 8 2 3];
>> a{1,2}=eye(3,3);
>> a{2,1}='matemática';
>> a{2,2}=[3 7; 4 8; 2 6];
```

Los componentes de un arreglo de celdas se pueden manejar con la notación de llaves:

```
>> a{1,1}
ans =
    5    7    8    2    3
```

Con la notación de indices común, se puede accede al contenido de cada celda:

```
>> a{1,1}(3)
ans =
    8

>> a{2,1}(1:4)
ans =
mate

>> a{2,2}(2,1)
ans =
    4
```

7.5 Arreglos de celdas con cadenas de caracteres

Es conveniente usar arreglos de celdas para almacenar cadenas de caracteres que puedan tener diferente longitud, en vez de usar vectores comunes

Ejemplo. Almacenar algunos nombres en un vector de celdas

```
>> c{1}='Matemáticas';
>> c{2}='Física';
>> c{3}='Quimica';
```

Para manejar los componentes se usan llaves

```
>> c{1}
ans =
Matemáticas
```

El contenido de cada componente requiere la notación común de indices

```
>> c{1}(3:10)
ans =
temática
```

La longitud de de cada componente se ajusta a la longitud del dato almacenado

```
>> length(c{1})
ans =
   11
>> length(c{2})
ans =
    6
```

Se puede programar el manejo de los componentes con la notación de índice entre llaves

```
for i=1:3
   disp(c{i})
end
```

7.6 Ejercicios con matrices

1. Lea una matriz cuadrada. Descomponga la matriz en tres matrices: submatriz debajo de la diagonal, submatriz diagonal, y submatriz sobre la diagonal. Verifique que la suma de las tres matrices coincide con la matriz original. Muestre las matrices obtenidas

2. Lea una matriz nxm. Para cada fila, muestre el producto de los elementos cuyo valor es un número par.

3. Lea una matriz cuadrada. Muestre la suma de los elementos que no están en las dos diagonales principales.

4. Uno de los pasos que se requieren en los algoritmos para resolver un sistema de ecuaciones lineales consiste en intercambiar las filas de una matriz cuadrada para colocar en la diagonal principal los elementos de mayor magnitud de cada columna.

Escriba un programa que reciba una matriz cuadrada nxn, intercambie las filas desde arriba hacia debajo de tal manera que el elemento de mayor magnitud de cada columna se ubique en la diagonal y sustituya con ceros el resto de la fila hacia la derecha, como se muestra en el ejemplo.

$$\begin{bmatrix} 2 & 7 & 6 \\ 4 & 5 & 3 \\ 9 & 8 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{9} & 0 & 0 \\ 4 & 5 & 3 \\ 2 & 7 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 9 & 0 & 0 \\ 2 & \mathbf{7} & 0 \\ 4 & 5 & \mathbf{3} \end{bmatrix}$$

5. Lea una matriz nxn, siendo n un dato inicial. Suponga que cada celda contiene un dato (peso en kg.). Determine cuales son las celdas interiores en las cuales el valor del peso es mayor que el promedio de las cuatro celdas ubicadas a sus cuatro lados inmediatos, es decir que no debe considerar las celdas en los bordes.

6. Se dice que una matriz es 'Diagonal dominante' si en cada fila, el valor del elemento ubicado en la diagonal, es mayor a la cada uno de los otros elementos de esa fila.

Escriba un programa que lea una matriz nxn y determine si es tipo 'Diagonal dominante'

7. Escriba un progrma que lea una matriz nxn y dos números de fila. Intercambie las filas indicadas en los números y muestre la matriz resultante

8. Lea un vector **x** de **n** componentes. Construya y muestre la matriz **D** según la siguiente definición indicada con el ejemplo

x=[2 3 5 4],

$$\mathbf{D} = \begin{bmatrix} \mathbf{2^3} & \mathbf{2^2} & \mathbf{2^1} & \mathbf{1} \\ \mathbf{3^3} & \mathbf{3^2} & \mathbf{3^1} & \mathbf{1} \\ \mathbf{5^3} & \mathbf{5^2} & \mathbf{5^1} & \mathbf{1} \\ \mathbf{4^3} & \mathbf{4^2} & \mathbf{4^1} & \mathbf{4} \end{bmatrix}$$

9. Escriba un programa para controlar la cantidad de contenedores en un patio. Ingrese como dato la cantidad inicial y ofrezca las siguientes opciones:

1. Salida de contenedores
2. Llegada de contenedores
3. Cantidad actual de contenedores
4. Terminar el control

En cada repetición el operador elige la opción ingresando el número y la cantidad de contenedores.

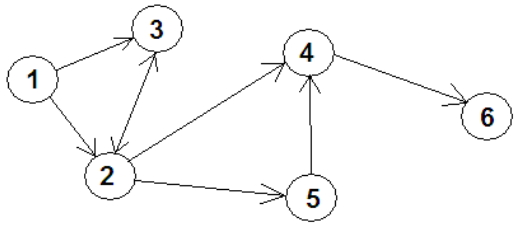
10. Escriba un programa para controlar el uso de los camiones de una empresa. Cada camión tiene un código. Ingrese como dato inicial la lista de los códigos de los camiones. Programe una aplicación con las siguientes opciones:

1. Salida de un camión
2. Devolución de un camión
3. Disponibilidad de un camión
4. Terminar

El operador elige la opción ingresando el número.

11. El área de un patio está distribuida en celdas ordenadas en filas y columnas. En cada celda están almacenados paquetes del mismo tipo.
Escriba un programa que lea una matriz cuyo contenido representa la cantidad de paquetes ubicados en cada celda
a) Encuentre el valor promedio de la cantidad de paquetes existentes en todas las celdas.
b) Encuentre cual celda contiene más paquetes.
c) Muestre las coordenadas y la cantidad de paquetes que contiene una celda cuya posición: número de fila y número de columna, son valores elegidos al azar en el programa..

12. Un grafo consta de vértices que pueden representarse mediante círculos y arcos que los conectan. Esta conectividad puede describirse mediante una matriz en la que el valor **1** indica que existe un arco en esa dirección, mientras que el valor **0** indica que no existe el arco con esa dirección, como se muestra en el ejemplo



Matriz de conectividad

	1	**2**	**3**	**4**	**5**	**6**
1	1	1	1	0	0	0
2	0	1	1	1	1	0
3	0	1	1	0	0	0
4	0	0	0	1	0	1
5	0	0	0	1	1	0
6	0	0	0	1	0	1

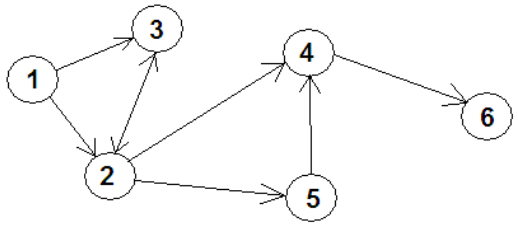
Escriba un programa para almacenar 0 o 1 aleatoriamente en una matriz nxn, siendo n un dato que debe leerse inicialmente. Dentro del programa llene la diagonal con 1's para indicar que cada nodo está conectado consigo mismo. El programa examinar las filas para determinar
a) Cual nodo no tiene conecciones con otros nodos (no tiene arcos)
b) Cual es el nodo que tiene más conecciones con otros nodos

13. Diseñe un programa para administrar el uso de los casilleros de una institución. Los casilleros están organizados en n filas y m columnas. Inicialmente los casilleros contienen el valor cero, lo cual significa que están vacíos. El programa debe usar un menú con las siguientes opciones:

1) Consultar casillero
2) Asignar casillero
3) Devolver casillero
4) Buscar usuario
5) Salir

En la opción 1, verificar si el casillero contiene cero, mostrar mensaje DISPONIBLE u OCUPADO
En la opción 2, almacenar en el casillero el código del usuario asignado
En la opción 3, almacenar cero en el casillero que es devuelto
En la opción 4, ingresar el código de algún usuario. Buscar la ubicación del casillero asignado.

14. Un grafo consta de vértices que pueden representarse mediante círculos y arcos que los conectan. Esta conectividad puede describirse mediante una **matriz** en la que el valor **1** indica que existe un arco en esa dirección, mientras que el valor **0** indica que no existe el arco con esa dirección, como se muestra en el ejemplo



Matriz de conectividad

	1	**2**	**3**	**4**	**5**	**6**
1	1	1	1	0	0	0
2	0	1	1	1	1	0
3	0	1	1	0	0	0
4	0	0	0	1	0	1
5	0	0	0	1	1	0
6	0	0	0	1	0	1

Escriba un programa para manejar interactivamente la conectividad de un grafo mediante un menú con las siguientes opciones de un menú:

1) Agregar arco
2) Eliminar arco
3) Consultar arco
4) Vértices libres
5) Salir

Al inicio debe pedir el número de vértices n. Llenar con **1** la diagonal y **0** en el resto de la matriz

En la opción 1) debe pedir los vértices inicial y final, y colocar **1** en la celda de la matriz ubicada en la fila y columna respectivas.

En la opción 2) debe pedir los vértices inicial y final, y colocar **0** en la celda de la matriz ubicada en la fila y columna respectivas.

En la opción 3) debe pedir los vértices inicial y final, y mostrar un mensaje dependiendo del contenido de la celda de la matriz ubicada en la fila y columna respectivas

En la opción 4) busque cada fila (vértice inicial) que tiene todas las columnas con ceros (vértices finales), excepto la diagonal

8 Funciones en MATLAB

En general una función en los lenguajes de programación es un conjunto de instrucciones que se escriben separadamente del programa y que realizan alguna tarea especificada. Los usuarios pueden definir funciones y agregarlas a las funciones propias de MATLAB.

El mecanismo usual para trasmitir datos a las funciones es mediante una lista de variables que se denominan parámetros. Sin embargo, a diferencia de los programas, las variables que se usan dentro de una función, no están disponibles fuera de ella, a menos que se use una declaración explícita y que se verá mas adelante.

Declaración de una función en MATLAB

```
function  variable = nombre (parámetros)
    instrucciones
```

variable contendrá el resultado que entrega la función
parámetros son variables que reciben los datos que entran a la función.
nombre identificación de la función
instrucciones se incluyen en la función según la tarea especificada

El nombre asignado a una función debe coincidir con el nombre usado en la declaración de la función.

Las funciones se escriben en la ventana de edición de MATLAB y se las almacena en alguna carpeta. En la ventana de comandos debe especificarse la ruta a esta carpeta.

El uso de una función es similar al uso de las funciones comunes de MATLAB. El nombre debe coincidir con el nombre asignado, aunque los parámetros pueden tener nombres diferentes, pero su uso debe ser coherente.

El concepto de función es fundamental en matemática y MATLAB proporciona un dispositivo natural para su instrumentación

Ejemplo: Instrumentar en MATLAB la siguiente función de variable real:

$$y = f(x) = 2x^2 + 1$$

Instrumentación en MATLAB

```
function y=f(x)
y=2*x^2 + 1;
```

Usar en la ventana de comandos la función creada:

```
>> y=f(2)
    y = 9
```

Es importante distinguir entre un programa y una función. El siguiente ejemplo debe ayudar a entender esta diferencia:

Ejemplo. Escriba un programa que **lea** dos datos y **muestre** como resultado, el mayor valor

```
a=input('Ingrese el primer dato ');
b=input('ingrese el segundo dato ');
if a>b
  m = a;
else
  m = b;
end
disp('El mayor es');
disp(m);
```

Normalmente un programa **lee** datos, y **muestra** los resultados calculados

Ejemplo. Escriba una función que **reciba** dos datos y **entregue** como resultado el mayor valor

```
function m = mayor(a, b)
if a>b
  m = a;
else
  m = b;
end
```

Normalmente una función **recibe** los datos en variables (se denominan parámetros) y **entrega** el resultado mediante alguna variable, por lo tanto no necesita leer los datos o mostrar el resultado.

Para este ejemplo:

m	es la variable que entrega el resultado
mayor	es el nombre de la función
a, b	son los parámetros que ingresan los datos a la función

La función debe guardarse en un archivo y debe tener el mismo de la función

Uso de la función en la ventana de comandos:

Suponer que quiere escoger el mayor entre e^{π} y π^{e}.

```
>> a = exp(pi);
>> b = pi^exp(1);
>> m = mayor(a, b)
        23.1407          (respuesta que muestra MATLAB)
```

Los nombres de las variables pueden ser diferentes:

```
>> x = exp(pi);
>> y = pi^exp(1);
>> t = mayor(x, y)
        23.1407          (respuesta que muestra MATLAB)
```

Ejemplo. Escriba una función que reciba un número y determine la cantidad de divisores enteros exactos que tiene el número.

```
function  c = ndiv( n )
c = 0;
for d = 1: n
  if mod(n, d) == 0
    c = c + 1;
  end
end
```

Guarde la función en el disco con el nombre **ndiv**
Pruebe la función directamente desde la ventana de comandos

```
>> n = 25;
>> c = ndiv(n)
        3                    (resultado que muestra MATLAB)
>> c = ndiv(43)
        2                    (resultado que muestra MATLAB)
```

Ejemplo. Escriba una función que reciba un número y determine si es un número primo. El resultado que entrega la función será 1 o 0 según corresponda;

```
function  p = primo( n )
c = 0;
for d = 1: n
  if mod(n, d) == 0
    c = c + 1;
  end
end
if c > 2
  p = 0;
else
  p = 1;
end
```

Guarde la función en el disco con el nombre **primo**
Pruebe la función directamente desde la ventana de comandos

```
>> n = 25;
>> p = primo(n)
        0                    (resultado que muestra MATLAB)
>> p = primo(43)
        1                    (resultado que muestra MATLAB)
```

Ejemplo. Escriba en una nueva ventana de edición un programa que use la función **primo** para encontrar todos los números primos menores a 20:

```
for n = 1: 20
  if primo(n) == 1
    disp(n);
  end
end
```

Interacción

```
        1                    (resultados mostrados por MATLAB)
        2
        3
        5
        7
```

```
11
13
17
19
```

Ejemplo. Escriba un programa que use la función **primo** para encontrar todas las parejas de números primos cuya suma sea igual a un número par dado:

```
n=input('ingrese un numero par');
for a=1:n
   for b=a:n
      if primo(a)==1 & primo(b)==1 & a+b == p
         disp([a,b]);
      end
   end
end
```

Prueba del programa

```
ingrese un numero 50
   3    47
   7    43
  13    37
  19    31
```

Si es necesario verificar que el dato es un número par se puede incluir una validación:

```
res=1;
while res~=0
    n=input('ingrese un numero par');
    res=mod(n,2);
end
for a=1:n
   for b=a:n
      if primo(a)==1 & primo(b)==1 & a+b == p
         disp([a,b]);
      end
   end
end
```

Una función puede entregar más de un resultado

Las variables que entregan los resultados deben definirse entre corchetes.

Ejemplo. Escriba una función que entregue el área y el volumen de un cilindro dados su radio (r) y su altura (h)

```
function [a, v] = cilindro(r, h)
a = 2*pi*r*h + 2*pi*r^2;
v = pi*r^2*h;
```

Escriba y almacene la función con el nombre cilindro.

Use la función para calcular el área y el volumen de una lata de cilíndrica que tiene un diámetro de 10cm y una altura de 12cm

Escriba en la ventana de comandos:

```
>> r = 5;
>> h = 12;
>> [a, v] = cilindro(r,h)        Se muestran ambos resultados
>> a = cilindro(r,h)             Se muestra el primer resultado
```

8.1 Variables locales y variables globales

Las variables definidas dentro de una función son locales, es decir que a diferencia de los programas, no son visibles fuera de la función

Ejemplo. Escriba la función:

```
function  x=fn(a, b)
c = a + b;
x = 2*c;
```

Almacenar con el nombre **fn** y usar desde la ventana de comandos:

```
>> a = 3;
>> b = 5;
>> t = fn(a, b)
      t = 16          (resultado que muestra MATLAB)
>> c                  (intentamos conocer el valor de c en la función)
```

??? Undefined function or variable 'c'.

mensaje de error de MATLAB que indica que **c** no está definida

Las variables en los programas son visibles (disponibles) fuera del programa.

Ejemplo. Un programa con acción similar a la función anterior:

```
a = input('ingrese dato ');
b = input('ingrese dato ');
c = a + b;
x = 2*c;
disp(x);
```

Almacene con el nombre prueba y active el programa:

```
>> prueba
      ingrese dato  3          (interacción para ingreso de datos)
      ingrese dato  5
        16                     (resultado que muestra MATLAB)
>> c
      c = 8                    (la variable c puede ser utilizada)
```

Es posible hacer que las variables de una función sean visibles fuera de su ámbito mediante una declaración especial

El comando **global** permite tener acceso a variables de las funciones

Ejemplo. Modifique la función **fn** para que la variable **c** sea visible:

```
function  x=fn(a, b)
global c;
c = a + b;
x = 2*c;
```

Almacene con el nombre fn y use la función:

```
>> global c;
>> a = 3;
>> b = 5;
>> t = fn(a, b)
        t = 16              (resultado que muestra MATLAB)
>> c                        (intentamos conocer el valor c de la función)
        c=8                 (la variable c está disponible ahora)
```

Una función puede no necesitar parámetros

Ejemplo. Escriba una función que lea y valide un entero entre 1 y 5

```
function n=entero
x=0;
while x==0
  n=input('ingrese un entero entre 1 y 5 ');
  if n>0 & n<6
    x=1;
  end
end
```

Una función puede no entregar resultados ni usar parámetros

Ejemplo. Escriba una función que muestre un menú en la pantalla

```
function menu1
clc;
disp('1) ingresar');
disp('2) borrar');
disp('3) salir');
```

para usar esta función escriba

```
>> menu1
```

Una función puede recibir como parámetros vectores o matrices.

Ejemplo. Escriba una función que reciba un vector y entregue el promedio del valor de sus elementos.

```
function p=prom(x)
n=length(x);
s=0;
for i=1:n
  s=s+x(i);
end
p=s/n;
```

Esta función es equivalente a la función **mean** de MATLAB

Para usar la función creada debe definir el vector antes de llamarla
La longitud del vector se determina con la función **length** de MATLAB

```
>> x=[2 7 3 5 4 7 6];
>> t=prom(x)
```

t = 4.8571 (es el resultado que muestra MATLAB)

Ejemplo. Escriba una función que reciba un vector y entregue la suma de los valores pares únicamente.

```
function s=sumapares(x)
n=length(x);
s=0;
for i=1:n
  if mod(x(i), 2) == 0
     s=s+x(i);
   end
end
```

```
>> x=[2 7 3 5 4 7 6];
>> t=sumapares(x)
```

t = 12 (es el resultado que muestra MATLAB)

Ejemplo. Una función convertir un entero positivo a su representación binaria:

```
function b=bin(n)
b='';
while n>0
   d=mod(n,2);
   n=fix(n/2);
   b=[num2str(d),b];
end
```

Use la función:

```
>> b=bin(789028)
   b =
        11000000101000100100
```

Una función puede recibir y entregar vectores o matrices

Ejemplo. Escriba una función que reciba un vector y entregue otro vector conteniendo los elementos cuyo valor es un número par.

```
function y=pares(x)
n=length(x);
y=[ ];
for i=1:n
  if mod(x(i), 2) == 0
     y = [y, x(i)];
   end
end
```

```
>> x=[2 7 3 5 4 7 6];
```

```
>> t=pares(x)
   t =
       2  4  6          (es el resultado que muestra MATLAB)
```

Ejemplo. Escriba una función que entregue un vector de longitud n conteniendo números aleatorios enteros con valor entre 1 y 6:

```
function d=dados(n)
for i=1:n
   d(i)=fix(rand*6+1);
end
```

Para usar esta función debe enviar un valor para el parámetro n:

```
>> t=dados(5)
        t = 6   3   4   3   2        (es un vector resultante de MATLAB)
```

Una solución alternativa para el ejemplo anterior:

```
function d=dados(n)
d=[ ];
while length(d) < n
   t =  fix(rand*6+1);
   d = [d, t]
end
```

Ejemplo. Escriba una función para emular la función **max** de MATLAB

```
function [m,p]=mayor(x)
n=length(x);
m=x(1);
p=1;
for i=2:n
   if x(i)>m
      m=x(i);
      p=i;
   end
end
```

```
>> x=[6 7 9 2 8 3];
>> m=mayor(x)
m =
    9
>> [m, p]=mayor(x)
m =
    9
p =
    3
```

Ejemplo. Escriba una función para emular la función **ismember** de MATLAB

```
function [r,p]=buscar(e,v)
n=length(v);
r=0;
p=0;
for i=1:n
   if e==v(i)
      r=1;
      p=i;
   end
end
```

```
>> v=[6 7 9 2 8 7 3];
>> r=buscar(7,v)
r =
    1
>> [r,p]=buscar(7,v)
r =
    1
p =
    6
```

Ejemplo. Modifique la función para entregar la posición del primer elemento si es encontrado.

```
function [r,p]=buscar(e,v)
n=length(v);
r=0;
p=0;
for i=1:n
   if e==v(i)
      r=1;
      p=i;
      break
   end
end
```

```
>> v=[6 7 9 2 8 7 3];
>> r=buscar(7,v)
r =
    1
>> [r,p]=buscar(7,v)
r =
    1
p =
    2
```

Ejemplo. Modifique la función para entregar un vector con las posiciones de los elementos encontrados y que son iguales al dato.

```
function p=buscarp(e,v)
n=length(v);
p=[ ];
for i=1:n
   if e==v(i)
      p=[p, i];
   end
end
```

```
>> v=[6 7 9 2 8 7 3];
>> p=buscarp(7,v)
p =
    2    6
```

Esta función también se puede usar con cadenas de caracteres, pues son almacenados en celdas, igual que los vectores

Ejemplo. Una función para eliminar elementos repetidos de un vector

```
function y=filtro(x)
n=length(x);
y=[ ];
for i=1:n
   if buscar(x(i),y)==0
      y=[y, x(i)];
   end
end
```

Ejemplo. Escriba una función que reciba dos conjuntos A, B y entregue un tercer conjunto que contenga los elementos que están en ambos conjuntos. Use vectores para almacenar los datos de los conjuntos.

```
function c=interseccion(a, b)
n=length(a);
m=length(b);
k=1;
for i=1:n
   for j=1:m
      if a(i) == b(j)
         c(k) = a(i);
         k = k + 1;
      end
   end
end
c=filtro(c);                %Elimina elementos repetidos
```

Para usar esta función debe definir los vectores que entran. Recuerde que pueden tener nombres diferentes a los que usa la función:

```
>> a=[2 7 5 4 3 8];
>> b=[7 1 3 9 0];
>> c=interseccion(a, b)
  c =    7    3          (Es el vector resultante que entrega MATLAB)
```

Una forma alterna para escribir la función anterior

```
function c=interseccion(a, b)
c=[ ];
for i=1: length(a)
   for j=1: length(b)
      if a(i) == b(j)
            c = [c , a(i)];
      end
   end
end
c=filtro(c);                %Elimina elementos repetidos
```

Una forma más eficiente para escribir la función anterior

```
function c=interseccion(a, b)
n=length(a);
c=[ ];
for i=1:n
   if buscar(a(i), b)
       c=[c, a(i)];
   end
end
c=filtro(c);                %Elimina elementos repetidos
```

Unión de conjuntos

```
function c=union(a,b)
m=length(b);
c=a;
for i=1:m
   if ~buscar(b(i),c)
      c=[c,b(i)];
   end
end
c=filtro(c);                %Elimina elementos repetidos
```

Versión más eficiente

```
function c=union(a,b)
c=[a, b];
c=filtro(c);
```

Diferencia de conjuntos

```
function c=diferencia(a, b)
n=length(a);
c=[ ];
for i=1:n
   if ~buscar(a(i), b)
       c=[c, a(i)];
   end
end
c=filtro(c);                %Elimina elementos repetidos
```

Ejemplo. Defina una función para elegir una muestra aleatoria de tamaño **n** de una población de tamaño **m.** Pueden haber elementos repetidos.

```
function t=muestra(m, n)
t=[ ];
for i=1:n
   x=fix(rand*m)+1;
   t=[t, x];
end
```

Ejemplo. Defina una función para elegir una muestra aleatoria de tamaño **n** de una población de tamaño **m**. Los elementos **no deben estar repetidos** en la muestra

```
function t=muestra(m,n)
t=[ ];
while length(x)<n
   x=fix(rand*m)+1;
   if ~buscar(x, t)
      t=[t, x];
   end
end
```

Uso de la función muestra

```
>> x=muestra(10,4)
x =
   5   10    9    6
```

Ejemplo. Escriba una función para obtener una tabla de 15 elementos (diferentes) con números aleatorios entre 1 y 25.

```
function t=tabla
v=[ ];
while length(v)<15
   x=fix(rand*25)+1;
   if buscar(x,v)==0
      v=[v, x];
   end
end
r=sort(v);
k=1;
for i=1:5
   for j=1:3
      t(i,j)=r(k);
      k=k+1;
   end
end
```

```
>> t=tabla
t =
    1    2    3
    4    7   11
   13   14   15
   16   17   18
   19   20   22
```

8.2 La instrucción **RETURN**

Si la salida de una función es antes del final, se debe salir con la instrucción **return**

Ejemplo. Defina una función para elegir una muestra aleatoria de tamaño **n** de una población de tamaño **m**. Los elementos **no deben estar repetidos** en la muestra. Valide **n**, **m**

```
function t=muestra(m,n)
t=[ ];
if n>m
   return
end
while length(x)<n
   x=fix(rand*m)+1;
   if ~buscar(x, t)
      t=[t, x];
   end
end
```

Ejemplo. Escriba una función para determinar si los elementos de un vector están en orden creciente. El resultado será **1** si los elementos están en orden creciente, caso contrario, el resultado será **0**

```
function t=chequeo(x)
n=length(x);
t=1;
for i=1:n-1
   if x(i) > x(i+1)
      t=0;
      return;
   end
end
```

Ejemplo. Una funcion para ordenar un vector en forma creciente. los datos de un vector

```
function r=ordenar(x)
n=length(x);
r=[ ];
while length(x)>0
   [e,p]=min(x);                    %Tomar el mayor valor y eliminar de  x
   r=[e,r];
   x(p)=[ ];
end
```

```
>> x=[7 9 3 8 4 5 2 6 7 8];
>> r=orden(x)
r =
  2  3  4  5  6  7  7  8  8  9
```

Ejemplo. Una funcion para ordenar un vector en forma creciente. Otra solución

```
function x=ordenar(x)
n=length(x);
for i=1:n
   [r,p]=min(x(i:n));
   t=x(i);                    %intercambiar elementos
   x(i)=x(p+i-1);             %corregir posición relativa
   x(p+i-1)=t;
end
```

Ejemplo. Una funcion para ordenar un vector en forma creciente. Solución detallada

```
function x=ordenar(x)
n=length(x);
for i=1:n-1
   for j=i+1:n
      if x(j)<x(i)                %intercambiar elementos
         t=x(i);
         x(i)=x(j);
         x(j)=t;
      end
   end
end
```

Ejemplo. Escriba una función **p=comb(r, s)** que reciba dos listas **r, s** que contienen números ordenados en forma ascendente. La función debe intercalar los números de ambas listas y entregar una tercera lista **p** con todos los datos ordenados en forma ascendente. No use la función SORT de MATLAB

```
function p=comb(r,s)
t=[r,s];
p=[ ];
while length(t)>0;
   [m,i]=min(t);
   p=[p, m];
   t(i)=[ ];
end
```

Ejemplo. Una función para eliminar espacios intermedios de una frase:

f: contiene una frase que entra a la función
x contiene la frase sin espacios intermedios

```
function x=compactar(f)
n=length(f);
x='';
for i=1:n
   if f(i) ~= ' '
      x = strcat(x, f(i));
   end
end
```

Una forma alterna de escribir la función anterior

```
function x=compactar(f)
n=length(f);
x='';
for i=1:n
   if f(i) ~= ' '
      x = [x, f(i)];
   end
end
```

8.3 Programas que llaman a funciones

Las funciones se escriben para ser usadas desde la ventana de comandos, pero son útiles también como instrumentos para facilitar la escritura de programas

Ejemplo. Un programa que lee una frase, usa la función **compactar** para eliminar los espacios intermedios, y luego muestra un mensaje en caso de que la frase sea simétrica: sus caracteres opuestos son iguales

```
f=input('ingrese una frase ');
f=compactar(f);
n=length(f);
sim=1;
for i=1:n/2
   if f(i) ~= f(n-i+1)
      sim=0;
   end
end
if sim == 1
   disp('la frase es simetrica');
else
   disp('la frase no es simetrica');
end
```

Probamos este programa suponiendo que lo hemos almacenado con el nombre **prueba**:

```
>> prueba
ingrese una frase 'anita lava la tina';        (dato que ingresa)

        la frase es simetrica                  (resultado de MATLAB)
```

MATLAB tiene la función **error** para desplegar mensajes de error y terminar la ejecución:

Ejemplo

.

```
if d<0
   error('valor incorrecto');
end
```

Ejemplo. Para cada número natural existe al menos otro número natural tal que el resultado de multiplicar estos dos números solamente tiene ceros y unos.
Ejemplo: El número **6** multiplicado por **185**, el resultado es **1110**
El número **38** multiplicado por **2895**, el resultado es **110010**
El número **253** multiplicado por **43917**, el resultado es **11111001**
Etc.
Escriba una función **n = factor(t)** que reciba un número natural **t** y entregue otro número natural **n**, tal que el producto **t** por **n** solamente tenga ceros y unos, como en los ejemplos anteriores.

```
function n=factor(t)
p=1;
n=1;
while p>0
   p=t*n;
   while p>0
      d=mod(p,10);
      if d>1
         n=n+1;
         break;
      else
         p=fix(p/10);
      end
   end
end
```

Ejemplo.

```
>> n=factor(237)
n =
     426203
>> n*237
ans =
  101010111
```

Este algoritmo es una búsqueda exhaustiva simple

8.4 Funciones recursivas

Una función puede llamarse a si misma. Estas funciones se denominan recursivas. El uso de la recursón es una técnica que se puede usar en programación para resolver problemas complejos. Una función recursiva normalmente se la describe mediante una decisión en la cual la misma función es invocada. La repetición se realiza internamente y no requiere definirse con un ciclo como en una solución convencional. Sin embargo, la recursión puede ser costosa en tiempo computacional si cada llamada genera más de una llamada a la misma función.

Ejemplo. Escriba una función para sumar los cubos de los primeros **n** números naturales.

$$\mathbf{sc(n) = 1^3 + 2^3 + 3^3 + ... + (n-1)^3 + n^3}$$

La manera usual de escribir esta función es mediante un ciclo explícito:

```
function r=sc(n)
 r=0;
for i=1:n
    r= r + i^3;
end
```

Sin embargo, la suma **sc(n)** se la puede expresar en forma recurrente, invocando la misma suma de términos sc, pero con un anterior a n:

$$\mathbf{sc(n) = sc(n-1) + n^3}$$

Esta definición requiere una regla para que la invocación no continue indefinidamente hacia atrás. Entonces, la definición correcta es:

$$\mathbf{sc(n)} = \begin{cases} \mathbf{sc(n-1) + n^3}, & \mathbf{n > 1} \\ \mathbf{1}, & \mathbf{n = 1} \end{cases}$$

Calcular sc(4) con la definición recursiva anterior:

$$sc(4) = sc(3) + 4^3 = sc(2) + 3^3 + 4^3 = sc(1) + 2^3 + 3^3 + 4^3 = 1 + 2^3 + 3^3 + 4^3 = 100$$

Algunos lenguajes, entre ellos MATLAB, admiten definir funciones recursivas.

Ejemplo. Escriba una función recursiva para calcular la suma de los cubos de los primeros n números naturales.

```
function r=sc(n)
if n>1
    r=sc(n-1)+n^3;
else
    r=1;
end
```

Use la función:

```
>> r=sc(4)
   r =
        100
```

La función desarrolla el ciclo internamente mediante una secuencia de llamadas a si misma, como en el ejemplo anterior

Ejemplo. La siguiente definición recursiva produce el máximo común divisor entre dos números enteros. Escriba una función con esta definición

$$mcd(a,b) = \begin{cases} mcd(a-b,b), & a > b \\ mcd(a,b-a), & b > a \\ a, & a = b \end{cases}$$

```
function c=mcd(a, b)
if a>b
    c=mcd(a-b, b);
else
    if b>a
        c=mcd(a, b-a);
    else
        c=a;
    end
end
```

Use la función:

```
>> x=mcd(36, 48)
```

Ejemplo. Una función para determinar cuantos dígitos tiene un número entero:

$$nd(n) = \begin{cases} nd(fix(n/10)), & n \geq 10 \\ 1, & n < 10 \end{cases}$$

```
function r=nd(n)
if n>=10
   r=nd(fix(n/10))+1;
else
   r=1;
end
```

Use la función:

```
>> r=nd(87963)
   r =
        5
```

Ejemplo. Una función recursiva para conteo de cifras de un entero positivo:

```
function c=nd(n)
if n>0
   c=nd(fix(n/10))+1;
else
   c=0;
end
```

Use la función:

```
>> c=bin(789028)
   c =
        6
```

Funciones que entregan resultados simbólicos

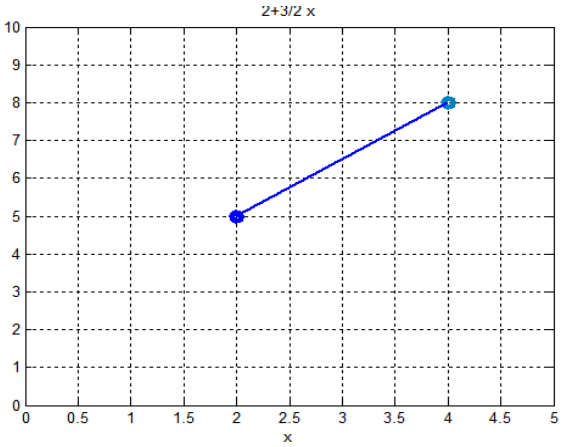
Ejemplo. Escriba y almacene una función que reciba dos puntos y entregue la ecuación de la recta que incluye a estos dos puntos:

Debido a que se manejará una expresión matemática en forma simbólica, las variables deben definirse con el tipo especial **syms**

```
function y=recta(x1, y1, x2, y2)
syms x;
m=(y2 - y1)/(x2 - x1);
y=y1 + m*(x - x1);
```

Uso de la función desde la línea de comandos y su gráfico

```
>> x1=2;
>> y1=5;
>> x2=4;
>> y2=8;
>> y=recta(x1, y1, x2, y2)
        y =
             2+3/2*x
>> plot(x1,y1,'o',x2,y2,'o'), hold on
>> ezplot(y, [2,4]), grid on
>> axis([0,5,0,10])
```



8.5 Ejercicios con funciones

1. Escriba una función **conteo(n)** que entregue la **cantidad** de divisores enteros positivos que tiene un número entero dado n. Escriba un programa de prueba que use la función escrita para encontrar cual número entre 1 y 100 tiene más divisores enteros.

2. Escriba una función **primo(n)** para determinar si un número **n** dado es primo. Escriba un programa de prueba que mediante la función escrita, encuentre los números primos existentes entre 1 y **n**, siendo **n** un dato.

3. Escriba una función **primo(n)** para determinar si un número **n** dado es primo. Escriba un programa de prueba que use la función primo y encuentre dos números enteros aleatorios menores que 100 tales que su suma sea también un número primo.

4. Escriba una función **divisores(n)** que entregue un vector conteniendo todos los divisores enteros positivos que tiene un número entero dado n. Escriba un programa de prueba que use la función escrita para encontrar para cada número entero del 20 al 30, sus divisores enteros

5. Escriba una función **mayor(x)** que reciba un vector **x** y devuelva el mayor valor. Escriba un programa de prueba que genere y almacene en un vector **n** números aleatorios entre 1 y 100. Use la función escrita y encuentre y muestre el mayor valor generado.

6. Escriba una función **perfecto(n)** que determine si un número entero dado n es un número perfecto. Un número perfecto debe ser igual a la suma de todos sus divisores enteros menores que el valor del número.

Ejemplo: 28 = 1 + 2 + 4 + 7 + 14

Escriba un programa de prueba que use la función escrita y encuentre los números perfectos entre 1 y 1000

7. Escriba una función **suma(n)** que entregue la suma de las cifras de un número dado **n**. Con esta función escriba un programa que genere 10 números aleatorios entre 1 y 100 y encuentre cual de ellos tiene la mayor suma de sus cifras.

8. Escriba una función **cuad(n)** que determine si el cuadrado de un número natural n dado, es igual a la suma de los primeros n números impares.

Ej. $6^2 = 1+3+5+7+9+11$

Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

9. Escriba una función **secuencia1(n)** que entregue el n-ésimo término de la siguiente secuencia, en la cual cada término, a partir del tercero se obtiene sumando los dos anteriores: 1, 1, 2, 3, 5, 8, 13, 21, ... Escriba un programa de prueba que ingrese un dato desde el teclado use la función y muestre el resultado en la pantalla.

15. Escriba una función **secuencia2(n)** que entregue el n-ésimo término de la siguiente secuencia, en la cual cada término, a partir del cuarto se obtiene sumando los tres anteriores: 1, 1, 1, 3, 5, 9, 17, 31, 57, ... Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

10. Escriba una función **conteo(x)** que reciba una cadena de caracteres x, y determine la cantidad de palabras que contiene. Suponga que las palabras están separadas por un espacio. Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

11. Escriba una función **sim(x)** que reciba un entero y determine si es simétrico, es decir si los dígitos opuestos alrededor del centro son iguales. Escriba un programa de prueba que genere números aleatorios entre 1 y 10000 hasta obtener un número que sea simétrico

12. Escriba una función **alfin(n)** que entregue como resultado la cantidad de veces que debe lanzarse un dado hasta que salga un número n dado como parámetro. Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

13. Escriba una función **conteo(x)** que determine la cantidad de términos que deben sumarse de la serie: 1*2*3 + 2*3*4 + 3*4*5 + 4*5*6+..... hasta que la suma exceda a un valor x dado. Escriba un programa de prueba que genere un número aleatorio para x entre 1 y 1000, use la función y muestre el resultado en la pantalla.

15. Escriba una función **fact(n)** que reciba un numero entero n y devuelva su factorial. Escriba un programa de prueba que genere un número aleatorio entero menor que 8, use la función y muestre la suma de los factoriales de los primeros k números naturales

16. Escriba una función **suma(n)** que reciba un numero entero n y devuelva la suma de sus dígitos. Escriba un programa de prueba que genere números aleatorio entre 1 y 100 hasta que la suma de los dígitos de alguno de ellos sea múltiplo de 7

17. Escriba una función **sumad(n)** que reciba un número entero n y devuelva la suma de sus divisores. Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla

18. Escriba una función **mcd(a, b)** que reciba dos números enteros a y b, y devuelva el máximo común divisor entre ellos. Escriba un programa de prueba que genere dos números aleatorios entre 1 y 100, use la función y muestre el resultado en la pantalla

19. Escriba una función **primos(v)** que reciba un vector **v** y entregue otro vector conteniendo los elementos que son números primos.

20. Escriba un a función **rango(v,a,b)** que reciba un vector **v**, y determine cuantos elementos están en el rango **[a, b]**

21. Escriba una función **codificar(x,k)** que reciba una cadena **x** y una constante **k** y entregue otra cadena con los caracteres desplazados **k** posiciones en el alfabeto. **k** puede ser positivo para codificar o negativo para decodificar

22. Escriba una función recursiva para obtener la cantidad de dígitos impares que contiene un número entero positivo.

23. Escriba una función recursiva para calcular el resultado un numero entero elevado a una potencia entera.

24. Escriba una función **polar(x, y)** que reciba las coordenadas cartesianas de un punto y entregue sus coordenadas polares **r, theta**

25. Escriba una función **cartesiana(r, theta)** que reciba las coordenadas polares de un punto y entregue sus coordenadas rectangulares **x, y**

26. En una expresión aritmética en notación INFIX se escriben los **operandos** (números) separados por un **operador** aritmético conocido (+, –, *, /).
En una expresión aritmética en notación POSTFIX, primero se escriben los operandos y luego el operador, como se muestra en los ejemplos.

INFIX	POSTFIX
2 + 3	2 3 +
9 – 6	9 6 –
5 * 4	5 4 *
8 / 7	8 7 /

Suponga que los operandos aritméticos son números de una sola cifra.

a) Escriba la función **infix** que reciba una cadena de 3 caracteres y verifique que está bien escrita en notación infix. La función devuelve 1 si es una cadena con la expresión infix válida y 0 si no lo es.
b) Escriba la función **postfix** que reciba una cadena de 3 caracteres, previamente validada con la función **infix** y cambie la expresión de notación INFIX a POSTFIX.
c) Escriba un programa de prueba para verificar que las funciónes producen resultados correctos.

27. Escriba una función que reciba un vector y entregue como resultado otro vector conteniendo los mismos elementos del vector ingresado pero con las elementos ubicados aleatoriamente en otro orden

Ejm. Entra **[3 7 6 2 9 8]**, sale **[6 8 3 2 7 9]**

Escriba un programa que llene un vector de **n** números aleatorios de una cifra. El programa debe enviar el vector a la función creada y recibir otro vector. El programa debe determinar cuantos números coinciden en la misma posición en ambos vectores

28. Escriba una función que reciba una matriz. La función debe debe entregar un vector con la cantidad de elementos pares que contiene cada columna de la matriz

Ejm. Entra $\begin{bmatrix} 3 & 4 & 5 \\ 6 & 1 & 8 \\ 8 & 6 & 3 \\ 7 & 8 & 7 \end{bmatrix}$ sale $\begin{bmatrix} 2 & 3 & 1 \end{bmatrix}$

Escriba un programa que lea una matriz, llame a la función creada y determine cual es la columna con la mayor cantidad de números pares

29. Escriba una función que reciba un entero, y entregue un vector conteniendo los dígitos del número equivalente en el sistema binario

Ejm. Entra **39** sale **[1 0 0 1 1 1]**

Escriba in programa que lea un entero, llame a la función creada y determine cuantos dígitos binarios son cero y cuantos son 1.

30. La operación **xor** en el sistema binario produce el siguiente resultado

m	**k**	**m xor k**
0	0	0
0	1	1
1	0	1
1	1	0

Esta operación se usa para encriptar mensajes en binario en los cuales **m** representa el mensaje, **k** la clave para encriptar el mensaje, y **e** el mensaje encriptado.

Ejemplo. Mensaje que se envía: m = 11011001
Clave: k = 01100011
Mensaje encriptado: e = 10111010

El receptor del mensaje encriptado, aplicando la misma clave puede conocer el mensaje:

Ejemplo. Mensaje encriptado: e = 10111010
Clave: k = 01100011
Mensaje recibido: m = 11011001

Escriba una función que reciba un dos vectores conteniendo números en el sistema binario y entregue otro vector conteniendo los números binarios con la operación xor. Esta función se usará para encriptar un mensaje y para conocer el menaje enviado. La función debe validar que los vectores contengan números binarios, caso contrario, el resultado es un vector nulo.

31. El siguiente es un algoritmo para generar un número aleatorio entero (seudo aleatorio)

1) Dado un número entero **x**
2) Sume los cuadrados de los dígitos del número. Este resultado es el número aleatorio

a) Escriba una función **c=aleatorio(x)** que entregue el resultado producido por el algoritmo anterior.

b) Escriba un programa que lea un valor inicial para **x** y llame a la función **aleatorio** repetidamente, enviando como nuevo dato, el resultado que entrega la función. Determine cuantas veces hay que llamar a la función **aleatorio** hasta que el resultado sea igual a algún valor que ya salió anteriormente. Esta cantidad se denomina longitud de la secuencia aleatoria

32. La función **strfind** entrega la posición inicial de todas las ocurrencias de algún texto en una cadena de caracteres, como se muestra en el ejemplo:

```
>> c='Cada proyecto tiene programas y compromisos';
>> t='pro';
>> v=strfind(c,t)
v =
   6   21   36
```

Escriba una función equivalente **subcadena(c,t)** que reciba iguales datos que la función **strfind** y produzca el mismo resultado, como en el ejemplo anterior.

33. Escriba una función recursiva para calcular números combinatorios:

$$C(m,n) = C(m-1, n-1) + C(m-1, n)$$

Sabiendo que: $C(m, n)=1$, si $n=0$ ó si $n=m$
$C(m, n)=m$, sin=1 ó si $n=m-1$

34. Escriba una función recursiva que permita invertir una cadena de caracteres.

9 Desarrollo de aplicaciones en MATLAB

El desarrollo de programas computacionales para resolver un problema extenso o complejo no debe intentarse mediante la instrumentación de un solo programa. Una técnica adecuada consiste en dividir el problema en segmentos o componentes y escribir programas o funciones para resolver cada uno de los componentes. Esto facilita escribirlos y probarlos individualmente e integrados para resolver el problema completo. Este método facilita la localización de errores y la realización de cambios y además permite organizar el trabajo en equipos de más de una persona.

En las siguientes secciones se desarrollará una aplicación elemental de proceso de datos mediante refinamientos socesivos aplicando este concepto de diseño e instrumentación modular. Se usarán funciones para describir cada uno de los componentes.

Ejemplo. Desarrollo en varias versiones de una aplicación para manejo de datos simples con un menú con las opciones:

1) Agregar
2) Consultar
3) Eliminar
4) Salir

VERSION 1: Acciones incluidas dentro de un solo programa

Instrumentación inicial: Las acciones son desarrolladas dentro del programa. Se usa función **ismember** de **MATLAB** para determinar cual celda contiene a un dato dado. Los datos son valores simples almacenados en un vector.

x: vector para almacenamiento de datos
ac: opción del usuario para control de acciones

```
%Programa
x=[ ];
ac=0;
while ac ~= 4
   clc;
   disp('1) Agregar');
   disp('2) Consultar');
   disp('3) Eliminar');
   disp('4) Salir');
   ac=input('Elija una acción ');
   switch ac
      case 1,   t=input('Ingrese dato ');
                [e,p]=ismember(t,x);
                if e==0
                   x=[x, t];
                else
                   disp('Ya existe el dato ');
                   pause;
                end
      case 2, t=input('Ingrese dato ');
                [e,p]=ismember(t,x);
                if e==0
                    disp('No está almacenado');
                    pause;
                else
                    disp('Si está almacenado');
                    pause;
                end
```

```
            case 3, t=input('Ingrese dato ');
                    [e,p]=ismember(t,x);
                    if e==0
                       disp('No está almacenado ');
                       pause;
                    else
                       x(p)=[ ];
                    end
        end
    end
```

VERSION 2:

Acciones almacenadas en funciones con trasmisión de parámetros
x: vector para almacenamiento de datos
ac: opción elegida por el usuario para control de acciones

Se usará la función **buscar** instrumentada antes, en lugar de la función **ismember**

```
%Programa
x=[ ];
ac=0;
while ac ~= 4
   menu1;
   ac=input('Elija una acción ');
   switch ac
      case 1, x=agregar(x);
      case 2, consultar(x);
      case 3, x=eliminar(x);
   end
end
```

```
function menu1
clc;
disp('1) Agregar');
disp('2) Consultar');
disp('3) Eliminar');
disp('4) Salir');
```

```
function x=agregar(x)
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
  x=[x, t];
else
  disp('Ya existe este dato ');
  pause;
end
```

```
function consultar(x)
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
   disp('No está almacenado');
   pause;
else
   disp('Si está almacenado');
   pause;
end
```

```
function x=eliminar(x)
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
   disp('El dato no está almacenado ');
   pause;
else
   x(p)=[ ];
end
```

```
function [r,p]=buscar(e,v)
n=length(v);
r=0;
p=0;
for i=1:n
   if e==v(i)
      r=1;
      p=i;
   end
end
```

VERSION 3:

Acciones descritas en funciones y uso de variables globales
x: vector para almacenamiento de datos
ac: opción elegida por el usuario para control de acciones

Se usará la función **buscar** instrumentada antes, en lugar de la función **ismember**

```
%Programa
global x
x=[ ];
ac=0;
while ac ~= 4
   menu1;
   ac=input('Elija una acción ');
   switch ac
      case 1, agregar;
      case 2, consultar;
      case 3, eliminar;
   end
end
```

```
function menu1
clc;
disp('1) Agregar');
disp('2) Consultar');
disp('3) Eliminar');
disp('4) Salir');
```

```
function agregar
global x;
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
  x=[x, t];
else
  disp('Ya existe este dato ');
  pause;
end
```

```
function consultar
global x;
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
   disp('No está almacenado');
   pause;
else
   disp('Si está almacenado');
   pause;
end
```

```
function eliminar
global x;
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
   disp('El dato no está almacenado ');
   pause;
else
   x(p)=[ ];
end
```

```
function [r,p]=buscar(e,v)
n=length(v);
r=0;
p=0;
for i=1:n
   if e==v(i)
      r=1;
      p=i;
   end
end
```

9.1 Una función para control de excepciones

La instrucción **try-catch-end** permite capturar errores en la ejecución para tomar alguna acción en caso de que se produzcan, evitando así que el programa finalice con una condición de error:

try

instrucciones en las que se quiere detectar una condición de error

catch

Acción que se realizará en caso de producirse el error

end

Ejemplo. Prevenir que una función no exista

```
x=input('Dato ');
try
  r=frst(x);
  disp(r);
catch
   disp('Error: función no existe');
end
```

Ejemplo. Prevenir el uso de una variable que no está definida

```
try
  y = x;
catch
  disp('Error: variable no definida');
  y = 0;
end
```

9.2 Almacenamiento y recuperación de archivos de datos en el disco

Almacenar variables (y su contenido) en un archivo

save archivo variable variable ...;

archivo: Nombre del achivo en el disco
variables: Variables cuyo contenido se almacenará

Ejemplo. Guardar las variables a y b en un archivo arch1 en el disco

```
>> a=[4 6 8 3 5];
>> b=[5 7 6; 9 8 4; 3 2 9];
>> save arch1 a  b;
```

Recuperar las variables (y su contenido) de un archivo

load archivo variable variable ...;

Ejemplo. Recuperar del archivo arch1 en el disco, las variables almacenadas

```
>> load arch1 a b;
```

Ejemplo. Prevenir que un archivo no esté almacenado en el disco

```
try
  load arch1 x;          %Se intenta asignar el archivo arch1 a la variable x
catch
  x=[ ];                 %Si el archivo no existe se asigna un valor nulo a la variable
end
```

Ejemplo. Almacene en el disco un vector con datos con algún nombre. Ej. **data**

```
>> x=[23, 45, 38, 27, 56, 72, 34, 27, 44];
>> save data x;
```

Ejemplo. Recupere el vector del disco y determine si algún valor se encuentra en el vector.

```
>> load data x;
>> v=34
>> [r, p]=ismember(v, x)
  r
     1
  p
     7
```

NOTA
El vector **x** quedará almacenado permanentemente en el disco en un archivo con el nombre 'data' con extensión **.mat** Si desea borrarlo puede hacerlo desde fuera de MATLAB o desde la ventana de comandos de MATLAB con el comando **delete**.

```
>> delete data.mat
```

Los comandos **save** y **load** pueden usarse dentro de programas

VERSION 4 del programa de manejo de datos

Acciones incluidas en funciones con una variable global
Vector almacenado en un archivo en el disco
Uso de la instrucción try-catch-end para verificar si existe el archivo

x: vector para almacenamiento de datos
ac: opción elegida por el usuario para control de acciones
datos: nombre del archivo en el disco para almacenar el vector

```
global x
try
   load datos x;                          %intenta cargar el vector x del archivo datos
catch
   x=[ ];                          %Si no está, inicia x como un vector nulo
end
ac=0;
while ac ~= 4
   menu1;
   ac=input('Elija una acción ');
   switch ac
      case 1, agregar;
      case 2, consultar;
      case 3, eliminar;
   end
end
save datos x;
```

```
function menu1
clc;
disp('1) Agregar');
disp('2) Consultar');
disp('3) Eliminar');
disp('4) Salir');
```

```
function agregar
global x;
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
  x=[x, t];
else
  disp('Ya existe este dato ');
  pause;
end
```

```
function consultar
global x;
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
   disp('No está almacenado');
   pause;
else
   disp('Si está almacenado');
   pause;
end
```

```
function eliminar
global x;
t=input('Ingrese dato ');
[e,p]=buscar(t,x);
if e==0
   disp('El dato no está almacenado ');
   pause;
else
   x(p)=[ ];
end
```

```
function [r,p]=buscar(e,v)
n=length(v);
r=0;
p=0;
for i=1:n
   if e==v(i)
      r=1;
      p=i;
   end
end
```

10 Manejo de registros en MATLAB

Un registro o estructura es un dispositivo para contener datos cuyos componentes pueden ser de diferente tipo, a diferencia de los vectores y matrices cuyos elementos deben ser todos del mismo tipo.

Notación para manejo de los componentes de un **registro**:

Nombre del registro . nombre del componente

Ejemplo. Almacenar en un registro los datos de un estudiante:

e: nombre del registro

Componentes que constituirán el registro

cod: código del estudiante
nom: nombre del estudiante
cursos: lista de cursos tomados

Ingresar datos de un estudiante

```
>> e.cod  = 125;
>> e.nom = 'Luis';
>> e.cursos    = [20, 30, 50, 40];
```

Mostrar el contenido del registro

```
>> e
   ans =
        cod: 125
        nom: 'Luis'
        cursos: [20 30 50 40]
```

Notación para manejo de los componentes de un **vector (tabla) de registros**:

Nombre del vector(número de elemento) . nombre del componente

Ejemplo. Almacenar en el primer elemento de un vector, un registro con los datos de un estudiante:

e: **nombre del vector con registros**
cod: **código del estudiante**
nom; **nombre del estudiante**
cursos: **lista de cursos tomados**

```
>> e(1).cod  = 125;
>> e(1).nom = 'Luis';
>> e(1).cursos    = [20, 30, 50, 40];
```

Almacenar en el segundo elemento de vector, un registro con los datos de un nuevo estudiante:

```
>> e(2).cod  = 148;
>> e(2).nom = 'Maria';
>> e(2).cursos     = [20, 70, 80, 40];
```

Contenido actual del vector **e** con los datos de los dos estudiantes

e	cod	nom	cursos
1	125	'Luis'	20 30 40 50
2	148	'María'	20 70 80 40

Mostrar el contenido del segundo estudiante

```
>> e(2)
ans =
        cod: 148
        nom: 'Maria'
        cursos: [20 70 80 40]
```

Mostrar los cursos tomados por el estudiante 2

```
>> e(2).cursos
ans =
        20    70    80    40
```

Mostrar el tercer curso tomado por el estudiante 2

```
>> e(2).cursos(3)
ans
    80
```

También se puede definir previamente la estructura del registro antes de asignar los datos. Se deben indicar entre comillas los nombres de los componentes y asignar valores (nulos). Esta definición es opcional.

```
>> e=struct('cod',0,'nom','','cursos',[ ]);
>> e.cod(1)=125;
>> e.nom(1)='Luis';
>> e.cursos(1)=[20, 30, 50, 40];
```

Este vector de registros, si se almacena en un archivo en disco pudiera conceptualmente considerarse un archivo

Ejemplo. Almacenar en disco en un archivo con el nombre **arch** el vector **e** de registros.

```
>> save arch e
```

Si hay muchos datos, conviene realizar el ingreso mediante un programa.

Ejemplo. Desarrollo de una aplicación con menú, funciones, variables globales, registros y archivo en disco para control de inventario de los artículos de un almacén.

Datos de cada artículo:

Código
Nombre
Cantidad actual del artículo

Menú

1) Ingresar: Ingreso de un nuevo artículo con la cantidad inicial
2) Agregar: Agregar cantidad a un artículo existente
3) Vender: Vender una cantidad de un artículo existente
4) Consultar: Conocer la cantidad disponible de un artículo
5) Eliminar: Eliminar o dar de baja un artículo
6) Salir

Variables

art: Nombre del vector de registros en memoria (variable global)
data: Nombre del archivo en disco

Nombres de los componentes de cada registro

cod: Código del artículo
nomb: Nombre del artículo
cant: Cantidad actual

NOTA. La función **ismember** no permite buscar elementos en un vector de registros, por lo cual se modificará nuestra conocida función **buscar** adaptándola a esta aplicación.

Programa para manejo de la aplicación

```
%Manejo de inventario
global art
try
   load data art;
catch
   art=[ ];
end
opc=0;
while opc ~= 6
    menu1;
   opc=input('Elija una opción ');
   switch opc
      case 1, ingresar;
      case 2, agregar;
      case 3, vender;
      case 4, consultar;
      case 5, eliminar
   end
end
save data art;          %guarda la variable art (vector) en el archivo datos
```

```
function menu1
clc;
disp('1) Ingresar artículo');
disp('2) Agregar');
disp('3) Vender');
disp('4) Consultar');
disp('5) Eliminar');
disp('6) Salir');
```

```
function ingresar
global art
c=input('Código del artículo ');
r=buscar1(c);
if r==0
    t.cod=c;
    t.nomb=input('Nombre del artículo ');
    t.cant=input('Cantidad inicial ');
    art=[art, t];
else
   disp('Artículo ya existe');
   pause
end
```

```
function agregar
global art;
c=input('Ingrese código de artículo ');
[r,p]=buscar1(c);
if r==1
   ct=input('Ingrese cantidad ');
   art(p).cant=art(p).cant + ct;
else
   disp('Artículo no existe');
   pause;
end
```

```
function vender
global art;
c=input('Ingrese código de artículo ');
[r,p]=buscar1(c);
if r==1
   ct=input('Ingrese cantidad ');
   if art(p).cant>=ct
      art(p).cant=art(p).cant - ct;
   else
      disp('Cantidad insuficiente');
      pause
   end
else
   disp('Artículo no existe');
   pause;
end
```

```
function consultar
global art;
c=input('Ingrese código de artículo ');
[r,p]=buscar1(c);
if r==1
   t=art(p);
   disp(t.nomb);
   disp(t.cant);
   pause;
else
   disp('Artículo no existe');
   pause;
end
```

```
function eliminar
global art;
c=input('Ingrese código de artículo ');
[r,p]=buscar1(c);
if r==1
   art(p)=[ ];
else
   disp('Artículo no existe');
   pause;
end
```

```
function [r,p]=buscar1(c)
global art;
n=length(art);
r=0;
p=0;
for i=1:n
   if c==art(i).cod
      r=1;
      p=i;
      return
   end
end
```

Ejemplo. Desarrollo de una aplicación con menú, funciones, registros y archivo en disco para control de registro de los socios de un club

Datos de cada socio: **código, género, edad**

Menú

1) Ingresar socio	(Ingresar datos del socio)
2) Borrar socio	(Eliminar datos del socio)
3) Consultar	(Buscar si el socio está registrado)
4) Salir	

Variables

s:	Vector conteniendo códigos de socios (Variable global)
archsoc:	Nombre externo del archivo en disco

Componentes de cada registro de socio

c	Código (número entero)
g	Género ('M' o 'F')
e	Edad (Número entero, edad en años)

NOTA. La función **ismember** no permite buscar elementos en un vector de registros, por lo cual se modificará nuestra conocida función **buscar**

```
%Programa para manejo de los datos de los socios de un club
global s
try
   load archsoc s;              % nombre externo del archivo
   s=[ ];
end
opc=0;
while opc~=4
   clc;
   menu1;
   opc=input('Elija una opción ');
   switch opc
      case 1,  ingr;
      case 2,  elim;
      case 3,  cons;
   end
end
save archsoc s;
```

```
function menu1
%Lista de opciones
disp('1) Ingresar socio');
disp('2) Eliminar socio');
disp('3) Consultar');
disp('4) Salir');
```

```
function ingr
%Ingreso de los datos de un registro al archivo
global s;
t.c=input('ingrese código ');
t.g=input('ingrese género ','s');
t.e=input('ingrese edad ');
x=t.c;
[r,p]=buscar2(x)
if r==1
   disp('código ya existe');
   pause;
else
   s=[s,t];
end
```

```
function cons
%Consulta de un registro, dado su código
global s;
x=input('Ingrese código ');
[r,p]=buscar2(x)
if r==1
   disp('socio si existe');
   disp(s(p).g);
   disp(s(p).e);
   pause;
else
   disp('socio no existe');
   pause;
end
```

```
function elim
%Elimina un registro del archivo s
%dado un código
global s;
x=input('ingrese código ');
[r,p]=buscar2(x)
if r==0
   disp('socio no existe');
   pause;
else
   s(p)=[ ];
end
```

```
function [r,p]=buscar2(x)
global s;
r=0;
p=0;
n=length(s);
for i=1:n
   if x==s(i).c
      r=1;
      p=i;
      return;
   end
end
```

10.1 Ejercicios de desarrollo de programas de aplicación

1. Escriba un programa para el pago a los n vendedores por comisión de una empresa. Para cada vendedor se deben leer registros con los siguientes datos: Código de identificación, nombre del vendedor, nivel (entero que puede ser 1, 2, 3), y el monto en dinero vendido en el mes. El pago depende del nivel: nivel 1: $400, nivel 2: $500, nivel 3: $600. A este valor hay que agregar el 5% del valor de las ventas realizadas. Adicionalmente hay un bono de $100 al o los vendedores con el mayor valor de ventas. Lea los datos y muestre:

a) El valor que hay que pagar a cada vendedor
b) El monto total que se requiere para pagar a todos los vendedores

2. Se tiene una lista de n códigos de artículos (números enteros) y la cantidad disponible de cada uno, y otra lista de m clientes (números enteros) junto con el código del artículo que desea (un solo artículo por cliente) y la cantidad requerida. Almacene ambas listas en vectores de registros y determine la cantidad total sobrante o faltante de cada artículo para atender las solicitudes de todos los clientes.

3. Escriba un programa para el control de la cantidad de **n** artículos de una empresa mediante **menu, switch** y funciones para realizar cada opción. Al inicio lea **n** y asigne cero a la cantidad de todos los artículos

1) Comprar
2) Vender
3) Mostrar
4) Salir

4. Escriba un programa para control del registro de los estudiantes para un evento.
El sistema debe incluir las siguientes opciones en un menú:

1) Registrar estudiante
2) Eliminar estudiante
3) Consultar registro de estudiantes
4) Mostrar estudiantes registrados
5) Salir

Use la instrucción **switch** y funciones para instrumentar cada opción y variables globales

5. Escriba un programa con un menú para registrar estudiantes en uno de los dos paralelos de una materia mediante las opciones indicadas a continuación. Cada paralelo debe ser representado mediante un vector y cada acción debe ser instrumentada mediante una función

1) Registrar
Lea el numero del paralelo elegido (1 o 2), luego lea el código del estudiante y agréguelo al vector correspondiente

2) Consultar
Lea el código del estudiante, búsquelo en los vectores y muestre el paralelo en el que está registrado

3) Cambiar
Lea el código del estudiante. Si está registrado elimínelo del vector y agréguelo al otro vector

4) Salir

6. Desarrolle una aplicación para registro de socios en un club.
Opciones

1) Ingresar socio (Ingresar el código del socio en un vector)
2) Consultar (Buscar si el código del socio está en el vector)
3) Borrar socio (Eliminar el código del socio del vector)
4) Salir

Use la instrucción **switch** y funciones para instrumentar cada opción y variables globales

7. Un grupo de **n** personas debe elegir a su representante. Será elegido si tiene al menos la mitad de los votos, caso contrario se debe repetir la votación. Cada persona es identificada con un número entero entre 1 y **n,** y cualquier persona pueden ser elegida. Luego de realizar la votación, se debe realizar el conteo de los votos y validar si es necesario realizar una nueva votación.

Escriba un programa para el proceso electoral con el siguiente menú:

1) Ingresar voto	Cada votante ingresa su numero de identificación y el número de la persona por la cual vota.
2) Consulta	Dado un número de identificación del votante, muestra si ya votó
3) Conteo	Muestra cuantas personas han votado y cuantas faltan de votar
4) Resultado	Muestra el resultado de la votación.

Sugerencia: Use un vector de n componentes para almacenar el conteo de votos de cada una de las n personas. Y otro vector para registrar quienes ya han votado.
Nota: Valide que cada persona vote una sola vez.

8. El Ministerio de Salud requiere implementar un programa para gestión de donantes de sangre que permita registrar y consultar resultados con el menú mostrado.

La información que se registra por paciente es: cédula, nombre, edad y tipo de sangre
La consulta por donante muestra el nombre del donante y su tipo de sangre, dado el número de cédula.
La consulta por tipo de sangre presenta el número de donantes por tipo de sangre

.

El tipo de sangre es un número (**1, 2, 3, 4, 5, 6, 7, 8**) los cuales corresponden a los siguientes tipos:
(**1**) O-, (**2**) O+, (**3**) A-, (**4**) A+, (**5**) B-, (**6**) B+, (**7**) AB-, (**8**) AB+

MENU
1. Ingreso de donante
2. Consulta de donante
3. Cosulta por tipo de sangre
4. Salir

9. Diseñe un sistema para registro y control de los pacientes que son atendidos en un hospital incluyendo las siguientes opciones:

Menú para Control de Atención Hospitalaria

1.- Ingreso de Paciente
- Código del paciente
- Nombre
- Código de la enfermedad
- Código del médico tratante

2.- Consulta de paciente
- Dado el código del paciente, mostrar el tipo de enfermedad y el médico tratante

3.- Consulta de médico
- Dado el código de un médico, mostrar la lista de pacientes asignados

4.- Consulta de enfermedad
- Dado el código de una enfermedad, mostrar los pacientes que tienen esta enfermedads

5.- Salir

10. Diseñe un sistema para registro y control de las películas de video de un local comercial incluyendo las siguientes opciones:

1.- Ingreso de película
- Código de la película
- Nombre de la película
- Código del actor principal
- Tipo de película (1: acción, 2: drama, 3: infantil, etc)
- Cantidad disponible

2.- Consulta de película

Dado el código de una película ,mostrar el nombre y la cantidad disponible

3.- Consulta de actor principal

Dado el código de un actor, mostrar la lista de películas con ese actor

4.- Consulta de tipo de película

Dado el tipo de película, mostrar los nombres de las películas disponibles

5.- Salir

11 Estructuras de datos

Las estructuras de datos son dispositivos especiales usados como contenedores de datos. Son diseñados para facilitar las operaciones en ciertas aplicaciones.

En esta sección se describirá el concepto de algunas estructuras básicas mediante programas con un menú con las opciones para acceder a las operaciones básicas permitidas en cada estructura.

Una instrumentación más conveniente y general usa una función para cada una de las operaciónes con las que se puede acceder a la estructura de datos. Estas funciones son llamadas desde un programa para desarrollar alguna aplicación.

Cada estructura es instrumentada mediante un vector o arreglo simple, aprovechando el manejo dinámico que permite MATLAB. Instrumentaciones más complejas y formales se pueden realizar usando otros dispositivos para conectar los componentes de la estructura, por ejemplo los punteros dinámicos.

Al final de la sección se describen dos aplicaciones de interés que usan estas estructuras de datos

11.1 Pila

Es una estructura lineal que permite agregar y sacar elementos solamente de uno de los dos extremos.

```
%Manejo de una pila
p=[ ];
x=0;
while x~=4
   clc
   disp('1) Agregar elemento')
   disp('2) Retirar elemento');
   disp('3) Mostrar pila');
   disp('4) Salir');
   x=input('Elija opción ');
   switch x
      case 1
         e=input('Ingrese elemento ');
         p=[p,e];
      case 2
         n=length(p);
         if n==0
            disp('Pila vacía');
            pause;
         else
            disp(p(n));
            pause;
            p(n)=[ ];
         end
      case 3
         disp(p);
         pause;
   end
end
```

11.2 Cola

Esta estructura tambien es lineal y permite agregar y sacar elementos de ambos extremos.

```
%Manejo de una cola
c=[ ];
x=0;
while x~=4
   clc
   disp('1) Agregar elemento')
   disp('2) Retirar elemento');
   disp('3) Mostrar cola');
   disp('4) Salir');
   x=input('Elija opción ');
   switch x
      case 1
         e=input('Ingrese elemento ');
         c=[c,e];
      case 2
         n=length(c);
         if n==0
            disp('Cola vacía');
            pause;
         else
            disp(c(1));
            pause;
            c(1)=[ ];
         end
      case 3
         disp(c);
         pause;
   end
end
```

11.3 Lista

Esta estructura lineal es más general que la pila y la cola pues permite permite agregar y sacar elementos de cualquier lugar de la lista.

```
%Manejo de una lista
l=[ ];
x=0;
while x~=4
   clc
   disp('1) Agregar elemento')
   disp('2) Retirar elemento');
   disp('3) Mostrar lista');
   disp('4) Salir');
   x=input('Elija opción ');
   switch x
      case 1
         n=length(l);
         e=input('Ingrese elemento ');
         if n==0
            l=e;
         else
            p=input('Ingrese posición ');
            if p<1 | p>n
               disp('Posición no válida');
               pause
            else
               l=[l(1:p-1),e,l(p:n)];
            end
         end
      case 2
         p=input('Ingrese posición ');
         n=length(l);
         if p<1 | p>n
            disp('Posición no válida');
            pause
         else
            disp(l(p));
            pause;
            l(p)=[ ];
         end
      case 3
         disp(l);
         pause;
   end
end
```

11.4 Aplicaciones

11.4.1 Uso de una pila para buscar la salida en un laberinto

Suponga el problema de encontar la ruta de salida de un laberinto de nxm casillas. La casilla inicial es (2,2), y la casilla de llegada es (n-1, m-1). Si la casilla es libre contiene 0 y si está bloqueada es 1. De cualquier casilla se puede seguir a alguna de las 8 casillas adyacente, siempre que esté libre y no haya sido visitada anteriormente. Todos los bordes están bloqueados. Se necesita una pila para almacenar la ruta y poder retroceder y continuar con otra casilla libre y no visitada.

```
q=input('Ingrese laberinto ');
[n,m]=size(q);
dx(1)=0;  dy(1)=1;                    % Rutas posibles (8 direcciones)
dx(2)=1;  dy(2)=1;                    % en sentido del reloj
dx(3)=1;  dy(3)=0;                    % X positivo hacia abajo
dx(4)=1;  dy(4)=-1;                   % Y positivo hacia la derecha
dx(5)=0;  dy(5)=-1;
dx(6)=-1; dy(6)=-1;
dx(7)=-1; dy(7)=0;
dx(8)=-1; dy(8)=1;
v=zeros(n,m);                         %Matriz de casillas visitadas
pila=[ ];                             %Pila para guardar la ruta recorrida
x=2;y=2;                              %Nodo inicial
pila(1).x=x;                          %Se coloca en la pila
pila(1).y=y;
display([pila(1).x,pila(1).y]);
salida=0;
while length(pila)>0 & salida==0
   k=length(pila);
   x=pila(k).x;                       % Nodo en el tope de la pila
   y=pila(k).y;
   if x==n-1 & y==m-1                 % Si llega al nodo final, salir
      salida=1;
      break;
   end
   pila(k)=[ ];                       % Sacar el nodo de la pila
   poner=1;
   while poner==1
      poner=0;
      for i=1:8
         px=x+dx(i);
         py=y+dy(i);
         if px>1 & px<n & py>1 & py<m & q(px,py)==0 & v(px,py)==0
            t.x=px;
            t.y=py;
            pila=[pila,t];    %Colocar nodo en la pila, si hay ruta y no fue visitado
            poner=1;
            x=px;
            y=py;
            v(px,py)=1;      %Marcar nodo visitado
            break;
         end
      end
   end
end
```

```
if salida==1                        %Ruta de salida del laberinto
   for i=1:length(pila)
      disp([pila(i).x,pila(i).y]);
   end
else
   disp('No hay ruta de salida');
end
```

Ejemplo:

```
>> q =[ 1   1   1   1   1   1   1   1;
        1   0   0   0   1   0   0   1;
        1   1   1   0   1   1   1   1;
        1   0   0   1   1   1   1   1;
        1   0   1   0   1   0   1   1;
        1   0   1   1   0   1   0   1;
        1   0   1   0   0   0   0   1;
        1   1   1   1   1   1   1   1 ];
```

Interacción

```
Ingrese laberinto q
 ans =
     2     2
     2     3
     2     4
     3     4
     4     3
     5     4
     6     5
     7     6
     7     7
```

11.4.2 Uso de una cola para simular la atención en una estación de servicio

Suponer una cola de clientes que son atendidos en una estación de servicio. Se conoce que la llegada de un cliente en cada minuto tiene distribución uniforme con un valor de probabilidad dado. El tiempo de atención del cliente que está frente a la cola también es un valor aleatorio real entre 0 y un valor máximo dado como dato.

Se desea conocer cuantos clientes quedarían en la cola luego de trancurrir una cantidad de minutos especificada.

```
%Simulación de una cola de clientes
m=input('Cantidad de minutos para la simulación ');
pri=input('Probabilidad de ingreso de un cliente en cada minuto ');
atenc=input('Tiempo máximo para atención de cada cliente en minutos ');
t=0;                    %Tiempo en minutos para la simulación (reloj)
c=[ ];
while t<m
   p=rand;              %Probabilidad para la llegada de un cliente
   if p<=pri
      d=atenc*rand;     %Duración de la atención del cliente
      c=[c, d];         %Agregar cliente (se guarda su duración en la cola)
   end
   n=length(c);
   if n>0
      c(1)=c(1)-1;
      if c(1)<=0
         c(1)=[ ];      %Eliminar el cliente del frente de la cola (terminó)
      end
   end
   t=t+1;
end
disp('Cantidad de clientes que quedaron');
n=length(c);
disp(n);
```

Interacción

Cantidad de minutos para la simulación 120

Probabilidad de ingreso de un cliente en cada minuto 0.25

Tiempo máximo para atención de cada cliente en minutos 5

Cantidad de clientes que quedaron

3

12. Eficiencia de algoritmos

La eficiencia de un algoritmo está relacionada con el tiempo necesario para obtener la solución. Este tiempo depende de la cantidad de operaciones que se deben realizar. Así, si se tienen dos algoritmos para resolver un mismo problema, es más eficiente el que requiere menos operaciones para producir el mismo resultado.

Sea **n** el tamaño del problema, y **T(n)** la eficiencia del algoritmo (cantidad de operaciones requeridas). Para obtener **T(n)** se pueden realizar pruebas en el computador con diferentes valores de **n** registrando el tiempo de ejecución. Siendo este tiempo proporcional a la cantidad de operaciones que se realizaron, se puede estimar la función **T(n)**.

Esta forma experimental para determinar **T(n)** tiene el inconveniente de requerir la instrumentación computacional del algoritmo para realizar las pruebas. Es preferible conocer la eficiencia del algoritmo antes de invertir esfuerzo en la programación computacional.

Para determinar la eficiencia de un algoritmo antes de su instrumentación,se puede analizar la estructura del algoritmo.

Ejemplo. El siguiente algoritmo calcula la suma de los cubos de los primeros n números naturales. Encontrar T(n)

Sea **T** la cantidad de sumas que se realizan

```
. . .
s ← 0
Para i=1, 2, ..., n
  s ← s + i^3
fin
. . .
```

La suma está dentro de una repetición que se realiza **n** veces, por lo tanto,

T(n) = n

Ejemplo. El siguiente algoritmo suma los elementos de una matriz cuadrada **a** de orden **n**.

Encontrar **T(n)**

Sea **T** la cantidad de sumas

```
. . .
s←0
Para i=1, 2, ..., n
  Para j=1, 2, ..., n
    s ← s + a_{i, j}
  fin
fin
. . .
```

La suma está incluida en una repetición doble. La variable **i,** cambia **n** veces y para cada uno de sus valores, la variable **j** cambia **n** veces. Por lo tanto.

$T(n) = n^2$

Ejemplo. El siguiente algoritmo es una modificación del anterior. Suponga que se desea sumar únicamente los elementos de la sub-matriz triangular superior. Obtener **T(n)**

```
. . .
s←0
Para i=1, 2, ..., n
  Para j=i, i+1, ..., n
    s ← s + a_{i, j}
  fin
fin
```

Si no es evidente la forma de **T(n)**, se puede recorrer el algoritmo y anotar la cantidad de sumas que se realizan

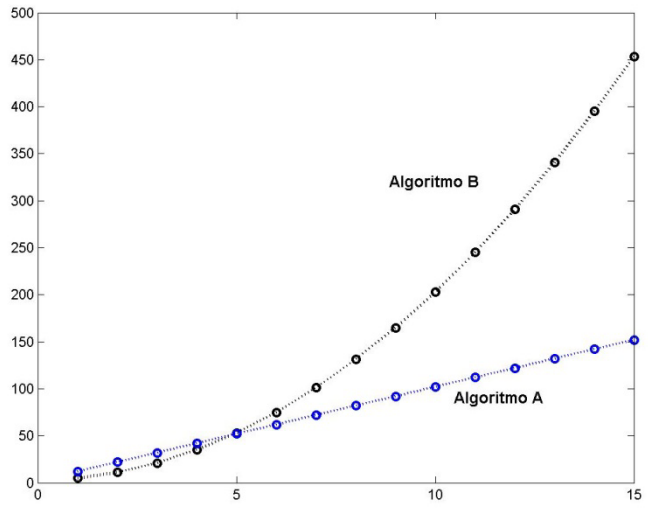
i	**j**
1	n
2	n-1
3	n-2
...	...
n-1	2
n	1

Entonces, **T(n) = 1 + 2 + ... + n =** $\frac{n}{2}(n+1) = \frac{n^2}{2} + \frac{n}{2}$ (suma de una serie aritmética)

12.1 La notación O( )

Supongamos que para resolver un problema se han diseñado dos algoritmos: **A** y **B**, con eficiencias $T_A(n) = 10n+2$, $T_B(n) = 2n^2 + 3$, respectivamente. ¿Cual algoritmo es más eficiente?.

Para valores pequeños de **n**, $T_B(n) < T_A(n)$, pero para valores grandes de **n**, $T_A(n) < T_B(n)$. Es de interés práctico determinar la eficiencia de los algoritmos para valores grandes de **n**, por lo tanto el algoritmo **A** es más eficiente que el algoritmo **B** como se puede observar en el gráfico:



Si **T(n)** incluye términos con **n** que tienen diferente orden, es suficiente considerar el término de mayor orden pues es el que determina la eficiencia del algoritmo cuando **n** es grande. No es necesario incluir los coeficientes y las constantes.

Ejemplo. Suponga $T(n) = n^2 + n + 10$. Evaluar **T** para algunos valores de **n**

T(2) = 4 + 2 + 10
T(5) = 25 + 5 + 10
T(20) = 400 + 20 + 10
T(100) = 10000 + 100 + 10

Se observa que a medida que **n** crece, **T** depende principalmente de n^2. Este hecho se puede expresar usando la notación **O( )** la cual indica el "orden" de la eficiencia del algoritmo, y se puede escribir: $T(n) = O(n^2)$ lo cual significa que la eficiencia es proporcional a n^2, y se dice que el algoritmo tiene eficiencia cuadrática o de segundo orden.

En general, dado un problema de tamaño **n**, la medida de la eficiencia **T(n)** de un algoritmo se puede expresar con la notación **O(g(n))** siendo **g(n)** alguna expresión tal como: **n, n^2, n^3, ..., log(n), n log(n), ..., 2^n, n!, ...** la cual expresa el orden de la cantidad de operaciones que requiere el algoritmo.

Es de interés medir la eficiencia de los algoritmos para problemas de **tamaño grande**, pues para estos valores de **n** se debe conocer la eficiencia. En el siguiente cuadro se ha tabulado **T(n)** con algunos valores de **n** y para algunas funciones típicas **g(n)**.

Tabulación de T(n) con algunos valores de n para algunas funciones típicas g(n)

n	[log(n)]	n	[n log(n)]	n^2	n^3	2^n	n!
1	0	1	0	1	1	2	1
3	1	3	3	9	27	8	6
5	1	5	8	25	125	32	120
7	1	7	13	49	343	128	5040
9	2	9	19	81	729	512	3.62×10^5
11	2	11	26	121	1331	2048	3.99×10^7
13	2	13	33	169	2197	8192	6.22×10^9
15	2	15	40	225	3375	32768	1.30×10^{12}
17	2	17	48	289	4913	1.31×10^5	3.55×10^{14}
19	2	19	55	361	6859	5.24×10^5	1.21×10^{17}
21	3	21	63	441	9261	2.09×10^6	5.10×10^{19}
23	3	23	72	529	12167	8.38×10^6	2.58×10^{22}
25	3	25	80	625	15625	3.35×10^7	1.55×10^{25}
50	3	50	195	2500	125000	1.12×10^{15}	3.04×10^{64}
100	4	100	460	10000	1000000	1.26×10^{30}	9.33×10^{157}

Los algoritmos en las dos últimas columnas son de tipo **exponencial** y **factorial** respectivamente. Se puede observar que aún con valores relativamente pequeños de **n** el valor de **T(n)** es extremadamente alto. Los algoritmos con este tipo de eficiencia se denominan **no-factibles** pues ningún computador actual pudiera calcular la solución en un tiempo aceptable para valores de **n** grandes.

.

12.2 Funciones de MATLAB para medir experimentalmente eficencia de programas

Para medir el tiempo real de ejecución de un programa se usan las funciones **TIC** y **TOC**

TIC inicia el cronómetro

TOC detiene el cronómetro y muestra el tiempo transcurrido

Ejemplo. Suponga que se tiene un programa cuyo nombre es **prueba** y se desea medir el tiempo real de ejecución resolviendo un problema de tamaño **n**.

Para que el tiempo se refiera solamente al tiempo de computación, los datos deben estar dentro del programa para que no intervenga el tiempo que se usa en el ingresos de los datos.

Digitar en la ventana de comandos de MATLAB

>> tic; prueba; toc

Se mostrará el tiempo de proceso del programa **prueba**

Se pueden realizar varios intentos para definir la curva de eficiencia del programa.

13 Algunos commandos de MATLAB para mejorar la interacción visual

Con el propósito de facilitar la interacción MATLAB dispone de un lenguaje para diseñar la interfaz gráfica para los usuarios. Su especificación require una planificación previa y el uso de formularios para detallar las propiedades de los objetos que se definen. Los siguientes dos commandos son versiones simplificadas de los objetos que se pueden definir al diseñar esta interfaz.

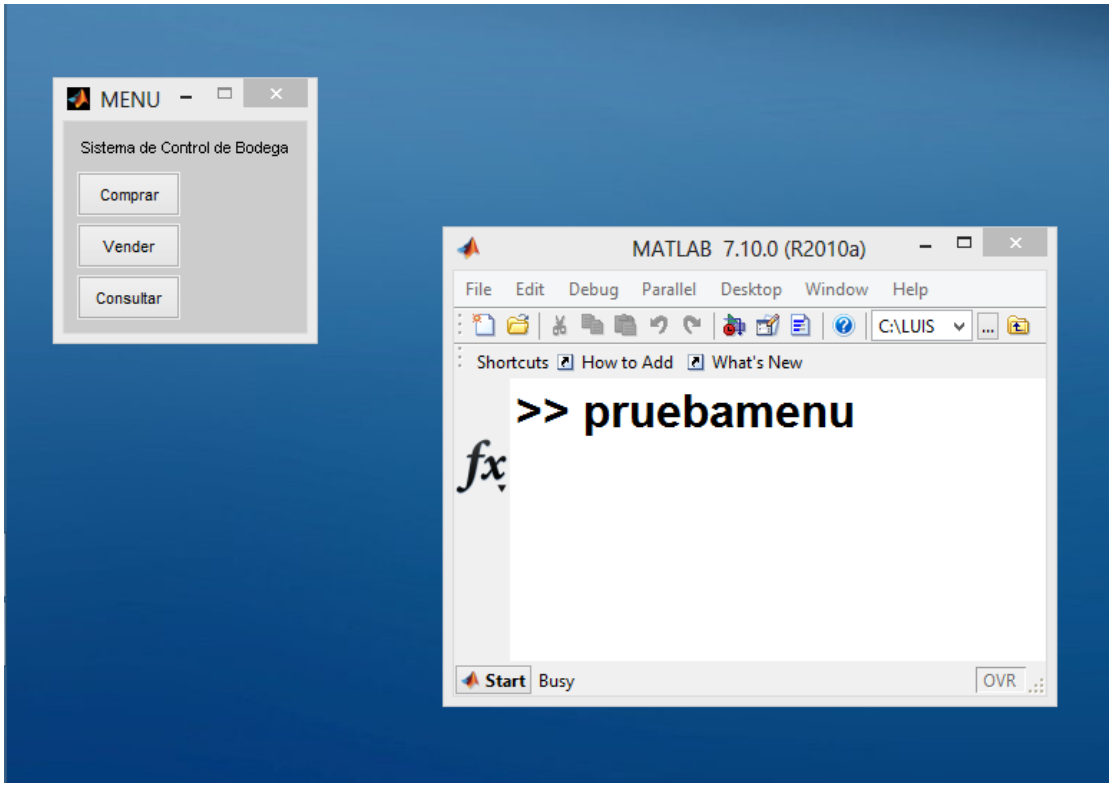
13.1 Interacción mediante el comando MENU

El comando menú se puede usar para mostrar con más claridad las opciones para dirigir las acciones de un programa. Se deben especificar el rótulo para la caja y las opciones, mediante un arreglo de celdas de cadenas de caracteres.

Ejemplo. Defina una caja para mostrar un menú con tres opciones: Comprar, Vender, y Consultar para control de una bodega:

```
opciones={'Comprar','Vender','Consultar'};
v=menu('Sistema de Control de Bodega',opciones);
 switch v
    case 1

    case 2

    case 3

 end
```

Al ejecutar este segmento se muestra en la pantalla una caja con el menú. Mediante el mouse se presiona en alguna de las opciones, y la acción resultante es el valor 1, 2, o 3 que se asignará a la variable v con la cual se dirije la acción que será realizada.

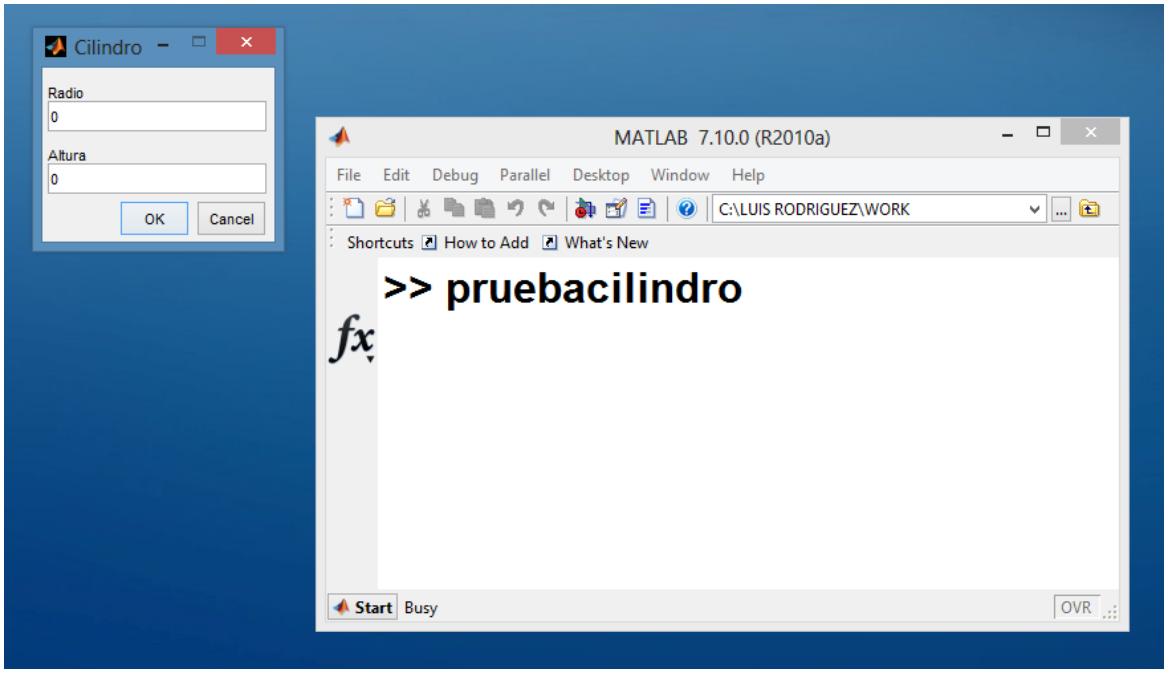


13.2 Ingreso de datos con el comando INPUTDLG

Este commando muestra una caja con un diseño amigable para facilitar el ingreso de los datos. Se usan arreglos de celdas de cadenas de caracteres para definir rótulos para cada linea de datos y los valores que se usarán por omision. Además el rótulo para la caja y la cantidad de líneas para cada dato, como se muestra en el ejemplo siguiente

Ejemplo. Use una caja de diálogo para ingresar datos de radio y altura de un cilindro. Use valores nulos como valores por omisión. Convierta los datos ingresados a su representación real y calcule el volumen del cilindro:

```
datos={'Radio','Altura'};
valores={'0','0'};
s=inputdlg(datos,'Cilindro',1,valores);
r=str2num(s{1});
a=str2num(s{2});
vol=pi*r^2*a;
disp(vol)
```



14 Interacción de MATLAB con otros entornos

Interacción con EXCEL

1) Importar una tabla de datos desde Excel a una matriz en MATLAB

a) En **Excel** cree la tabla y almacénela con formato tipo texto delimitado con tabulaciones. Elija algún nombre. Ejemplo **t.txt**

b) En **MATLAB** cargue la tabla **t** y úsela como una matriz:

```
>> load  t.txt;
>> a=t
```

2) Exportar una matriz de datos desde MATLAB a una tabla en Excel

a) En **MATLAB** cree una matriz y almacénela con el comando **save** con el siguiente formato. Elija los nombres. Ejemplo
a: nombre de la matriz en MATLAB
t: nombre para la tabla almacenada

```
>> save  t  a  -ascii
```

b) En **Excel** abra el archivo **t** y úselo como una tabla de datos

15 Bibliografía

The MathWorks, Inc. *Using MATLAB Computation, Visualization, Programming*, version 7

ANEXO

MATLAB INTERACTIVO

En esta sección se describe el componente interactivo de MATLAB para los usuarios que desean resolver problemas utilizando las funciones disponibles en este programa y que son suficientes para resolver una gran cantidad de problemas de aplicación.

El componente programable cubierto en la primera parte de este documento es obligatorio para los estudiantes que desean desarrollar su capacidad lógica para enfrentar problemas más complejos para cuya solución no es suficiente el componente interactivo de MATLAB.

CONTENIDO

1 Introducción

1.1 Objetivo

Proporcionar a los interesados los conocimientos básicos para usar el entorno de MATLAB mediante instrucciones en forma interactiva.

1.2 Metodología

Mediante los ejemplos y la descripción de su uso, el interesado puede adquirir en forma autónoma y en pocas horas, los conocimientos básicos para utilizar MATLAB.

Para facilitar el aprendizaje se sugiere abrir dos ventanas en la pantalla del computador, una con el programa MATLAB y otra con este documento, entonces puede escribir y probar cada ejemplo en la ventana de comandos de MATLAB.

1.3 El programa MATLAB

MATLAB (Matrix Laboratory) es un programa interactivo y programable de uso general. Es un instrumento computacional simple, versátil y de gran poder para aplicaciones numéricas, simbólicas y gráficas y contiene una gran cantidad de funciones predefinidas para diversas aplicaciones en ciencias e ingeniería.

La interacción se realiza mediante instrucciones denominadas comandos que son realizadas por un interpretador muy eficiente. También se pueden usar funciones y programas escritas por los usuarios en un lenguaje estructurado. Los objetos básicos con los cuales opera MATLAB son matrices. La asignación de memoria a cada variable se realiza en forma dinámica.

1.4 Características de MATLAB

- Cálculo numérico rápido y con alta precisión
- Capacidad para manejo matemático simbólico
- Funciones para graficación y visualización avanzada
- Programación mediante un lenguaje de alto nivel
- Soporte para programación estructurada y orientada a objetos
- Facilidades para diseño de interfaz gráfica
- Extensa biblioteca de funciones
- Paquetes especializados para diversas áreas de ciencias e ingeniería

Operación

- Simple y eficiente
- Interactivo y programable
- Sistema de ayuda en línea
- Interacción con otros entornos

1.5 Uso interactivo de MATLAB

El entorno de MATLAB está organizado mediante ventanas. Las principales son

Command Window	Es la ventana de comandos para ineractuar con MATLAB
Command History	Contiene el registro de los comandos que han sido ingresados.
Workspace	Contiene la descripción de las variables usadas en cada sesión.

Al ingresar al programa MATLAB se abren varias ventanas. Se sugiere al inicio dejar activa únicamente la ventana de comandos, cerrando las otras ventanas. Si desea restaurarlas use la opción **view** de la barra de herramientas de MATLAB.

El símbolo **>>** indica que el programa está en la ventana de comandos y que está listo para recibir sus instrucciones (comandos) como se ilustra a continuación:

Ejemplo. Suponga que desea evaluar la siguient expresión

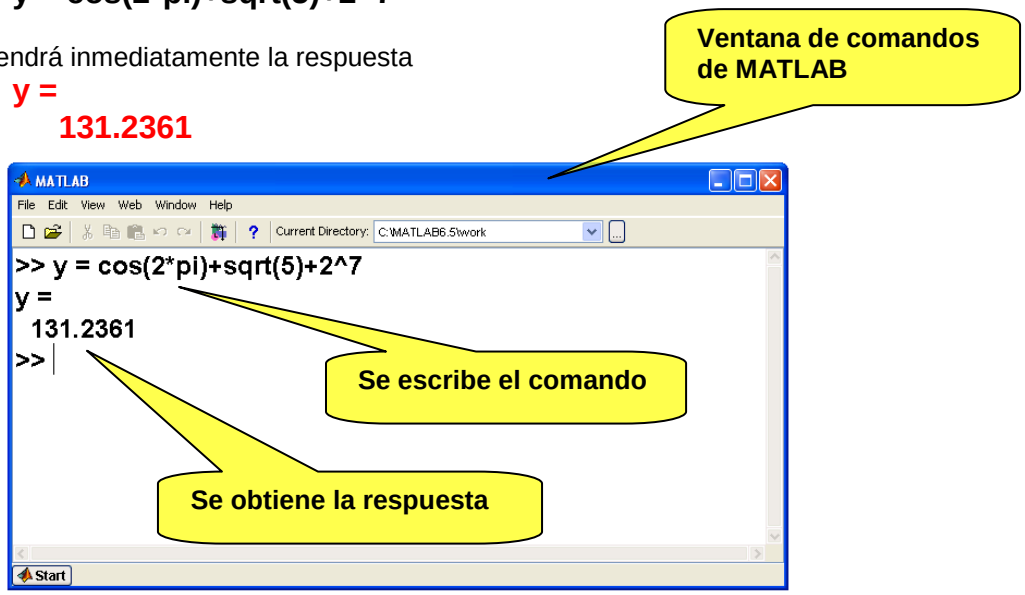
$$y = \cos(2\pi) + \sqrt{5} + 2^7$$

Entonces deberá digitar en la ventana de comandos de MATLAB

```
y = cos(2*pi)+sqrt(5)+2^7
```

Se obtendrá inmediatamente la respuesta

```
y =
    131.2361
```



1.6 Uso de comandos de MATLAB

En esta sección se revisa el uso de los comandos básicos de MATLAB. Debe escribir cada ejemplo y presionar la tecla de ingreso para obtener la respuesta.

MATLAB mostrará el resultado inmediatamente, o un mensaje si hubo algún error. Recuerde que la mejor manera de aprender un lenguaje es la práctica.

Solamente en los primeros ejemplos se ha escrito el resultado que produce MATLAB para cada comando, evitando así que este tutorial sea innecesariamente extenso. Los resultados los puede observar el usuario al probar cada ejemplo. Al final de cada ejemplo se ha escrito con **letra azul** una breve explicación para faciltar la comprensión de cada comando y el resultado.

Digite cada uno de los siguientes ejemplos en la ventana de comandos.

```
>>  exp(2)/3                    calcule e^2 / 3 y muestre inmediatamente el resultado
    ans =
          2.4630                respuesta mostrada por MATLAB, ans significa answer

>>  x = exp(2)/3                calcule e^2 / 3 y asigne el resultado a la variable x
    x =
          2.4630                respuesta mostrada por MATLAB

>>  y = exp(2)/3;               el ; evita que el resultado se muestre inmediatamente
>>  y                           para conocer el contenido se escribe el nombre de la variable
    y =
          2.4630                respuesta mostrada por MATLAB
>>  u = 2*x+1                   puede usar el contenido de las variables
    u =
          5.9260                respuesta mostrada por MATLAB, es el contenido de u
>> x = exp(2)/3; y=2*x+1        Puede escribir y ejecutar varios comandos juntos
    x =
          2.4630                respuestas mostradas por MATLAB
    y =
          5.9260
```

1.7 Reutilización de comandos

Para reutilizar comandos presione las teclas del cursor ↑ ↓ realice los cambios y presione la tecla de ingreso. Adicionalmente puede copiar porciones del texto para escribir nuevos comandos.

1.8 El sistema de ayuda de MATLAB

MATLAB ofrece una descripción detallada del uso de cada comando y cada función digitando **help** y el nombre del comando.

Ejemplo. Para conocer el uso de la función **sqrt**, digite

```
>> help sqrt
```

```
SQRT   Square root.
SQRT(X) is the square root of the elements of X. Complex
results are produced if X is not positive.
```

Esta información aparece en pantalla

```
>> help        despliega temas de ayuda
>> help ops    despliega comandos de un tema.  Ej.  lista de operadores
>> help exp    uso de un comando específico.   Ej. función exponencial
```

Adicionalmente, presionando el ícono **Help** en la barra superior de la ventana de MATLAB puede entrar al sistema de ayuda de MATLAB organizado con un índice con opciones de búsqueda por contenido con ejemplos y demostraciones.

1.9 Algunos ejemplos para practicar en MATLAB

Como una introducción al uso de este programa, digite cada uno de los siguientes ejemplos en la ventana de comandos. Observe los resultados obtenidos.

1) Para resolver el sistema:

$$2x + 3y = 4$$
$$5x - 2y = 6$$

Digite en la ventana de comandos

```
>> a = [2, 3; 5, -2];      Ingresar la matriz de coeficientes
>> b = [4; 6];             Ingresar el vector columna de constantes
>> x = inv(a)*b;           Obtener la solución con la matriz inversa
   x = 1.3684              Vector solución
       0.4211
```

2) Resolver la ecuación cúbica $5x^3 + 2x^2 - 3x + 1 = 0$;

```
>> a = [5, 2, -3, 1];      Ingresar los coeficientes de la ecuación
>> x = roots(a)            Obtener las tres raíces
   x =  -1.1060            Una raíz real y dos raíces complejas
        0.3530 + 0.2371i
        0.3530 - 0.2371i
```

3) Graficar la función $f(x)=sen(x)\,e^x$ en el intervalo $0 \le x \le \pi$

```
>> f = 'sin(x)*exp(x)';     Escribir la función entre comillas simples
>> ezplot(f, [0, pi]);      Función para graficar
>> grid on;                 Mostrar cuadrículas en el gráfico
```

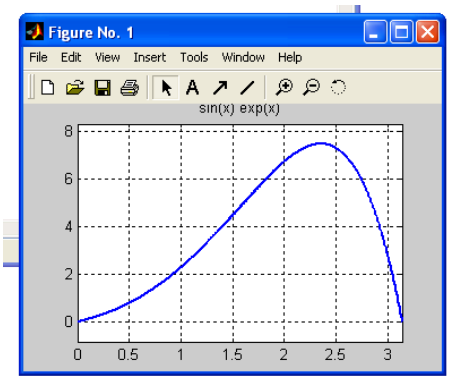


Gráfico producido por MATLAB

2 Cálculo numérico

2.1 Símbolos especiales en MATLAB

[ ] para definir vectores y matrices
( ) para definir precedencia en expresiones y para subíndices
, para separar elementos de un vector use comas o espacios
; para separar filas y para evitar mostrar contenido de variables
% para iniciar un comentario (programas y funciones)
... para continuar un comando en la siguiente línea

2.2 Formatos de exhibición de números en la pantalla

```
>> format long                  formato largo: quince decimales
>> x=exp(2)                     ejemplo para ilustración
    x =
       7.389056098930650
>> format bank                  formato para manejo monetario: 2 decimales
>> x
   x =
       7.39
>> format rat                   notación racional (fracciones)
>> x
    x =
       2431/329
>> format short e               notación científica
>> x
    x =
       7.3891e+000

>> format long e                notación científica con quince decimales
>> format +                     muestra signos +,  , -
>> format compact               suprime líneas adicionales en la salida
>> format loose                 inserta líneas en blanco en la salida(recomendado)
>> format hex                   formato hexadecimal
>> vpa(sqrt(2), 20)             format para precisión variable( variable precision arithmetic)
    ans =                       (muestra la raíz cuadrada de 2 con 20 dígitos
       1.4142135623730950488

>> format short                 formato con el que se inicia de MATLAB
>> x=exp(2)
    x =                         en el formato estándar se muestran cuatro decimales
       7.3891                   con el cuarto decimal redondeado
```

2.3 Operadores aritméticos

```
+ - * / \ ^                     ^  se usa para potenciación
                                /  es división a la derecha
                                \  es división a la izquierda (para matrices)
>> help ops                     listar los operadores y caracteres especiales
```

2.4 Funciones matemáticas

```
>> help elfun                   listar las funciones matemáticas elementales
      Trigonometric.
      sin       - Sine.
      sinh      - Hyperbolic sine.
      asin      - Inverse sine.
      asinh     - Inverse hyperbolic sine.
      etc.
```

2.5 Operadores relacionales y lógicos

```
<  <=  >  >=  ==  ~=  &  |  ~     los tres últimos corresponden a: ∧ ∨ ¬
                                  ==  representa al símbolo  =
                                  ~=  representa al símbolo  ≠

>> t=sin(2) < 0.8 & log(2) > 0.5  el resultado es un valor lógico  (0  o  1)
```

2.6 Símbolos numéricos especiales

```
>> 2/0
        Inf                 es el símbolo  ∞
>> 0/0
        NaN                 significa “Not A Number” (valor indeterminado)
>> pi                       contiene la constante  π
>> eps                      es la precisión del tipo real en MATLAB
```

2.7 Números complejos

i representa al símbolo $\sqrt{-1}$

```
>> x = 3+2i                 asignar un número complejo
>> t = 2*x + 3 - 5i         operación con números complejos
   t =
      9.0000 - 1.0000i
>> y =exp(x)                el resultado también es complejo
   y =
     -8.3585 +18.2637i
>> y = log(-2)              el referencial de MATLAB son los complejos
   y =
      0.6931 + 3.1416i
```

2.8 Funciones para números complejos

conj, real, imag, abs, angle, complex

```
>> z=3+2i;
>> t=conj(z)                obtener el conjugado
```

2.9 Algunos comandos del sistema operativo

```
>> clc          despeja la ventana de comandos
>> who          lista las variables en uso
>> whos         lista las variables en uso y su descripción
>> clear        borra variables del area de trabajo
>> pwd          muestra cual es el directorio actual
>> cd           cambia la ruta del directorio actual
>> dir          lista el contenido del directorio actual
>> save         almacena las variables en un archivo en el disco
>> load         carga variables y su contenido desde el disco
>> delete       elimina un archivo en el disco
>> quit         para terminar la sesión con MATLAB (no lo digite aun)
>> date         fecha
>> clock        fecha hora, vea su uso con help.
```

2.10 Variables

- No requieren ser declaradas
- Su tipo depende del valor asignado
- Pueden ser redefinidas
- Sensible al tipo de letra (mayúsculas o minúsculas)
- **ans** es la variable por omisión provista por MATLAB
- MATLAB realiza la asignación de memoria a las variables durante la ejecución.

```
>> x=3                      x es de tipo real
>> x=x+1                    se puede modificar el contenido de la variable
>> x='mensaje'              ahora x es de tipo literal  (use comillas simples)
>> x=[2 7 4]                x es ahora un vector
>> x=2*t + 1                es un error si  t  no ha sido asignada
```

2.11 Ejercicios. Digite y obtenga el resultado para cada ejercicio

1) $x = \sqrt{5} + 3^5$

2) $y = \dfrac{\sqrt{2}}{e^2 - 1}$

3) $u = \dfrac{\cos(\pi + 0.2) - 3}{\tan(e^2 - 2) + 0.2^7}$

4) $t = \dfrac{\sqrt{2} + 1}{\dfrac{\text{sen}(2) - 0.1}{e^2 - 1}}$

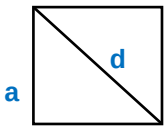
5) $a = \dfrac{\sqrt{2}}{e^2 - 1}$

$b = \cos(\sqrt{a} + 1)$

2.12 Problemas para resolver interactivamente en la ventana de comandos

Ejemplo Calcule la diagonal de un cuadrado cuya área es 40 m^2

```
>> a=sqrt(40);
>> d=sqrt(2)*a
d =
8.9443
```



$a^2 = 40$

$d^2 = 2a^2$

1) Calcule el área de un triángulo cuyos lados miden 5, 7, 8 cm.

2) Calcule el área total de un bloque de dimensiones 20, 30, 40 cm

3) Calcule el área total de un cilindro de radio 5 y altura 4 metros

4) El costo mensual **c** en dólares al fabricar una cantidad de **x** artículos está dado por: **c = 50 + 2x**, mientras que el ingreso por la venta de x artículos está dada por: **v= 2.4x**
a) Calcule la ganancia que se obtendrá al fabricar y vender 400 artículos
b) Determine cuantos artículos deben fabricarse y venderse para que el ingreso iguale a los gastos

5) Un modelo de crecimiento poblacional está dado por $f(t) = kt + 2e^{0.1t}$, siendo **k** una constante que debe determinarse y **t** es tiempo en años. Se conoce que f(10)=50. Determine la población en el año 25

6) Un ingeniero desea tener una cantidad de dólares acumulada en su cuenta de ahorros para su retiro luego de una cantidad de años de trabajo. Para este objetivo planea depositar un valor mensualmente. Suponga que el banco acumula el capital mensualmente mediante la siguiente fórmula:

$$a = p\left[\frac{(1+x)^n - 1}{x}\right]$$

a: valor acumulado luego de n depósitos mensuales
p: valor de cada depósito mensual
x: valor nominal del interés mensual
n: número de depósitos mensuales realizados

a) Calcule el valor acumulado en 15 años depositando mensualmente cuotas de 300 con un interés **anual** de 0.04
b) Determine el valor de la cuota que debe depositar mensualmente si desea reunir 200000 en 20 años con un interés **anual** de 0.04
c) Determine cuantos depósitos mensuales de 400 debe realizar para reunir 250000 con una tasa de interés **anual** de 0.04

3 Vectores

3.1 Asignación de vectores

```
>> x=[3, -1, 4, 7, -2]          asignación directa de un vector fila
>> x=[3  -1  4  7  -2]          puede separar con comas o con espacios
>> x(2)=5                       manejo de un componente del vector.
                                   En MATLAB los índices se escriben entre
                                   paréntesis y son numerados desde 1
>> y=x(2: 4)                    para asignar parte de un vector use  (inicio: final)
   y =
       -1    4    7
>> t=[3; -1; 4; 5]              para asignar un vector columna use  ;
   t =
       3
      -1
       4
       5
>> t=x'                         para obtener la transpuesta de un vector use '
>> x=[x, 8];                    agregar un elemento al final del vector
   x =
       3   -1    4    7   -2    8
>> x=[7, x];                    agregar un elemento al inicio del vector
   x =
       7    3   -1    4    7   -2    8
>> x(3)=[ ];                    eliminar el tercer elemento del vector
   x =
       7    3    4    7   -2    8
>> x=[x(1:2),9,x(3:6)]          insertar 9 en la posición 3
   x =
       7    3    9    4    7   -2    8
>> y = 2:1:10                   puede asignar un vector mediante una secuencia
   y =
       2    3    4    5    6    7    8    9   10
                                   En MATLAB las secuencias se escriben con:
                                   valor inicial : incremento : valor final
                                   si el incremento es  1  puede omitirlo
>> y=[2, 5, 4, ...              Para continuar en la siguiente línea use  ...
       7, -3]                   Es la continuación de la línea anterior
>> x=[3, 5, 2, 0]
>> y=2*x                        puede realizar operaciones escalares
>> y=exp(x)                     o crear vectores con funciones
```

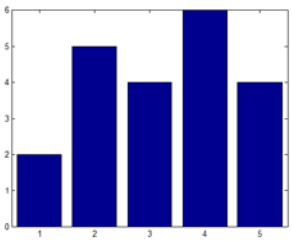
3.2 Algunas funciones para vectores

```
>> x=[2, 5, 4, 6, 4];           un vector
>> n=length(x)                  longitud de un vector
>> t=max(x)                     el mayor valor del vector x
   t =
       6
>> [t,p]=max(x)                 El mayor valor y su posición
    t =
       6
   p =
       4
>> v=[2 4 7 3 5 7 8 6];         Determinar si algún elemento pertenece al vector
>> e=ismember(8,v)
   e =
       1
>> e=ismember(9,v)
   e =
       0
```

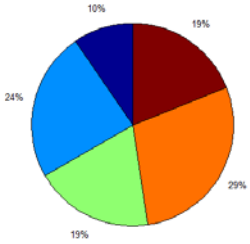
>> [e,p]=ismember(8,v) **Determinar la posición del elemento en el vector**

```
e =
   1
p =
   7
```

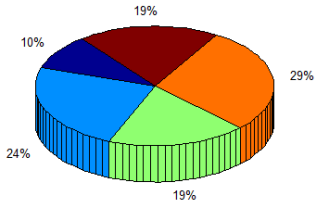
>> **any**(x) determina si el vector contiene algún valor no cero
>> t=find(x) obtiene índices de elementos del vector no ceros
>> t=find(x>3) obtiene los índices de cada elemento > 3
>> t=sum(x) suma de los componentes del vector
>> t=prod(x) producto escalar
>> t=cumsum(x) suma acumulada
>> t=cumprod(x) producto acumulado
>> t=mean(x) media aritmética
>> t=median(x) mediana
>> t=std(x) desviación estándar
>> h=norm(x, inf) norma de fila o columna del vector **x**
>> t=sort(x) ordenamiento ascendente
>> t=dsort(x) ordenamiento descendente
>> bar(x) diagrama de barras de un vector



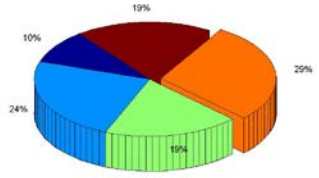
>> hist(x) histograma
>> stairs(x) dibuja x mediante escalones
>> pie(x) gráfico tipo pastel



>> pie3(x) pastel en relieve



>> v=[0,0,0,1,0] índicadores para extraer sectores del pastel
>> pie3(x,v) gráfico tipo pastel 3-d con un sector separado



4 Matrices

4.1 Asignación de matrices

```
>> a = [6  3 ; 5  1]          asignación directa de una matriz 2x2
   a =                              separe elementos con espacios o comas
      6    3                        separe filas con punto y coma
      5    1
>> a(2,1)                     manejo de los componentes de una matriz con índices
                              numerados desde 1:  (fila, columna)
>> a=[2, -3; 5, 1; 0, 7]      una matriz 3x2
>> x=[7, 3]
>> a=[x; x]                   una matriz 2x2
>> b=[5, 6]
>> c=[a; b]                   c es una matriz aumentada 3x2
>> d=[a, b']                  c es una matriz aumentada 2x3
>> x=c(1, :)                  asigne a  x la primera fila de c
>> x=c(: ,1)                  asigne a  x la primera columna de c
>> c(:,2)=[ ]                 elimine la segunda columna de  c
```

4.2 Asignación indirecta de matrices

```
>> x=[ 8 7 9 5 6];
>> p=[2 4 1];                 vector para direccionar al vector  x
>> t=x(p)                     t  contiene los elementos 2, 4 y.1 del vector x
>> a=[4 7 3; 5 7 8; 6 0 9];
>> p=[1 3];                   vector para direccionar las filas de la matriz  a
>> q=[2 3];                   vector para direccionar las columnas de la matriz a
>> t=a(p, q)                  t  contiene las filas 1 y 3, columnas 2 y 3 de a
```

4.3 Matrices especiales

```
>> a=ones(3)                  matriz 3x3 iniciada con unos
>> a=ones(3,5)                matriz 3x5 iniciada con unos
>> a=zeros(4,5)               matriz 4x5 iniciada con ceros
>> a=eye(5)                   matriz identidad 5x5
>> a=magic(4)                 cuadrado mágico 4x4
>> a=hilb(5)                  matriz de Hilberth 5x5
>> x=[2, 5, 3, 7];            un vector
>> a=vander(x)                matriz de Vandermonde 4x4 usando un vector
>> a=[ ]                      matriz nula
>> a = rand(3,4);             matriz 3x4 con números aleatorios
```

4.4 Editor de vectores y matrices

En la ventana **workspace** puede activar el editor de arreglos, similar a una hoja electrónica, con el cual puede modificar con facilidad las dimensiones y el contenido de vectores y matrices.

4.5 Operaciones con matrices

```
>> a=[3, 2; 1, 4];
   a =
      3    2
      1    4
>> b=[8, 6; 5, 7];
>> c=a'                       transpuesta de a
   c =
      3    1
      2    4
>> c=2*a                      producto de un escalar por matriz
   c =
      6    4
      2    8
```

```
>> c=a+b                    suma de matrices
   c =
      11    8
       6   11
>> c=a*b                    producto de matrices
   c =
      34   32
      28   34
>> c=a.*b                   producto elemento por elemento de matrices
   c =                              para operar elemento a elemento use un punto
      24   12                       antes del operador
       5   28

>> c=a^2                    matriz al cuadrado, equivale a:  a*a
>> c=a.^2                   cada elemento de la matriz a, elevar al cuadrado
>> c=a==b                   compare igualdad entre matrices (de igual tamaño)
                            el resultado es una matriz binaria (ceros y unos)
>> c=a~=b                   compare si dos matrices no son iguales
                            el resultado es una matriz binaria (ceros y unos)
>> c=a>3                    compare si cada elemento de a es mayor a  3
                            el resultado es una matriz binaria (ceros y unos)
```

4.6 Funciones para operar con matrices

```
>> a=[1, 2, 3; 4, 5, 6;7, 8, 9];   una matriz para los ejemplos
>> [n,m]=size(a)                   tamaño de la matriz a:  el resultado es un vector
>> n                               número de filas:  3
>> m                               número de columnas:  3
>> isempty(a)                      chequea si un vector o matriz está vacío
>> any(a)                          igual que arriba, pero por columnas de la matriz
>> [f,c]=find(a)                   obtiene los índices de filas y columnas de la matriz
                                   cuyos elementos son no ceros
>> k=rank(a)                       rango de a
>> t=trace(a)                      traza de a
>> d=det(a)                        determinante de a
>> b=inv(a)                        inversa de a
>> h=norm(a, 1)                    norma de columna de la matriz a
>> h=norm(a, inf)                  norma de fila de la matriz a
>> c=cond(a)                       número de condición de la matriz a
>> t=diag(a)                       vector con la diagonal de la matriz  a
>> x=[-2, 0, 6, 5];
>> t=diag(x)                       matriz con el vector  x  en la diagonal
>> t=rot90(a)                      rote  a  90 grados (sentido opuesto al reloj)
>> t=fliplr(a)                     voltee horizontalmente la matriz a
>> t=tril(a)                       obtenga la matriz triangular inferior de a
>> t=triu(a)                       obtenga la matriz triangular superior de a
>> b=[5,-1; 3, 4; 2, 7];
>> b=reshape(b, 2, 3)              reconfigura la matriz  b  de 3x2 a 2x3
>> [t,s]=lu(a)                     descomposición triangular de a en las matrices
>> t                               triangulares t y s tales que  t*s es igual que a
>> s
>> t*s                             se obtiene la matriz  a
>> t=cov(a)                        matriz de covarianza de a
>> e=eig(a)                        valores propios de a
>> p=poly(a)                       polinomio característico de a
>> r=roots(ans)                    valores propios de a
>> help matfun                     liste las funciones para matrices
```

4.7 Funciones adicionales para manejo de matrices

```
>> a=[5,-1; 3, 4; 2, 7]          una matriz
    a =
       5   -1
       3    4
       2    7
>> v=max(a)                      el mayor valor por columnas de la matriz a
   v =
      5    7
>> v=sum(a)                      suma de componentes por columnas de la matriz
>> v=prod(a)                     producto escalar por columnas
>> v=cumsum(a)                   suma acumulada por columnas
>> v=cumprod(a)                  producto acumulado
>> v=mean(a)                     media aritmética por columnas
>> v=median(a)                   mediana por columnas
>> v=std(a)                      desviación estándar por columnas
>> t=sort(a)                     ordenamiento ascendente por columnas
>> t=dsort(a)                    ordenamiento descendente
```

5 Cadenas de caracteres

5.1 Asignación de cadenas de caracteres

```
>> x='problema';                 Asignación directa de la cadena de caracteres

>> t = x(2:6)                    Subcadena
   t =
      roble
>> x=[x,'s']                     Agregar y eliminar caracteres
  x =
     problemas
>> x(8)=''                       Agregar y eliminar caracteres
   x =
      problems

>> x=[x(1:3),'x',x(4:9)]         Insertar caracteres
   x =
      proxblems

>> x='';                         Cadena nula. Puede usar [ ]
```

5.2 Algunas funciones para manejo de cadenas

```
>> x='programa';                 Determinar si un carácter está en una cadena
>> e=ismember('r',x)
   e =
      1
>> [e,p]=ismember('r',x)         También se puede conocer su posición
    e =
       1
   p =
       2

>> x='programa';
>> [e,p]=ismember('a',x)         Entrega la última posición si hay más coincidencias
   e =
      1
   p =
      8
```

```
>> [e,p]=ismember('rta',x)      Examina coincidencia y posición si hay más de uno
    e =
      1    0    1
    p =
      5    0    8

>> x='En esta prueba para esta materia';  Determinar si una cadena está en otra
>> t='esta';
>> e=strfind(x,t)
     e =
        4   21
>> n=length(e)
    n =
       2                          Es la cantidad de coincidencias
>> x='abc';
>> y='abc';
>> t=strcmp(x,y)                 Determinar si dos cadenas son idénticas
    t =
       1
>> x=234;
>> c=num2str(x)                  convertir numérico a cadena de caracteres
    c =
      234
>> c='345';                      convertir cadena de caracteres a numérico
>> n=str2num(c);
     n =
        345

>> a=['abc';'rst';'xyz']          Lista de cadenas (deben tener igual longitud)
    a =
         abc
         rst
         xyz
>> a(2,:)
    ans =
         rst

>> a=char('abc','rs','xyz')      La function char forma listas con cadenas
     a =                          que pueden tener diferente longitud
         abc
         rs
         xyz
>> [n,k]=size(a)
     n =
        3                         Cantidad de cadenas y la mayor longitud
     k =
        3
```

6 Números aleatorios

```
>> x=rand                  genera un número aleatorio entre 0 y 1
>> a=rand(5)               genera una matriz 5x5 con números aleatorios
>> b=rand(4,5)             genera una matriz 4x5 con números aleatorios
>> d=fix(rand*10)+1        transfomación para obtener un entero aleatorio
                           entre 1 y 10
```

7 Ingreso de puntos desde la pantalla mediante el mouse

```
>> ezplot('sin(x)');          ejemplo para tomar puntos desde un gráfico
>> grid on
>> [x,y]=ginput(5);           ingrese 5 puntos desde la pantalla .
                              Presione el botón del mouse para
                              ingresar cada punto
>> x                          observe las abscisas
>> y                          y las ordenadas ingresadas
>> plot(x, y, 'o')            grafique los puntos ingresados
```

8 Polinomios

```
>> a=[2, -3, 0, 5},           define el polinomio  2x^3 – 3x^2 + 5
>> y=polyval(a,4)             evaluación del polinomio con un valor
>> x=roots(a)                 obtenga un vector con las raíces (reales y complejas)
>> t=polyval(a, x(1))         verifique una raíz
>> p=poly(x)                  producto de todas las raíces
>> b=[3, 4, -2];              define el polinomio  3x^2+ 4x - 2
>> c=conv(a,b)                producto de polinomios
>> [c, r]=deconv(a,b);        división de polinomios
>> c                          cociente
>> r                          residuo
>> x=[2 3 5 7 8];             abscisas de puntos (x,y)
>> y=[3.2 4.1 5.8 6.4 6.3];   ordenadas de los puntos
>> z=3.2;                     valor para interpolar, z puede ser un vector
>> u=interp1(x,y,z,'linear')  resultado de la interpolación lineal
>> u=spline(x,y,z)            interpolación con un trazador cúbico
>> a=polyfit(x, y, 2);        polinomio de mínimos cuadrados de grado 2
>> a                          el vector a  contiene los coeficientes
```

9 Manejo simbólico

```
>> syms x;                    definición de variable tipo simbólico
>> 2*x+3*x                    suma algebraica
>> a=[x 5; 3*x 4];            matriz con elementos símbolos
>> t=inv(a)                   su inversa también contiene símbolos
>> f=3*x^2+5*x;               definición simbólica de una función
>> t=factor(f)                factorar la expresión
>> s=expand(t)                expandirla
>> e=taylor(exp(x))           expansión con la serie de Taylor
>> limit(sin(x)/x)            obtencíon de límites de funciones
>> syms y;
>> f=2*x^3+3*y^2              una función de dos variables
>> g=diff(f,x)                derivada parcial
>> u=int(f,x)                 integrar en x
>> f='2*t+1';                 definición de una función en forma literal
>> t=3;
>> y=eval(f)                  evaluación de la función
```

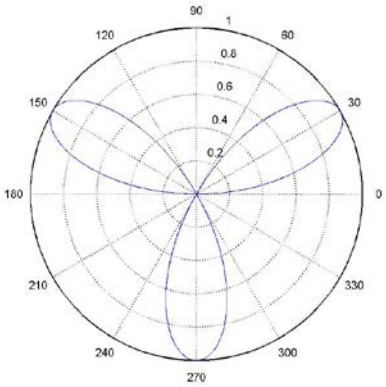
10 Funciones para medir el tiempo de ejecución de algoritmos

```
>> tic;                       Inicia cronómetro
>> toc;                       muestra el tiempo transcurrido

>> tic; a=inv(rand(500, 500)); toc
                              tiempo utilizado en invertir una matriz 500x500
```

11 Graficación

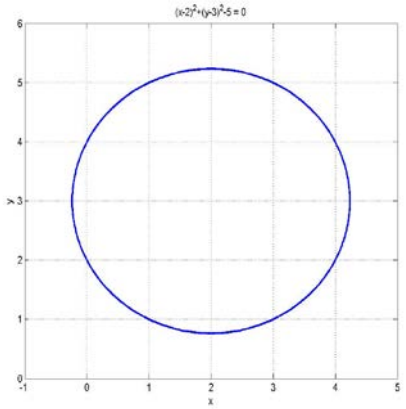
11.1 Gráfico de funciones de una variable

```
>> f='exp(x)-3*x';                 función para el ejemplo (use comillas simples)
>> ezplot(f)                       función básica para graficar f en [-2π, 2π]
>> ezplot(f, [0, 2])               función básica para graficar f en un dominio dado
>> grid on                         colocar cuadrículas en el dibujo

>> x=[0: 0.1: 2*pi];               puntos para evaluar alguna función
>> y=sin(x);                       puntos de la función seno
>> plot(x,y);                      función para graficar la función con línea contínua
>> plot(x,y,'o')                   gráfico con puntos. Puede elegir: o . * + x --
>> plot(x,y,'r')                   cambiar a color rojo. Puede elegir  r,b,y,m,g,w,k
>> plot(x,y,'og')                  grafique con círculos verdes.
>> grid on                         colocar cuadrículas en el dibujo

>> title('seno de x')              incluya un título en el gráfico
>> gtext('seno de x')              posicione el texto en el gráfico con el mouse
>> xlabel('X')                     rotule el eje horizontal
>> ylabel('Y')                     rotule el eje vertical

>> c=[0, 2*pi, -2, 2]              defina la región para el gráfico
>> axis(c)

>> hold on                         superponer siguientes gráficos
>> hold off                        deshabilitar opción anterior
>> clf                             borrar el gráfico

>> figure(1)                       puede tener varias figuras abiertas
                                   cada una en una ventana rotulada con 1, 2, ...
>> subplot(2,3,1)                  puede dividir una figura en subgráficos.
                                   Ej. en 2 filas y 3 columnas. Activando el gráfico 1
>> clf(1)                          borra el gráfico 1
>> clf                             borre todos los gráficos
>> x=[0:0.1:10};
>> y=exp(x);
>> semilogx(x,y)                   graficar en escalas logarítmicas
>> semilogy(x,y)
>> loglog(x,y)                     doble logarítmica
>> grid on

>> a=0:0.01:2*pi;
>> r=sin(3*a);                     'rosa' de 3 pétalos
>> polar(a, r);                    grafique en coordenadas polares
```

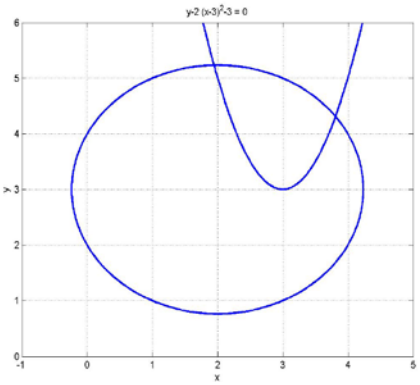


11.2 Gráfico de funciones implícitas y ecuaciones con dos variables

```
>> f='(x-2)^2+(y-3)^2-5';
>> ezplot(f,[-1,5,0,6])        Graficar f en el dominio  -1≤x≤5, 0≤y≤6
>> grid on;                    Colocar cuadrículas
```



```
>> hold on;                    Superponer el siguiente gráfico:
>> g='y-2*(x-3)^2-3';          una parábola y=2(x-3)^2-3  en el mismo dominio
>> ezplot(g,[-1,5,0,6])
```



11.3 Gráfico de funciones definidas en forma paramétrica

```
>> ezplot('sin(t)','cos(t)',[-pi,pi]);          Graficar x=x(t), y=y(t) en -π≤t≤π
>> ezplot('sin(3*t)*cos(t)','sin(3*t)*sin(t)',[0,pi]);   Una rosa de 3 pétalos
```

11.4 Editor de gráficos

Después que el gráfico ha sido realizado puede utilizar las facilidades del editor de gráficos para cambiar las propiedades de las figuras: color, tipo, etc. También puede realizar estadísticas básicas y ajuste de curvas. Adicionalmente puede insertar directamente en el gráfico texto, líneas, flechas, rótulos, etc.

Para habilitar el editor de gráficos seleccione el botón **tools** en la barra de opciones del gráfico y luego elija **edit plot**. Para realizar estadísticas básicas y ajuste de curvas, elija respectivamente **Data Statistics** y **Basic Fitting**

Ejercicio. Obtenga y grafique el polinomio de interpolación, la recta de mínimos cuadrados y el trazador cúbico para un conjunto de datos dados

```
>> x=[1 2 4 5 7];          cinco puntos (x, y) para el ejemplo
>> y=[5 3 6 7 4];
>> plot(x,y,'o')           grafique los datos con círculos
>> grid on                 poner cuadrículas
>> hold on                 superponer los siguientes gráficos

>> a=polyfit(x,y,4);       polinomio de interpolación, 5 puntos: grado 4
>> a                       coeficientes a(1)x^4 + a(2)x^3 + a(3)x^2 + ...
>> z=[1: 0.1: 7];          puntos para evaluar el polinomio
```

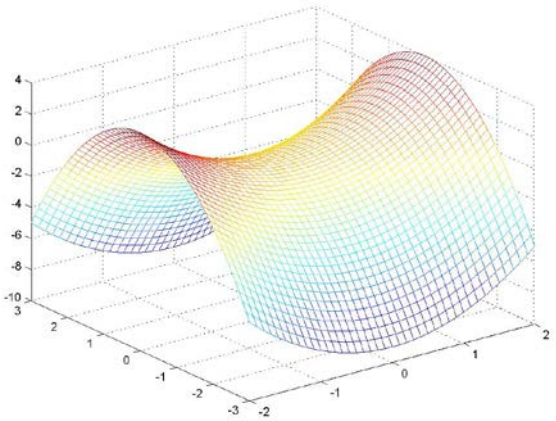
```
>> p=polyval(a,z);          evalúe el polinomio con z  obtenga puntos p
>> plot(z,p)                grafique el polinomio de interpolación

>> b=polyfit(x,y,1);        recta de mínimos cuadrados (grado 1)
>> b                        coeficientes de la recta:  b(1)x + b(2)
>> t=[1 7];                 puntos extremos de la recta (abscisas)
>> q=polyval(b,t);          obtenga las ordenadas respectivas de la recta
>> plot(t,q,'r')           grafique la recta en color rojo

>> s=spline(x,y,z);         evalúe con z  el trazador cúbico y obtenga  s
>> plot(z,s,'g')            grafique el trazador cúbico con verde
>> hold off                 deshabilite la superposición de gráficos
```

11.5 Gráfico de funciones de dos variables

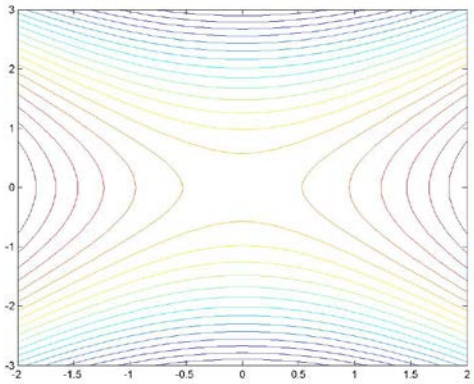
```
>> a=[1 3 2; 5 3 7; 4 5 2];  una matriz 3x3
>> mesh(a);                  graficar los elementos como puntos sobre el plano.
```

El siguiente ejemplo es una referencia para graficar funciones de dos variables
Graficar **$z = x^2 - y^2$, $-2 \le x \le 2$, $-3 \le y \le 3$**

```
>> x=-2:0.1:2;              dominio de la función para el ejemplo
>> y=-3:0.1:3;
>> [u,v]=meshgrid(x,y);      u, v: matrices q' contienen cada par ordenado  x,y
>> z=u.^2 - v.^2;           puntos de la función  z = x^2 – y^2
>> mesh(x, y, z)            gráfico de malla
```



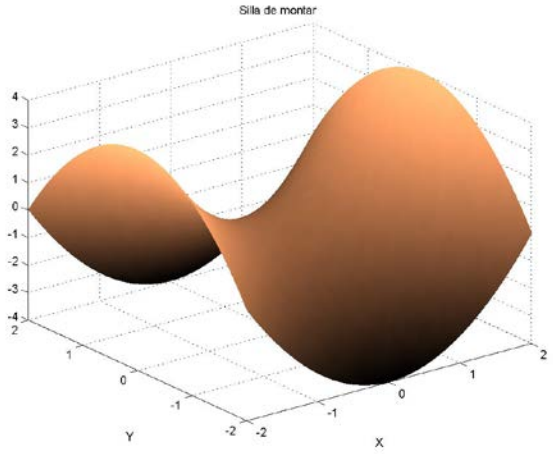
```
>> contour(x, y, z, 20)     gráfico de contorno con 20 curvas de nivel
```



```
>> surfc(x, y, z)           gráfico de superficie y contorno
>> surf(x, y, z)            gráfico de superficie
>> xlabel('X')              rotulación de eje x
```

```
>> ylabel('Y')          rotulación de eje y; también puede usar zlabel
>> title('Silla de montar')   título para el gráfico
>> colormap copper;     color del gráfico; también: gray, jet, pink
>> shading interp;      suavizado del gráfico
```

Gráfico final



Adicionalmente puede usar las opciones del editor de gráficos para editar la figura, rotar, cambiar la perspectiva, insertar títulos, etc.

11.6 Para insertar el gráfico en un documento

Si desea insertar el gráfico elaborado con **MATLAB** en un documento, usualmente escrito en **WORD**, puede seguir el siguiente procedimiento:

1) Elija en la barra de opciones del gráfico el botón **Edit** y luego la opción **Copy figure**
2) Pegue el gráfico en el documento, en el lugar elegido
3) Reduzca el tamaño para cuadrarlo en el documento.

12 Algunas funciones adicionales para métodos numéricos

12.1 Raíces de ecuaciones no lineales

```
>> f='exp(x)-pi*x';
>> x=solve(f)
>> x=eval(x)                    cambia la solución simbólica a real
       x =
         0.5538                 resultados de MATLAB
         1.6385
>> x=fzero(f,2)                 solución de una ecuación con un valor inicial
       x =
         1.6385                 resultado de MATLAB
>> x=fzero(f,[1,2])             solución usando un rango para la raiz
       x =
         1.6385                 resultado de MATLAB
```

12.2 Raíces de sistemas de ecuaciones no lineales

Resolver el sistema: $a^2 + ab - b = 3$, $a^2 - 4b = 5$

```
>> [a,b] = solve('a^2 + a*b - b = 3','a^2 - 4*b = 5');
>> a=eval(a)                    para expresar la solución en forma real
   a =
      -1.0000                   resultados entregados por MATLAB
       1.8284
      -3.8284
```

```
>> b=eval(b)                          resultados entregados por MATLAB
   b =
        -1.0000
        -0.4142
         2.4142
```

12.3 Integración

```
>> f = 'exp(x)-pi*x';
>> v = int(f)                         integración analítica
   v =
      exp(x)-1/2*pi*x^2
>> r = eval(int(f, 0, 2))             integración entre límites
   r =
        0.1059
>> g = 'x*exp(-x)';
>> r = int(g, 0, Inf);                integral impropia
   r =
        1
```

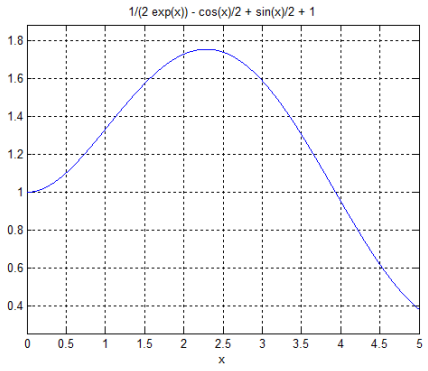
12.4 Diferenciación

```
>> u = diff(f)                        diferenciación con una variable
   u =
      exp(x)-pi
>> f = 'x*exp(x+y)';
>> u = diff(f,'x')                    diferenciación con más variables
   u =
       exp(x+y)+x*exp(x+y)
```

12.5 Ecuaciones diferenciales ordinarias de primer orden

Resolver la ecuación $y' + y - sen(x) - 1 = 0,\ y(0) = 1$

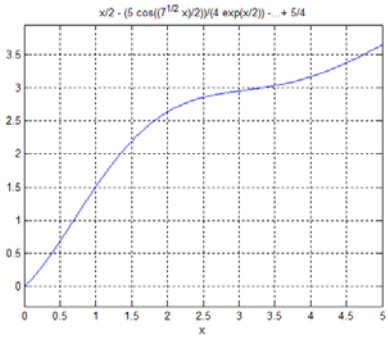
```
>> y=dsolve('Dy+y-sin(x)-1=0','y(0)=1','x')
   y =
     1/(2*exp(x)) - cos(x)/2 + sin(x)/2 + 1
>> ezplot(y,[0,5]),grid on
```



12.6 Ecuaciones diferenciales ordinarias de segundo orden con cond. en el inicio

Resolver la ecuación $y'' + y' + 2y - x - 3 = 0, y(0) = 0, y'(0) = 1$

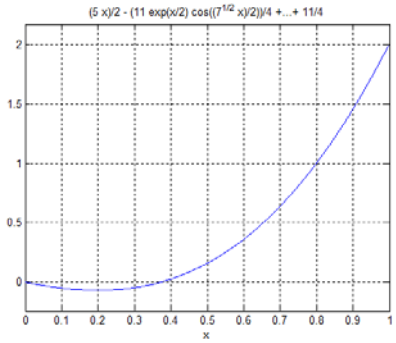
```
>> y=dsolve('D2y+Dy+2*y-x-3=0','y(0)=0,Dy(0)=1','x')
y =                                                    Solución calculada
x/2 - (5*cos((7^(1/2)*x)/2))/(4*exp(x/2)) - (7^(1/2)*sin((7^(1/2)*x)/2))/(28*exp(x/2)) + 5/4
>> ezplot(y,[0,2]), grid on
```



12.7 Ecuaciones diferenciales ordinarias de segundo orden cond. en los bordes

Resolver la ecuación $y'' - y' + 2y - 5x - 3 = 0, y(0) = 0, y(1) = 2$

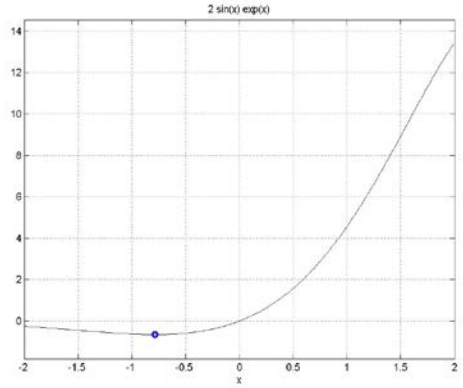
```
>> y=dsolve('D2y-Dy+2*y-5*x-3=0','y(0)=0,y(1)=2','x');
>> ezplot(y, [0, 1])
```



12.8 Optimización

Encontrar un mínimo local de $f(x) = 2\text{sen}(x)e^x$, $-4 \leq x \leq 4$

```
>> f='2*sin(x)*exp(x)';
>> [x,y]=fminbnd(f,-2,2)
x =
  -0.7854
y =
  -0.6448
>> ezplot(f,-2,2), grid on
>> hold on
>> plot(x,y,'o');
```



Ejemplo. Escribir las instrucciones necesarias para encontrar el valor del radio x de un cilindro de 1000 cc de capacidad, de tal manera que el valor del área sea el mínimo:

Primer enfoque:

1) Escribir una función f en términos del radio x
2) Grafique f con ezplot. Localice el intervalo para el mínimo de f(x)
3) Use la función **fminbnd** para obtener el mínimo
   x: radio, h: altura

```
>> f='2*pi*x*1000/(pi*x^2)+2*pi*x^2';
>> ezplot(f,0,10), grid on
>> x=fminbnd(f,4,6)
x =
   5.4193
>> area=eval(f)
area =
  553.5810
```

Segundo enfoque

1) Derive f y obtenga la función a minimizar g.
2) Grafique g con ezplot. Localice el intervalo de la raíz de g(x)=0
3) Use la función **fzero** para obtener la raíz
4) Use la funcion **solve** para obtener la raíz
   x: radio, h: altura

```
>> g=diff(f)
g =  -2000/x^2+4*pi*x
>> x=fzero(char(g),[4,6])
x =
   5.4193

>> x=solve(g)
x =
[                                  5/pi*4^(1/3)*(pi^2)^(1/3)]
[ -5/2/pi*4^(1/3)*(pi^2)^(1/3)+5/2*i*3^(1/2)/pi*4^(1/3)*(pi^2)^(1/3)]
[ -5/2/pi*4^(1/3)*(pi^2)^(1/3)-5/2*i*3^(1/2)/pi*4^(1/3)*(pi^2)^(1/3)]
>> x=eval(x)
x =
   5.4193
  -2.7096 + 4.6932i
  -2.7096 - 4.6932i
```

13 Bibliografía

The MathWorks, Inc. *Using MATLAB Computation, Visualization, Programming*, version 7