Unidad 1: Modelado Matemático y herramientas para su resolución

Prof.: Dr. Juan Ignacio Manassaldi

JTP: Ing. Amalia Rueda

Tipos de usuarios de software

- Existen dos tipos de usuarios de software:
  - **Usuario tradicional o limitado**: Se limita a las capacidades que encuentran en el modo estándar de operación del software existente.

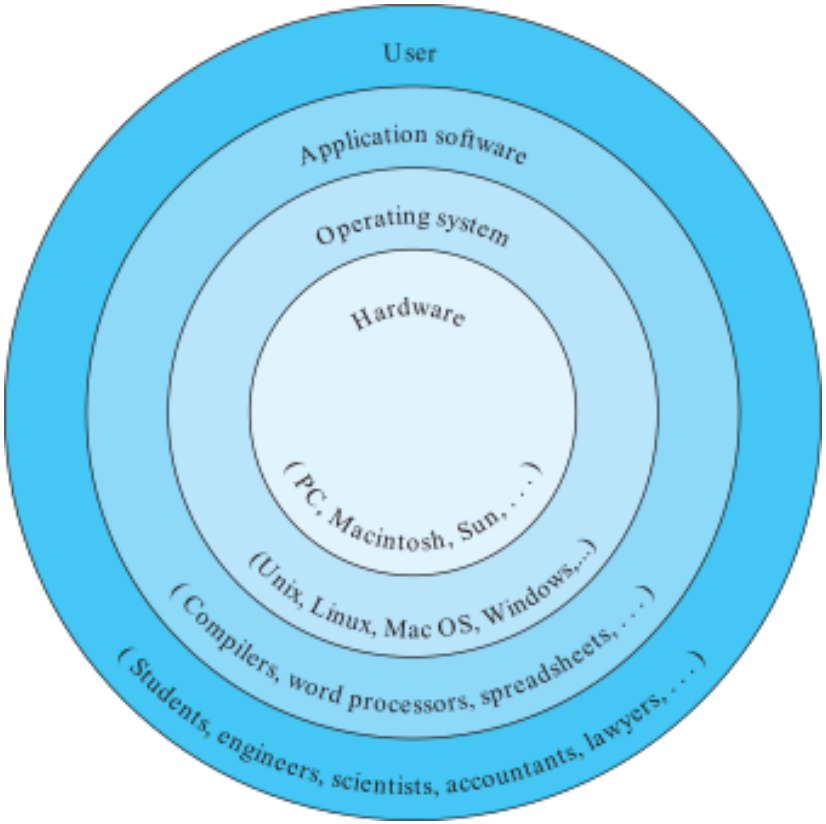
    El problema debe adaptarse al software.
  - **Usuario avanzado o potente**: Amplía las capacidades nativas del software mediante herramientas de programación.

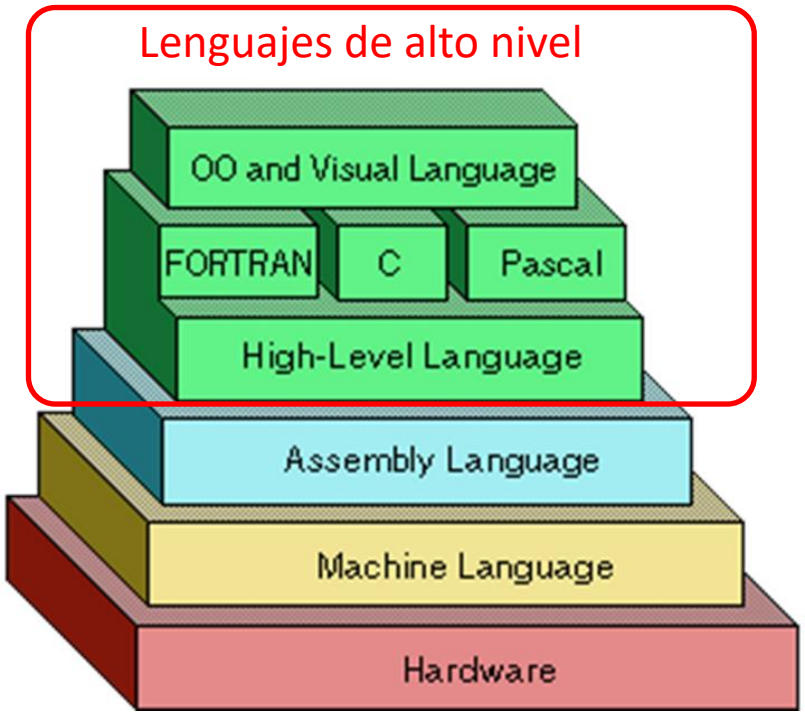
    El software se adapta al problema.

Un usuario avanzado puede comportarse como un usuario tradicional si el problema lo amerita.

Niveles de Programación

- Los programas computacionales son únicamente conjuntos de instrucciones que dirigen a la computadora para realizar una cierta tarea.





Softwares tradicionales para la implementación de métodos numéricos

Excel + VBA

MATLAB

Scilab

GNU Octave

Python

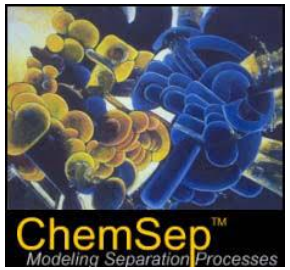
C y C++

Softwares tradicionales para la implementación de métodos numéricos

- El listado anterior corresponde a softwares de propósito general y sirven para la resolución de problemas numéricos de la mayoría de las ingenierías.
- Por otro lado, cada especialidad tiene también softwares para la resolución puntual de sus problemas específicos (software enlatado).
- En este tipo de herramientas informáticas la resolución ya viene programada internamente.
- Un ejemplo clásico de ingeniería química son los **simuladores de procesos:**

aspentech HYSYS

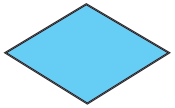
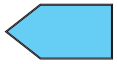
DWSIM



Programación estructurada

- Es un conjunto de reglas que desarrollan en el programador los hábitos para lograr un buen estilo.
- La idea clave detrás de la programación estructurada es que cualquier algoritmo numérico requiere tan sólo de tres estructuras de control fundamentales: **secuencia, selección y repetición**.
- Un diagrama de flujo es una representación visual o gráfica de un **algoritmo**.
- Otra manera de expresar algoritmos, y que constituye un puente de unión entre los diagramas de flujo y el código de la computadora, es el **pseudocódigo**. En esta técnica se utilizan expresiones semejantes a las del código, en lugar de los símbolos gráficos del diagrama de flujo.

Programación estructurada

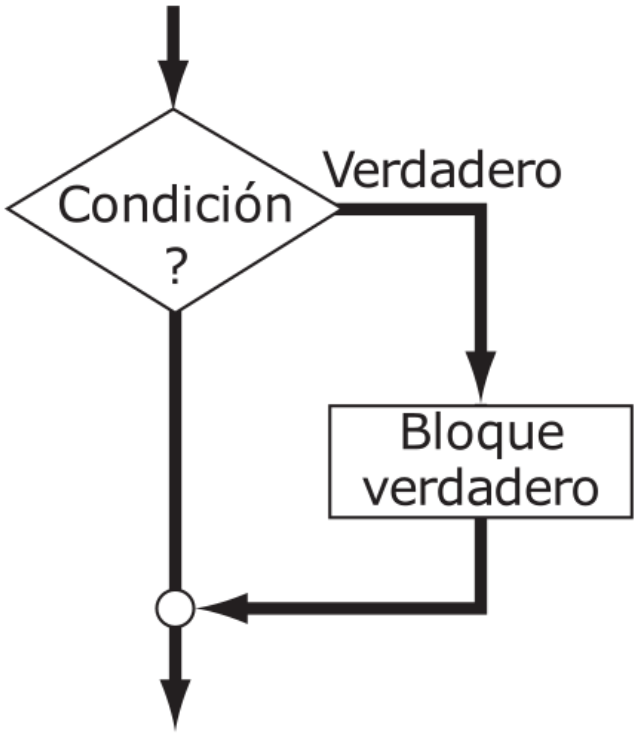
SÍMBOLO	NOMBRE	FUNCIÓN
	Terminal	Representa el inicio o el final de un programa.
	Líneas de flujo	Representan el flujo de la lógica. Los arcos en la flecha horizontal indican que ésta pasa sobre las líneas de flujo verticales y no se conecta con ellas.
	Proceso	Representa cálculos o manipulación de datos.
	Entrada/Salida	Representa entrada o salida de datos e información.
	Decisión	Representa una comparación, una pregunta o una decisión que determina los caminos alternativos a seguir.
	Unión	Representa la confluencia de líneas de flujo.
	Conexión de fin de página	Representa una interrupción que continúa en otra página.
	*Ciclo* de cuenta controlada	Se usa para *ciclos* que repiten un número predeterminado de iteraciones.

Representación lógica

- **Secuencia**. Expresa la trivial idea de que, a menos que se indique otra cosa, el código debe realizarse instrucción por instrucción.
- **Selección**. La selección nos ofrece un medio de dividir el flujo del programa en ramas considerando el resultado de una condición lógica.
  - Una sola alternativa: IF/THEN
  - Dos alternativas: IF/THEN/ELSE
  - Múltiples alternativas.

Una sola alternativa: IF/THEN

Diagrama de flujo

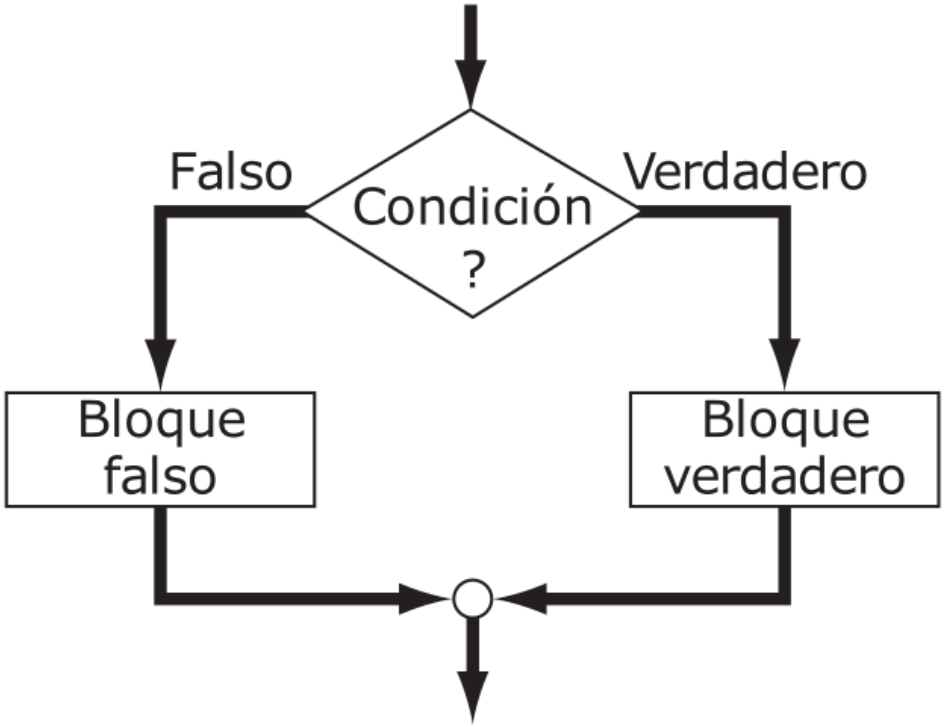


Pseudocódigo

IF *condición* ***THEN***
Bloque verdadero
ENDIF

Dos alternativas: IF/THEN/ELSE

Diagrama de flujo

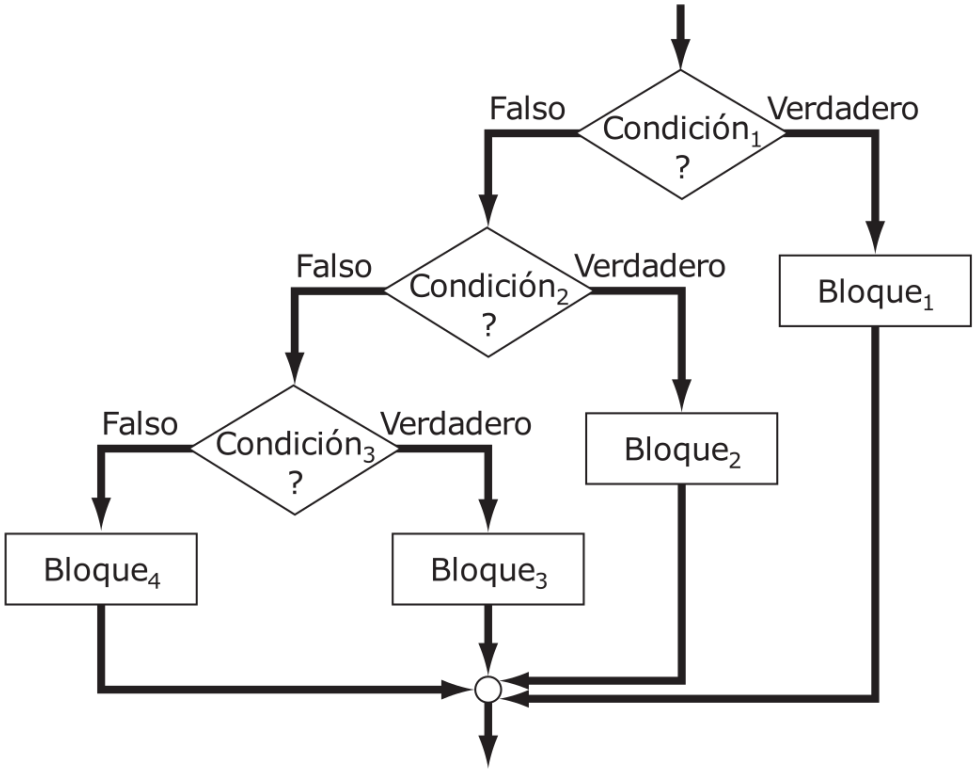


Pseudocódigo

IF *condición* ***THEN***
Bloque verdadero
ELSE
Bloque falso
ENDIF

Multiples alternativas

Diagrama de flujo

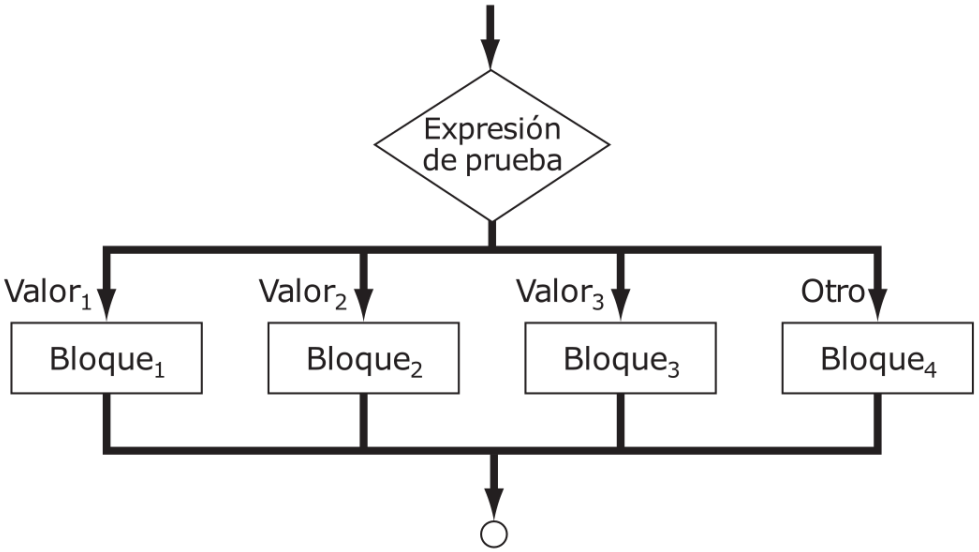


Pseudocódigo

IF *condición1* ***THEN***
Bloque1
ELSEIF *condición2*
Bloque2
ELSEIF *condición3*
Bloque3
ELSE
Bloque4
ENDIF

Multiples alternativas

Diagrama de flujo



Pseudocódigo

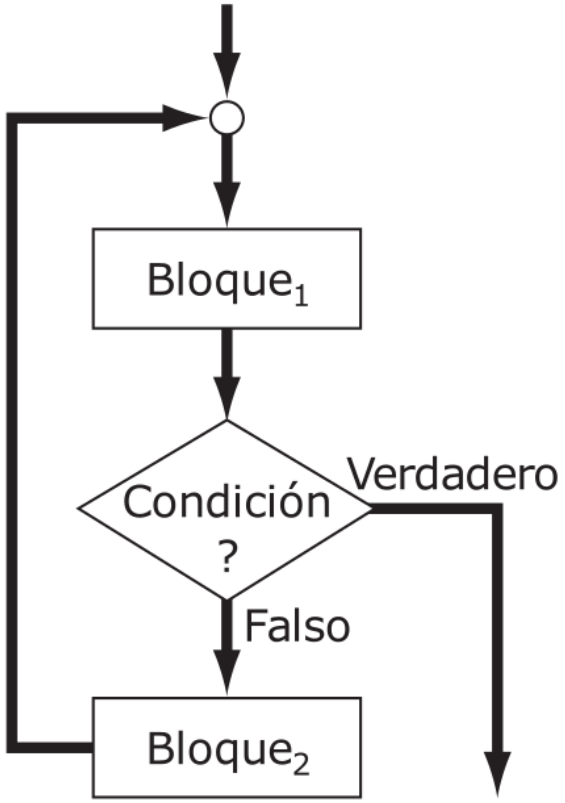
SELECT CASE *Expresión de prueba*
CASE *Valor1*
Bloque1
CASE *Valor2*
Bloque2
CASE *Valor3*
Bloque3
CASE ELSE
Bloque4
END SELECT

Representación lógica

- **Repetición**. La repetición nos proporciona una manera de llevar a cabo instrucciones repetidamente. Las estructuras resultantes, llamadas **loops** o **ciclos**, se presentan en dos formas distintas que se diferencian por la manera en que terminan.
  - El primer tipo, y el fundamental, es el llamado **loop de decisión** (o loop *DOEXIT)* debido a que termina basándose en el resultado de una condición lógica.
  - El segundo tipo es el **loop controlado por contador** (o loop DOFOR) que realiza un número determinado de repeticiones o iteraciones.

Loop de decisión

Diagrama de flujo

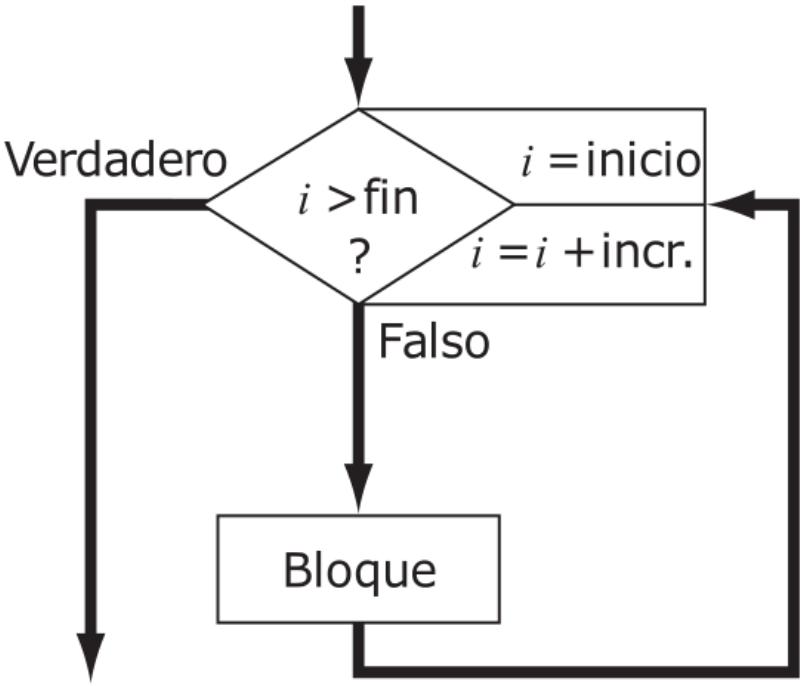


Pseudocódigo

DO
Bloque1
IF *condición* ***EXIT***
Bloque2
ENDDO

Loop controlado por contador

Diagrama de flujo

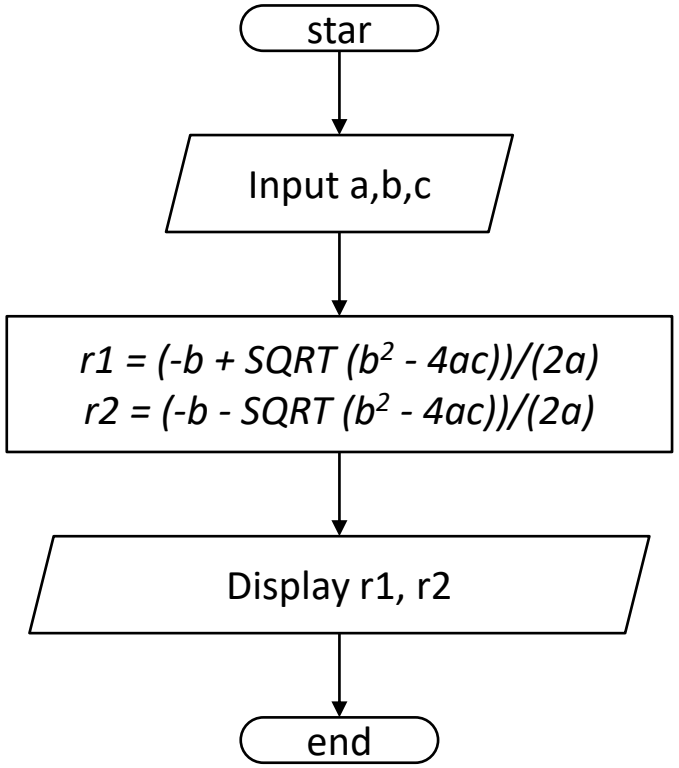


Pseudocódigo

DOFOR *i = inicio, incremento, fin*
Bloque
ENDDO

Algoritmo para las raíces de la ecuación cuadrática (I)

Diagrama de flujo



Pseudocódigo

INPUT *a, b, c*
r1 = (-b + SQRT (b^2 - 4ac))/(2a)
r2 = (-b - SQRT (b^2 - 4ac))/(2a)
DISPLAY *r1, r2*

Algoritmo para las raíces de la ecuación cuadrática (II)

```
INPUT a, b, c
IF a = 0 THEN
        IF b ~= 0 THEN
                r1 = –c/b
                DISPLAY r1
        ELSE
                DISPLAY "No hay solución"
        ENDIF
ELSE
        discr = b^2 – 4*a*c
        IF discr ≥ 0 THEN
                r1 = (–b + Sqrt(discr))/(2 * a)
                r2 = (–b – Sqrt(discr))/(2 * a)
                DISPLAY r1, r2
        ELSE
                DISPLAY "Raíces Complejas"
        ENDIF
ENDIF
```

Algoritmo para calcular la sumatoria de un vector de datos

```
INPUT v , n
suma = 0
DOFOR i = 1, 1, n
suma = suma + v(i)
ENDDO
DISPLAY suma
```

```
INPUT v , n
suma = v(1)
DOFOR i = 2, 1, n
suma = suma + v(i)
ENDDO
DISPLAY suma
```

v: vector de trabajo
n: tamaño del vector

```
DOFOR i = inicio, incremento, fin
Bloque
ENDDO
```

Programación modular

- Los programas de computación se dividen en subprogramas más pequeños, o módulos que pueden desarrollarse y probarse por separado. A esta forma de trabajar se le llama programación modular.
- La principal cualidad de los módulos es que son tan independientes y autosuficientes como sea posible. Además, en general, están diseñados para llevar a cabo una función específica y bien definida, y tienen un punto de entrada y un punto de salida.
- Finalmente combinando programación estructurada y modular es posible resolver la mayoría de los problemas de ingeniería.
- **A lo largo del año iremos programando “Módulos” o “Funciones” que aplican diferentes algoritmos de resolución. Luego, la combinación de estos nos permitirá resolver los problemas planteados.**
- **Obviamente también se aprovecharán los módulos ya elaborados e incluidos de manera nativa en el software.**

Algoritmo para calcular el promedio de un vector de datos

```
INPUT v , n
suma = v(1)
DOFOR i = 2, 1, n
suma = suma + v(i)
ENDDO
DISPLAY suma
```

```
INPUT v
n=SIZE(v)
suma = v(1)
DOFOR i = 2, 1, n
suma = suma + v(i)
ENDDO
DISPLAY suma
```

```
FUNCTION suma=SUM(v)
n=SIZE(v)
suma = v(1)
DOFOR i = 2, 1, n
suma = suma + v(i)
ENDDO
ENDFUNCTION
```

Modulo o función para realizar la sumatoria de un conjunto de valores

```
INPUT u
promedio = SUM(u)/ SIZE(u)
DISPLAY promedio
```

Pseudocodigo para el calculo de un promedio aprovechando los módulos disponibles

Algoritmo para calcular el promedio de un vector de datos

```
FUNCTION promedio=PROM(v)
n=SIZE(n)
suma = v(1)
DOFOR i = 2, 1, n
suma = suma + v(i)
ENDDO
promedio = suma/n
ENDFUNCTION
```

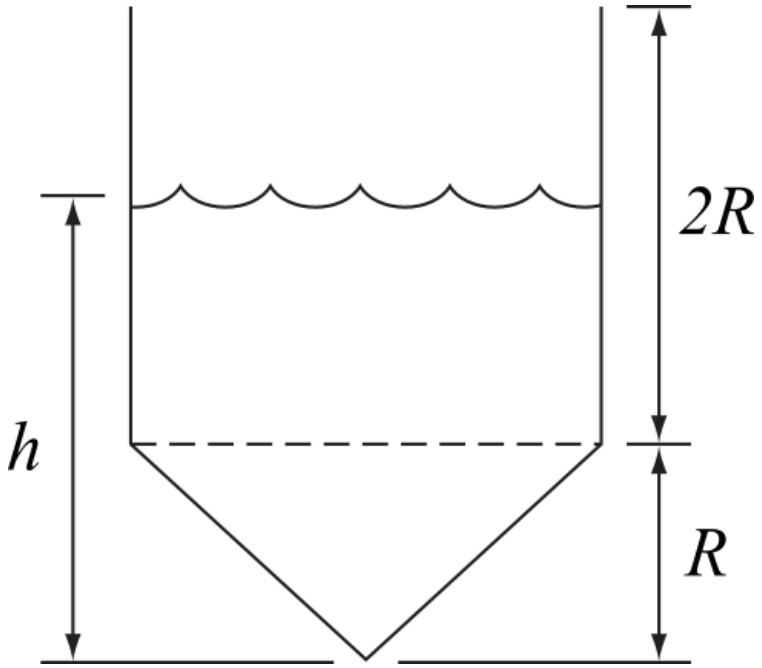
Modulo o función para calcular el promedio de un conjunto de valores

Modulo para la resolución de una ecuación cuadrática

```
FUNCTION [r1 r2]=CUADRATICA(a,b,c)
IF a = 0 THEN
        IF b ~= 0 THEN
                r1 = –c/b; r2 =[]
        ELSE
                DISPLAY "No hay solución"
                r1 = []; r2 =[]
        ENDIF
ELSE
        discr = b^2 – 4*a*c
        IF discr ≥ 0 THEN
                r1 = (–b + Sqrt(discr))/(2 * a)
                r2 = (–b – Sqrt(discr))/(2 * a)
        ELSE
                 DISPLAY "Raíces Complejas"
                r1 = []; r2 =[]
        ENDIF
ENDIF
ENDFUNCTION
```

Ejemplo: Volumen de un tanque

Escriba un procedimiento bien estructurado de función para calcular el volumen del tanque como función de los valores dados de *R (radio del tanque)* y *h (altura de liquido)*. Utilice estructuras de control de decisiones (como If/Then, ElseIf, Else, End If). Diseñe la función de modo que produzca el volumen en todos los casos y además genere un mensaje de error ("Sobrepasado") si se rebasa la altura del tanque.



IF *condición1* ***THEN***
Bloque1
ELSEIF *condición2*
Bloque2
ELSEIF *condición3*
Bloque3
ELSE
Bloque4
ENDIF

Ejemplo: Volumen de un tanque

```
FUNCTION [V]=TANQUE(h,R)
IF h > 3R THEN
        V = PI*R^2*2R + 1/3*PI*R^2*R
        DISPLAY "Sobrepasado"
ELSEIF h =< R
          V = 1/3*PI*h^2*h
ELSE
        V = 1/3*PI*R^2*R + PI*R^2*(h-R)
ENDIF
ENDFUNCTION
```

Matemática Superior Aplicada

2025

Prof.: Dr. Juan Ignacio Manassaldi

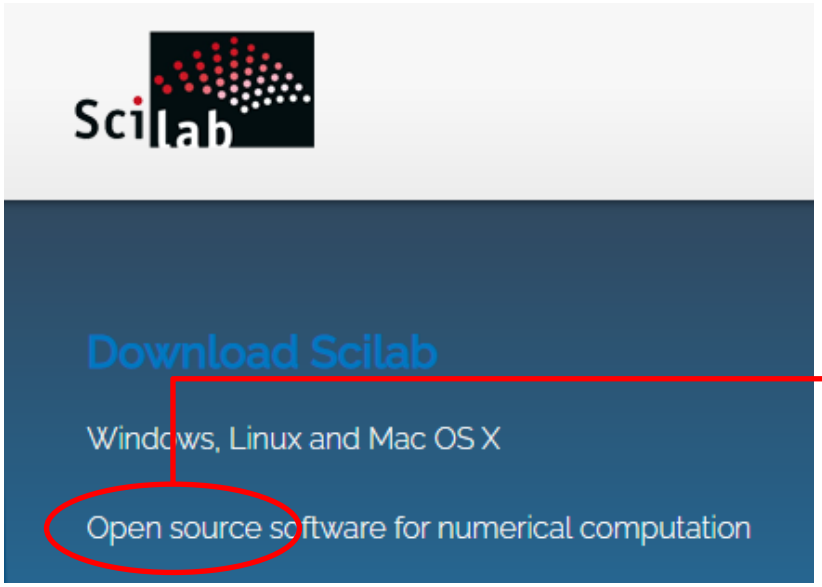
JTP: Ing. Amalia Rueda

Wiki

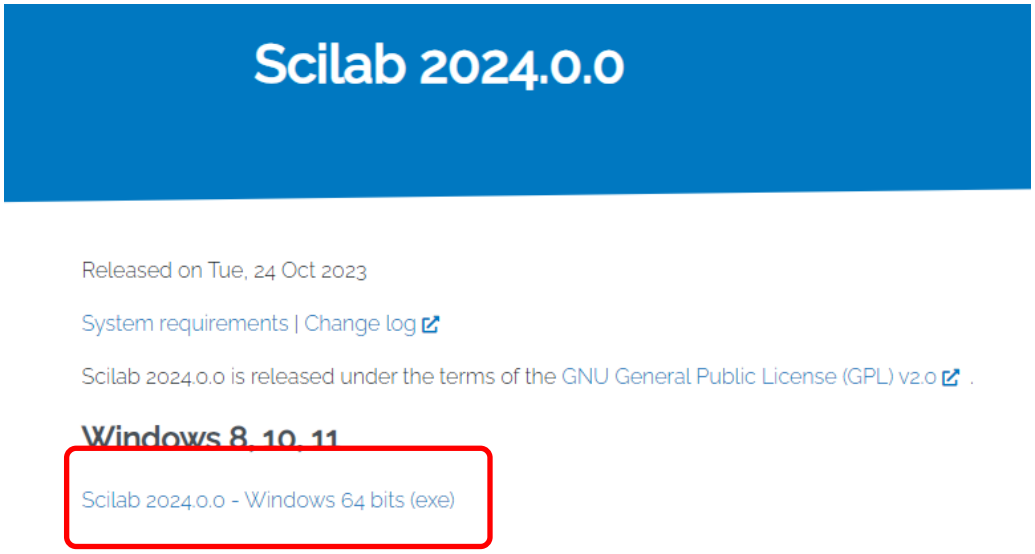
- Scilab es un software para análisis numérico, con un lenguaje de programación de alto nivel para cálculo científico.
- Es desarrollado por Scilab Enterprises, bajo la licencia CeCILL, compatible con la GNU General Public License.
- Las características de Scilab incluyen:
  - Análisis numérico
  - Visualización 2-D y 3-D
  - Optimización
  - Análisis estadístico
  - Diseño y análisis de sistemas dinámicos
  - Procesamiento de señales
  - Interfaces con Fortran, Java, C y C++.
  - Xcos, un editor gráfico para diseñar modelos de sistemas dinámicos.

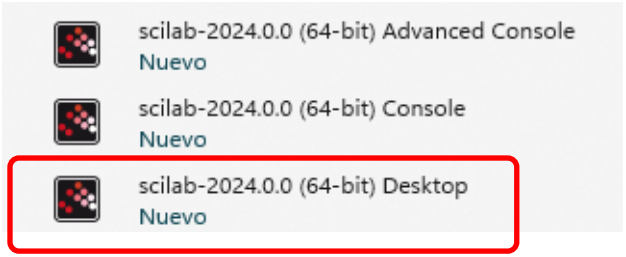
Descarga e instalación

https://www.scilab.org/



El código abierto es un modelo de desarrollo de software basado en la colaboración abierta.





Pseudocodigo a Scilab: **IF/THEN**

IF condición THEN
Bloque verdadero
ENDIF

```
if b ~= 0 then
r1 = -c / b;
end
```

Pseudocodigo a Scilab: **IF/THEN/ELSE**

IF condición THEN
Bloque verdadero
ELSE
Bloque falso
ENDIF

```
if a < 0 then
b = sqrt(abs(a));
else
b = sqrt(a);
end
```

Pseudocodigo a Scilab: **IF/THEN/ELSEIF**

IF condicion$_1$ THEN	`if class == 1 then`
Bloque$_1$	`  x = x + 8;`
ELSEIF condicion$_2$	`elseif class < 1 then`
Bloque$_2$	`  x = x - 8;`
ELSEIF condicion$_3$	`elseif class < 10 then`
Bloque$_3$	`  x = x - 32;`
ELSE	`else`
Bloque$_4$	`  x = x - 64;`
ENDIF	`end`

Pseudocodigo a Scilab: **CASE**

SELECT CASE Expresión de prueba
CASE Valor₁
Bloque₁
CASE Valor₂
Bloque₂
CASE Valor₃
Bloque₃
CASE ELSE
Bloque₄
END SELECT

```
select a + b
case 1
    x = -5;
case 2
    x = -5 - (a + b) / 10;
case 3
    x = (a + b) / 10;
else
    x = 5;
end
```

Pseudocodigo a Scilab: **DOEXIT**

DO
Bloque₁
IF condicion EXIT
Bloque₂
ENDDO

```
while (1)
    i = i + 1
    if i >= 10
        break
    end
    j = i
end
```

Pseudocodigo a Scilab: **DOEXIT**

DO
Bloque$_1$
IF condicion EXIT
Bloque$_2$
ENDDO

```
while i < 10
    i = i + 1
    j = i
end
```

Pseudocodigo a Scilab: **LOOP CONTROLADO POR CONTADOR**

DOFOR i = inicio, fin, incremento
Bloque
ENDDO

```
for i = 1:2:10
    x = x + i;
end
```