

Bloque 1

Conceptos Básicos

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos



Conceptos básicos

El objetivo de la asignatura es construir **programas**, mediante el seguimiento de **algoritmos** que resuelva un problema.

¿Qué es un **algoritmo**?

Conjunto de reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad.

Dados un estado inicial y normalmente unos datos de entrada, siguiendo dichos pasos sucesivos se debe llegar a un estado final y a la obtención de una solución.

Ejemplo de algoritmo para llenar de aire una rueda:

1. Aflojar la válvula
2. Colocar la bomba
3. Mientras la rueda no tenga la presión correcta,
 - 3.1 Bombear aire
 - 3.2 Comprobar la presión de la rueda
4. Retirar la bomba
5. Apretar la válvula



Conceptos Básicos

¿Qué es un **programa**?

Es un conjunto de *instrucciones* (también llamadas *sentencias*) que puede ser ejecutado por un ordenador y realizar una tarea específica.

Un programa que se ejecute en las mismas condiciones debe producir el mismo resultado.

Las *instrucciones* son expresiones con *variables*, *literales*, *operadores aritméticos*, *operadores de relación*, llamadas a otras funciones (predefinidas en las librerías del lenguaje de que se trate o construidas por el programador), formalmente bien construidas y con sentido.

Las *instrucciones* se escriben normalmente una debajo de otra y se ejecutan de forma secuencial, aunque algunas sentencias cambian el flujo de ejecución de un programa.

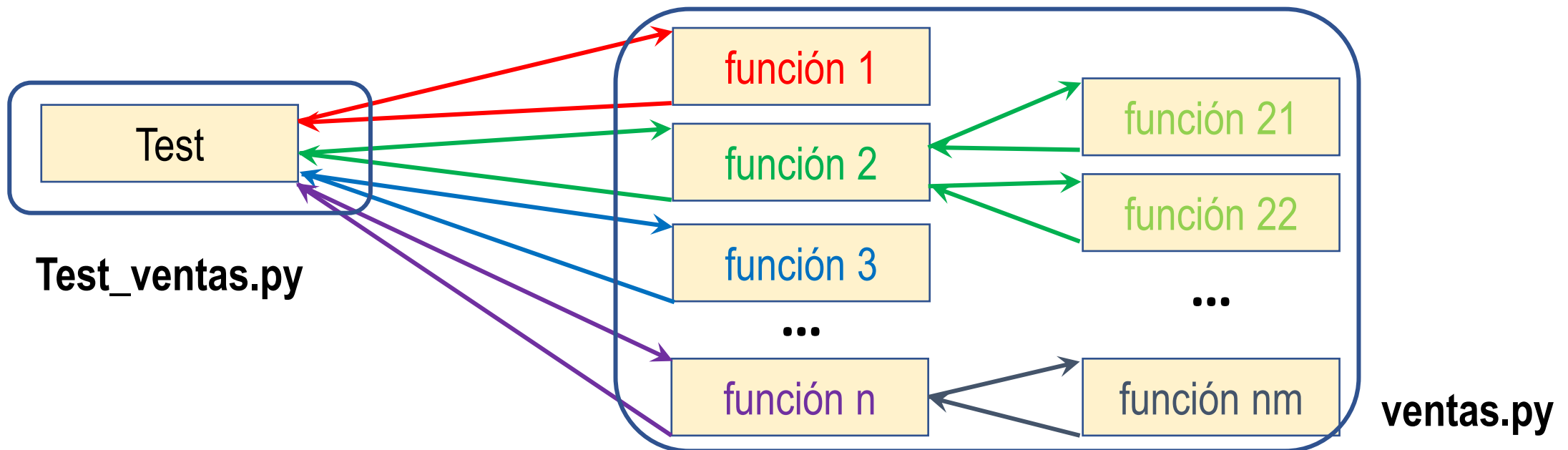
Ejemplo de programa (Python) que calcula el área de un triángulo a partir de su base y altura:

```
print("Cálculo del área de un triángulo")    # print visualiza
base=float(input("Teclea la base: "))        # input permite introducir valores por teclado
altura=float(input("Teclea la altura: "))
if base>=0 and altura>=0:                    # if evalúa una condición y redirige el flujo
    print("El área vale:",base*altura/2)
else:
    print("No se puede calcular el área con la base o la altura negativa")
```



Unidades de Programación

- Generalmente cualquier lenguaje de programación se organiza unidades de código *relativamente pequeñas* que atienden a un propósito concreto y permiten organizar mucho mejor los programas informáticos.
 - En el caso de **Python** estas unidades se denominan en su mayoría **funciones**, aunque también existe otras unidades.
 - En el caso de **Java** en lugar de funciones se denominan **métodos**.





Estructura de las funciones en Python

- Se componen
 - Tiene una *cabecera* en la que se escribe en el siguiente orden:
 - La palabra **def** (palabra reservada de Python)
 - El nombre de la función (inventada por el programador)
 - Los **parámetros formales** (entre paréntesis y separados por coma “,”, inventados por el programador)
 - Termina en dos puntos “:”
 - Contiene *sentencias o instrucciones* que pueden ser:
 - Expresiones con variables, literales, sentencias, llamadas a otras funciones (*predefinidas en las librerías de Python o construidas por el programador*)...
 - Cuando la función devuelve, al menos, un valor incluye la sentencia **return**
 - Una vez construida una función para poder usarla basta escribir su nombre con los correspondientes valores en los parámetros (se denominan **parámetros reales**)



Estructura de las funciones en Python

Ejemplo de función (Python) que devuelve las raíces de una ecuación de segundo grado:

Entrada: Recibe como parámetros formales tres valores numéricos (a, b y c) de tipo `float` (real)

Salida: Una tupla con las raíces de tipo `float`. Sino se pueden calcular, la función devuelve la tupla `(None, None)`

```
from math import sqrt ← Uso de funciones ya suministradas por el lenguaje
def raíces(a:float,b:float,c:float)->(float,float): ← Cabecera
    raiz1=None
    raiz2=None
    discriminante=b**2-4*a*c
    if a!=0 and discriminante>=0:
        raiz1=(-b+sqrt(discriminante))/(2*a)
        raiz2=(-b-sqrt(discriminante))/(2*a)
    return (raiz1,raiz2)
```

Instrucciones



Definición de variable en Python

Variable = Elemento de un programa que permite almacenar valores que será necesario utilizar después

- Se definen en el lugar del programa que sea necesario, pero antes de su uso, asignándoles un valor por defecto.
- Se nombran con *letras*, *número* o el símbolo de subrayado (_) . Pero no pueden comenzar por número y no pueden contener espacios en blanco.
- Se escriben en *minúsculas* y si la denominación de la variable es compuesta se separa cada palabra por _
año, importe_iva, raiz1, raiz2, _flag, nombre_de_pila,
- Tampoco puede denominarse igual que una palabra reservada de Python
'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'False', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'None', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'True', 'try', 'while', 'with', 'yield'



Tipos de datos que almacenan las variables

Las variables tienen un tipo que depende de valor que almacena en cada momento. Los tipos son los siguientes:

- **str**: abreviatura de String (en inglés) que significa cadena o literal. Para mensajes, frases y cualquier tipo de literal.
-nombre = "Manuel" -ape_nom= 'García Gonzalez, Ana' -sexo='M'
- **int**: Para valores numéricos enteros (sin decimales).
-edad=18 -año=2024 -número_lotería=45330
- **float**: Para almacenar número reales.
-importe=234.32 -estatura=1.78 -peso=54.2
- **complex**: Para almacenar números complejos.
-nc=complex(3,4) o nc=3+4j la parte real es 3 y la imaginaria es 4



Tipos de datos que almacenan las variables

- **boolean**: Para almacenar sólo dos valores: *True* o *False* y sirven para conocer el resultado de una evaluación.
-esbisiesto=*True* -existe_valor=*False*
- **nulo/vacio**: *None* para inicializar una variable **sin valor** predeterminado.
-resultado=*None* -fecha_defunción=*None* (*todavía está vivo*)
- **contenedor**: Son agrupaciones de datos que veremos pronto:
 - *Tuplas*
 - *Listas*
 - *Conjuntos*
 - *Diccionarios*



Operadores Aritméticos o Algebraicos

- Suma: + (*también sirve para concatenar cadenas*)
- Resta o cambio de signo de una expresión: -
- Multiplicación: *
- División: /
- División entera: //
- Resto de la división entera: %
- Potencia: **

Ejemplos:

$3*4**2 \rightarrow 48$ (*más adelante hay una diapositiva con la prelación de operadores*)

$(3*4)**2 \rightarrow 144$

$\text{numero}\%2 \rightarrow$ (devuelve 0 si número contiene un valor par y 1 si el valor es impar)

$7/2 \rightarrow 3.5$; $7//2 \rightarrow 3$ (sólo la parte entera) ; $7\%2 \rightarrow 1$ (el resto de la división entera)



Expresiones

Cualquier combinación de variables, valores, operadores matemáticos (,),+,-,*,/,%, funciones u otras expresiones con sentido.

Algunos *ejemplos* de expresiones con su tipo:

- `"Hola Don Pepito, hola Don José"` \rightarrow str
- `'Hola Don Pepito' + ' hola Don José'` \rightarrow str
- `23` \rightarrow int
- `38*4` \rightarrow int
- `(2+4)*6.0+(-3*4%2)` \rightarrow float
- `56/(2-4)*3` \rightarrow float
- `mi_tangente=sin(angulo)/cos(angulo)` \rightarrow float
- `raiz1=-b+sqrt(pow(b,2)-4*a*c)` \rightarrow float



Operadores Relacionales entre expresiones (devuelve True o False)

Permite evaluar los valores que contienen dos expresiones:

- Igual que: `==` (*son dos pulsaciones de teclado*)
- Menor que: `<`
- Mayor que: `>`
- Distinto que: `!=` (*son dos pulsaciones de teclado y en ese orden*)
- Menor o igual que: `<=` (*son dos pulsaciones de teclado y en ese orden*)
- Mayor o igual que: `>=` (*son dos pulsaciones de teclado y en ese orden*)

Ejemplos:

`5 > 6` → `False`

`"mensaje" == "texto"` → `False`

`"mensaje" != "texto"` → `True`

`9 >= 2*3` → `True`



Operadores Lógicos entre expresiones relacionales

Permite evaluar el resultado de expresiones de relaciones de forma simultánea:

- **and**: conjunción lógica (y) \rightarrow devuelve **True** si *todas* las expresiones son ciertas
 - **or**: disyunción lógica (o) \rightarrow devuelve **True** si *al menos* una expresión es cierta
 - **not**: negación \rightarrow devuelve **True** si la expresión evalúa **False** y viceversa
- A la hora de resolver una expresión con varios operadores lógicos “**and**” tiene prelación sobre “**or**”, pero puede ser alterada con el uso de paréntesis “()”.
 - El operador “**not**” afecta a la expresión inmediatamente a la que precede. Si se quiere un ámbito mayor hay que usar paréntesis para abarcar más expresiones.

Ejemplos:

$9 > 2 * 3$ **and** $5 > 6 \rightarrow \text{False}$

$9 > 2 * 3$ **or** $5 > 6 \rightarrow \text{True}$

not $(5 < 6) \rightarrow \text{False}$

not $(9 > 2 * 3)$ **or** $5 > 6 \rightarrow \text{False}$

not $((\text{exp1 or exp2}) \text{ and } (\text{exp3 or exp4})) \text{ and exp5 or exp6 or not}(\text{exp7 and exp8})$



Prelación de los operadores más habituales

**	Exponenciación o potencia
*, /, //, %	Multiplicación, multiplicación de matrices, división, "floor division", resto
+, -	Adición y sustracción
in, not in, is, is not, <, <=, >, >=, ==, !=	Comparaciones, identificación y pertenencia
not	"no" booleano
and	"y" booleano
or	"o" booleano
=	operador de asignación

- El uso de paréntesis permite alterar la prelación.
- La prelación de dos operadores en el mismo nivel se resuelve de izquierda a derecha, salvo el operador de asignación (=) que es de derecha a izquierda.



Ejemplo de un programa Python que resuelve una ecuación de 2º grado

```
from ecuación_segundo_grado import *  
print("Resolución de una ecuación de segundo grado")  
c1=float(input("Teclea coeficiente de x cuadrado: "))  
c2=float(input("Teclea coeficiente de x: "))  
c3=float(input("Teclea el término independiente: "))  
print("Las raíces son: ",raíces(c1,c2,c3))
```

nombre-> "test_ecuación_segundo_grado.py"

nombre-> "ecuación_segundo_grado.py"

Función que calcula las raíces de una ecuación de segundo grado.

Entrada: -Recibe como parámetros tres valores numéricos de tipo real (float)

Salida: -Una tupla con las raíces de tipo (float). Sino se puede calcular, la tupla es (None,None)

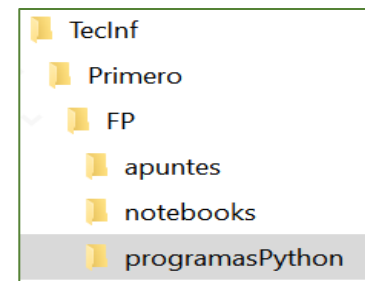
```
from math import sqrt  
def raíces(a:float,b:float,c:float)->(float,float):  
    raiz1=None  
    raiz2=None  
    discriminante=b**2-4*a*c  
    if a!=0 and discriminante>=0:  
        raiz1=(-b+sqrt(discriminante))/(2*a)  
        raiz2=(-b-sqrt(discriminante))/(2*a)  
    return (raiz1,raiz2)
```



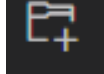
Ejercicio

Preparación del entorno de trabajo:


1. Seguir las indicaciones del documento “*FP-2024-25-TI3-Entorno de trabajo para trabajar con Python.v.1.0.0.pdf*” e instalar Python y Visual Studio Code (VSC)
2. Crear en la estructura de ficheros del ordenador una carpeta para los programas Python. *Por ejemplo*, si se ha creado una carpeta *TecInf* y dentro otra denominada *primero*, en la que se crearán carpetas para las distintas asignaturas, se puede crear una denominada *FP* y dentro una para los *apuntes*, otra para los *notebooks* y otra para los *programasPython*
3. Abrir VSC y elegir como carpeta para trabajar “*programasPython*”, mediante “File->Open Folder”



Construcción del programa:

1. Crear un proyecto “*T01_EcuaciónSegundoGrado*”, mediante el icono 
2. Crear los dos módulos Python copiando el código tal como está en la diapositiva anterior
 - a) *ecuación_segundo_grado.py*
 - b) *test_ecuación_segundo_grado.py*

Probando el programa

1. Seleccionar en el editor el fichero b) y ejecutar con 
2. Probar con 1,5 y 6: debe salir (-2.0, -3.0)
3. Probar con 1,-5 y 6: debe salir (3.0, 2.0)



Ejercicio

Cálculo de área de un triángulo conociendo sus tres lados.

La fórmula que hay que aplicar es la *fórmula de Herón*:

Siguiendo los pasos del ejercicio anterior:

1. Crear el proyecto *T02_ÁreaTriánguloPorHerón*
2. Crear los módulos Python:
 - a) *área_triángulo.py*
 - b) *test_área_triángulo.py*
3. Programar en cada módulo lo necesario para que el programa pida por teclado los tres lados de un triángulo y visualice el área usando la fórmula de Herón.

La función que calcule el área (que debe programarse en *área_triángulo.py*) debe recibir tres parámetros reales y devolver un solo valor real. Si alguno de los lados es negativo devolverá **None**.

