Programa - Instrucciones

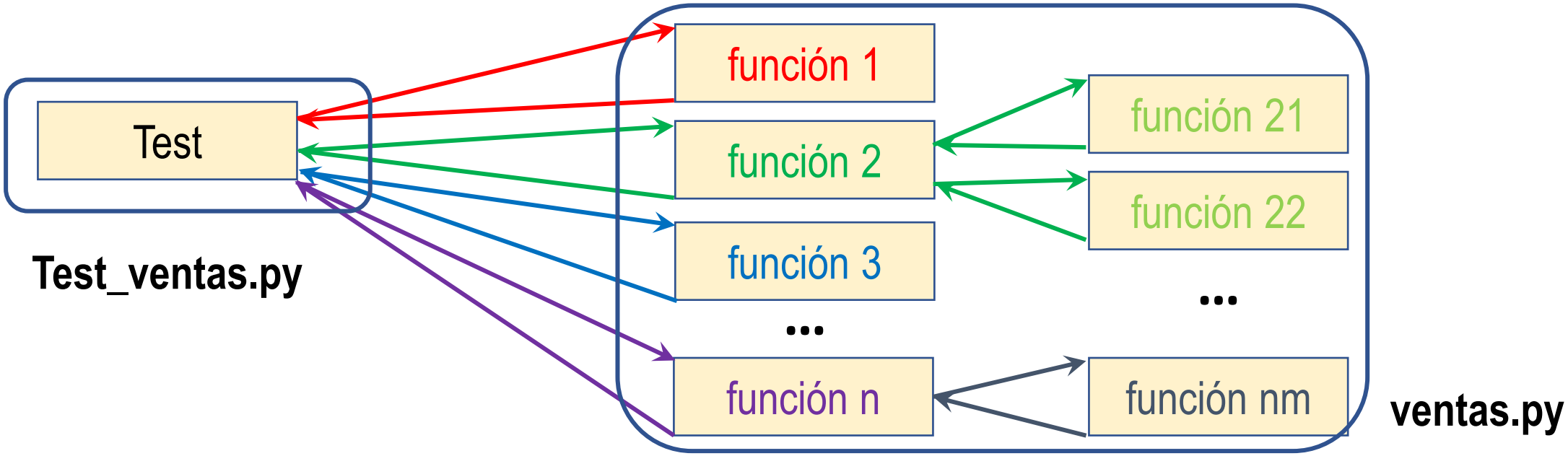
- Un programa está compuesto por instrucciones. También se llaman sentencias.
- Normalmente cada instrucción se escribe una debajo de otra
- Las instrucciones son expresiones con *variables, literales, operadores aritméticos, operadores de relación, llamadas a otras funciones* (predefinidas en las librerías de Python o construidas por el programador), formalmente bien construidas y con sentido.
- Cuando hay más de una sentencia se escriben una a continuación de otra
- Salvo que haya expresiones que modifiquen el flujo de ejecución de un programa, las instrucciones se ejecutan de forma secuencial una detrás de otra.

Ejemplo de un programa 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 evalua 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.



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*)

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 hay más de una palabra se separa por _
- 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=2022 -número_loteria=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/vacío*: *None* para inicializar una variable **sin valor** predeterminado.
  -resultado=*None* -fecha_defunción=*None*

- *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 →48 (*más adelante hay una diapositiva con la prelación de operadores*)

(3*4)**2 →144

numero%2 → (devuelve 0 si número contiene un valor par y 1 si el valor es impar)

7/2 →3.5 ; 7//2 →3 (sólo la parte entera) ; 7%2→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” →str
- ‘Hola Don Pepito’ + ‘ hola Don José’ →str
- 23 →int
- 38*4 →int
- (2+4)*6.0+(-3*4%2) →float
- 56/(2-4)*3 →float
- mi_tangente=sin(angulo)/cos(angulo) →float
- raiz1=sqrt(pow(b,2)-4*a*c) →float

Operadores Relacionales entre expresiones (devuelve *True* o *False*)

- 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

- *and*: conjunción lógica (y)→ devuelve *True* si todas las expresiones son ciertas
- *or*: disyunción lógica (o) →devuelve *True* si al menos una expresión es cierta
- *not*: negación→ 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 → *False*

9>2**3 *or* 5>6 → *True*

not (5<6) → *False*

not(9>2**3) *or* 5>6 → *False*

not ((exp1 *or* exp2) *and* (exp3 *or* exp4)) *and* exp5 *or* exp6 *or not*(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

Archivo-> "*test_ecuación_segundo_grado.py*"

```
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("La raíces son: ",raíces(c1,c2,c3))
```

Archivo-> "*ecuación_segundo_grado.py*"

```
Función que calcula las raíces de una ecuación de segundo grado.
Entrada:  -Recibe como parámetros tres valores numéricos
Salida:   -Una tupla con las raíces. Sino se puede calcular, la tupla es (None,None)

from math import sqrt
def raíces(a,b,c):
    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)
```

Función-> "*raíces*"