Unió Europea
Fons social europeu

Funciones

1. Que es una función
2. Definir una función
3. Retornar valores
4. Argumentos y parámetros
5. Argumentos por valor y referencia
6. Argumentos indeterminados
7. Funciones recursivas
8. Funciones integradas

Son fragmentos de código que se pueden ejecutar múltiples veces

Pueden recibir y devolver información para comunicarse con el proceso principal

Qué es una función ?

"Una **función es** una porción o bloque de código reutilizable que se encarga de realizar una determinada tarea."

```
def  nombre ( param1, param2 … paramN) :
        instrucciones
        instrucciones
```

```
#---------------------------- definicions
def saludar() :
   print("Hola! print de la función saludar()")


def entrar() :
   print("Pase con el pase")


def salir() :
   print("Adios y Gracias, seguimos en contacto")
#------------------------------------------------ Inici

saludar ()
```

```
Hola! print de la función saludar()
```

Ámbito de las variables

Una variable declarada en una función no existe en la función principal:

```
#---------------------------- definicions
def test():
    n = 10
#--------------------------- Inici ----------
test()
print(n)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-477af9773b64> in <module>
      3
      4 test()
----> 5 print(n)

NameError: name 'n' is not defined
```

La instrucción global

Para poder modificar una variable externa en la función, debemos indicar que es global de la siguiente forma:

```
#---------------------------------------- definicions
def test():
    global o  # variable que hace referencia
    o = 5     #  a la o externa
    print(o)

#---------------------------------------------------- Inici
o = 10
test()
print(o)
```

Sin embargo, una variable declarada fuera de la función (al mismo nivel), sí que es accesible desde la función:

```
#---------------------------- definicions
m = 10

def test():
    print(m)
#------------------------- Inici ------------
test()
```

Siempre que declaremos la variable antes de la ejecución, podemos acceder a ella desde dentro:

```
#---------------------------------------- definicions
def test():
    print(l)
#---------------------------------------------------- Inici
l = 10
test()
```

En el caso que declaremos de nuevo una variable en la función, se creará un copia de la misma que sólo funcionará dentro de la función.

Por tanto no podemos modificar una variable externa dentro de una función:

```
#---------------------------------------- definicions
def test():
    o = 5  # variable que sólo existe dentro de la función
    print(o)
#---------------------------------------------------- Inici
o = 10  # variable externa, no modificable
test()
print(o)
```

Retorno de valores

Para comunicarse con el exterior, las funciones pueden devolver valores al proceso principal gracias a la instrucción **return**.

En el momento de devolver un valor, la ejecución de la función finalizará:

```
#---------------------------------------- definicions
def test():
    return "Una cadena retornada"
#------------------------------------------------- Inici
test()
```

Los valores devueltos se tratan como valores literales directos del tipo de dato retornado:

```
print(test())
```

Por ejemplo no podemos sumar una cadena con un número:

```
c = test() + 10
```

También podemos devolver cualquier tipo de colección y manejarla directamente:

```
def test():
    return [1,2,3,4,5]
#---------------------------------------- Inici
print(test())
print(test()[-1])
print(test()[1:4])
```

Evidentemente es posible asignar el valor retornado a una variable:

```
lista = test()
print(lista[-1])
```

Tenemos una función que le suma un número a una constante

```
#------------------------------ definiciones
def sumak (numero, constante) :
    return (numero + constante)

#------------------------------ inicio
print (sumak (7, 3.14))
```

Vamos a modificarla para que si no informan la constante, asuma que es cero.

```
#------------------------------ definiciones
def sumak (numero, constante = 0) :
    return (numero + constante)

#------------------------------ inicio
print (sumak (7))
```

Estudiamos un caso concreto

```
#------------------------------ definiciones
def  calcular1 (a, b) :
       print ("resultado = ", a ** 2 + ( b * 2))

#------------------------------ inicio

num1 = int(input("Entre un número : "))
num2 = int(input("Entre un número : "))
calcular1( num1 , num2)
```

La función calcula e imprime. No devuelve el valor de la suma.

Si la llamamos así : `print (calcular1 (num1, num2))`

Que resultado obtendremos ?

```
#------------------------------ definiciones
def  calcular1 (a, b) :
       print ("resultado = ", a ** 2 + ( b * 2))

#------------------------------ inicio

num1 = int(input("Entre un número : "))
num2 = int(input("Entre un número : "))
calcular1( num1 , num2)
```

Esta segunda versión es más reutilizable:

```
#------------------------------ definiciones
def  calcular1 (a, b) :
       return ( a ** 2 + ( b * 2))

#------------------------------ inicio

num1 = int(input("Entre un número : "))
num2 = int(input("Entre un número : "))
print ("resultado = ", calcular1( num1 , num2))
```

Vamos a crear una función de pida un entero y se asegure que no falla, aunque introduzcan letras.

```
#------------------------------ definiciones

def  input_int (mensaje, valor_defecto = 0) :

    valor = input (mensaje)
    if valor.isdigit() :
        valor = int(valor)
    else :
        valor = valor_defecto

    return (valor)
#------------------------------ inicio

num = input_int("Entre un número : ")

print ("num = ", num)
```

Vamos a crear una función de pida un entero y se asegure que no falla, aunque introduzcan letras.

Práctica P01

```
#------------------------------ definiciones
def sumak (numero, constante) :
    return (numero + constante)

#------------------------------ inicio
print (sumak (7, 3.14))
```

1. Teclea y prueba esta función
2. Crea una función que llame sumalista_k ,
   - que recibe una colección de números y una constante
   - y devuelve la suma de (numero * k) para todos los números de la colección.
   - El valor por defecto para k es 1
3. Crea un programa que pida una constante y varios números hasta que pulsen “=” y calcule sumalista_k.

Introduzca una constante k :

Introduzca un número (= salir) :

Argumentos y parámetros

En la definición de una función los valores que se reciben se denominan **parámetros**, pero durante la llamada los valores que se envían se denominan **argumentos**.

Argumentos por posición

Cuando enviamos argumentos a una función, estos se reciben por orden en los parámetros definidos. Se dice por tanto que son argumentos por posición:

```
def resta(a, b):
    return a - b

resta(30, 10)  # argumento 30 => posición 0 => parámetro a
               # argumento 10 => posición 1 => parámetro b
```

Argumentos por nombre

Sin embargo es posible evadir el orden de los parámetros si indicamos durante la llamada que valor tiene cada parámetro a partir de su nombre:

```
resta(b=30, a=10)
```

Llamada sin argumentos

Al llamar una función que tiene definidos unos parámetros, si no pasamos los argumentos correctamente provocará un error:

Código Resultado

```
resta()
```

In [2]:
```
def resta(a,b):
    return a-b

resta(1,2)
```

Out[2]:
```
-1
```

In [3]:
```
resta(b=2,a=1)
```

Out[3]:
```
-1
```

In [4]:
```
resta()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-78c8f433960e> in <module>()
----> 1 resta()

TypeError: resta() missing 2 required positional arguments: 'a' and 'b'
```

Parámetros por defecto

Para solucionarlo podemos asignar unos valores por defecto nulos a los parámetros, de esa forma podríamos hacer una comprobación antes de ejecutar el código de la función:

```
def resta(a=None, b=None):
    if a == None or b == None:
        print("Error, debes enviar dos números a la función")
        return   # indicamos el final de la función aunque no devuelva nada
    return a-b

resta()
```

Práctica P02

Supongamos que vamos a crear un juego del dominó.
Programa las siguientes funciones :

```
funcion genera_fichas (fichas)
    # recibe una lista, la vacía, y la rellena con
    # todas las posibles fichas del dominó
    # en formato :    "número-número"
    # ejemplo ["1-1","1-2","1-3"]
```

```
funcion imprime_tablero (fichas)
   # recibe una lista de fichas y las imprime
```

```
funcion revuelve_fichas (fichas)
    # recibe una lista de fichas y la devuelve en sí misma,
    # revuelta al azar  (shuffle)
```

Prueba este fragmento de código y mira como funciona

```
import random

list = ["1-1", "1-2", "1-3", "1-4", "1-5"]
random.shuffle(list)
print (list)
```

```
funcion extrae_fichas (fichas, n ) devuelve sub_lista
    # desapila n fichas y las devuelve en una sub_lista
```

Paso por valor y referencia

Dependiendo del tipo de dato que enviemos a la función, podemos diferenciar dos comportamientos:

- **Paso por valor**: Se crea una copia local de la variable dentro de la función.
- **Paso por referencia**: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente:

- **Los tipos simples se pasan por valor**: Enteros, flotantes, cadenas, lógicos...
- **Los tipos compuestos se pasan por referencia**: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor

Como ya sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

```
def doblar_valor(numero):
    numero *= 2

n = 10
doblar_valor(n)
print(n)
```

Ejemplo de paso por referencia

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

Ejemplo de paso por referencia

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

```
def doblar_valores(numeros):
    for i,n in enumerate(numeros):
        numeros[i] *= 2

ns = [10,50,100]
doblar_valores(ns)
print(ns)
```

Para modificar los tipos simples podemos devolverlos modificados y reasignarlos:

```
def doblar_valor(numero):
    return numero * 2

n = 10
n = doblar_valor(n)
print(n)
```

Y en el caso de los tipos compuestos, podemos evitar la modificación enviando una copia:

```
def doblar_valores(numeros):
    for i,n in enumerate(numeros):
        numeros[i] *= 2

ns = [10,50,100]
doblar_valores(ns[:])  # Una copia al vuelo de una lista con [:]
print(ns)
```

Práctica P03

Estudia este ejemplo :

Creamos un programa para generar datos ficticios mensuales de varias ciudades, en formato csv separado por ;.

1. Pedimos el número de ciudades a crear y los nombres de esas ciudades.
2. Luego generamos los datos aleatorios para cada ciudad y para cada mes.
3. Finalmente se imprimen los datos generados.

```
Quantes ciutats (0=sortir): 3
Ciutat : giro
Ciutat : tarr
Ciutat : llei
---------------- DADES ------------
Ciutat; gen; feb; març; abr; maig; juny; jul; ago; set; oct; nov; des;
giro;105;39;111;377;178;248;265;489;381;181;126;125;
tarr;260;458;165;300;81;194;154;18;253;52;74;305;
llei;440;292;384;80;92;346;494;393;220;431;149;456;
```

```
from random import randrange

#------------------------------------------ FUNCIONES
def demanar_dades (ciutats) :
    num = int(input ("Quantes ciutats (0=sortir): "))

    for i in range(num):
        ciutats.append(input("Ciutat : "))

def generar_dades_ciutat (ciutat) :

    cadena = ciutat + ";"
    for i in range (1, 13) :
       cadena = cadena + str(randrange(10, 500)) + ";"
    return cadena

def imprimir (dades) :
    print ("---------------- DADES ------------")
    print ("Ciutat; gen; feb; març; abr; maig; juny; jul; ago; set; oct; nov; des;")
    for item in dades :
       print (item)
#------------------------------------------ Proceso principal
ciutats = []
dades = []

demanar_dades(ciutats)

for item in ciutats :
    tmp = generar_dades_ciutat(item)
    dades.append(tmp)

imprimir(dades)
```

```
#---------------------------- Procés principal
ciutats = []
dades = []

demanar_dades(ciutats)

for item in ciutats :
    tmp = generar_dades_ciutat(item)
    dades.append(tmp)

imprimir(dades)
```

```python
from random import randrange

#---------------------------------------- FUNCIONS
def demanar_dades (ciutats) :
    num = int(input ("Quantes ciutats (0=sortir): "))

    for i in range(num):
        ciutats.append(input("Ciutat : "))

    return ciutats
```

```python
#---------------------------- Procés principal
ciutats = []
dades = []

demanar_dades(ciutats)

for item in ciutats :
    tmp = generar_dades_ciutat(item)
    dades.append(tmp)

imprimir(dades)
```

```
def generar_dades_ciutat (ciutat) :

    cadena = ciutat + ";"
    for i in range (1, 13) :
        cadena = cadena + str(randrange(10, 500)) + ";"
    return cadena

def imprimir (dades) :
    print ("---------------- DADES ------------")
    print ("Ciutat; gen; feb; març; abr; maig; juny; jul;
    for item in dades :
        print (item)
```

```
#--------------------------- Procés principal
ciutats = []
dades = []

demanar_dades(ciutats)

for item in ciutats :
    tmp = generar_dades_ciutat(item)
    dades.append(tmp)

imprimir(dades)
```

Perfilando una función

- Buscar la forma más reutilizable
- Escoger un nombre de función representativo
- Decidir cuantos parámetros de entrada se necesitan y qué valor se devuelve en la salida

» Vamos a ver un ejemplo

Aunque las funciones en Python pueden acceder a cualquier variable del programa declaradas como variables globales o no locales, se necesita saber el nombre de las variables, como muestra el ejemplo siguiente:

```
def escribe_media() :
    media = (a + b) / 2
    print("La media de {} y {} es : {}".format(a,b,media))
    return

a = 3
b = 5
escribe_media()
print("Programa terminado")
```

```
La media de 3 y 5 es : 4.0
Programa terminado
```

El problema de una función de este tipo es que es muy difícil de reutilizar en otros programas o incluso en el mismo programa, ya que sólo es capaz de hacer la media de las variables "a" y "b".

Si en el programa no se utilizan esos nombres de variables, la función no funcionaría.

```
escribe_media()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-72157f484852> in <module>
----> 1 escribe_media()

<ipython-input-7-d8a62b08ae4a> in escribe_media()
      1 def escribe_media() :
----> 2     media = (a + b) / 2
      3     print("La media de {} y {} es : {}".format(a,b,media))
      4     return
      5

NameError: name 'a' is not defined
```

Para evitar ese problema, las funciones admiten argumentos, es decir, permiten que se les envíen valores con los que trabajar.

De esa manera, las funciones se pueden reutilizar más fácilmente, como muestra el ejemplo siguiente:

```
def escribe_media (a, b) :
    media = (a + b) / 2
    print("La media de {} y {} es : {}".format(a,b,media))
    return


escribe_media(3, 5)
print("Programa terminado")
```

```
La media de 3 y 5 es : 4.0
Programa terminado
```

```python
def calcula_media (x, y) :
    media = (x + y) / 2
    return (media)

#-----------------------------------------------
# PROCESO PRINCIPAL
#-----------------------------------------------

a = int (input ("Primer número : "))
b = int (input ("Segundo número : "))
media = calcula_media(a, b)
print("La media de {} y {} es : {}".format(a,b,media))
```

```
Primer número : 39
Segundo número : 44
La media de 39 y 44 es : 41.5
```

Pero esta función tiene todavía un inconveniente y es que sólo calcula la media de dos valores. Sería más interesante que la función calculara la media de cualquier cantidad de valores.

Práctica P04

Ejercicio 1

Realiza una función llamada **area_rectangulo(base, altura)** que devuelva el área del rectangulo a partir de una base y una altura. Calcula el área de un rectángulo de 15 de base y 10 de altura:

> **Recordatorio**
>
> El área de un rectángulo se obtiene al multiplicar la base por la altura.

Ejercicio 2

Realiza una función llamada **area_circulo(radio)** que devuelva el área de un círculo a partir de un radio. Calcula el área de un círculo de 5 de radio:

Recordatorio

El área de un círculo se obtiene al elevar el radio a dos y multiplicando el resultado por el número pi. Puedes utilizar el valor 3.14159 como pi o importarlo del módulo math:

Código Resultado

```
import math

print(math.pi)
```

Ejercicio 3

Realiza una función llamada **relacion(a, b)** que a partir de dos números cumpla lo siguiente:

- Si el primer número es mayor que el segundo, debe devolver 1.
- Si el primer número es menor que el segundo, debe devolver -1.
- Si ambos números son iguales, debe devolver un 0.

Comprueba la relación entre los números: '5 y 10', '10 y 5' y '5 y 5'.

Ejercicio 4

Realiza una función llamada **intermedio(a, b)** que a partir de dos números, devuelva su punto intermedio. Cuando lo tengas comprueba el punto intermedio entre -12 y 24:

> **Recordatorio**
>
> El número intermedio de dos números corresponde a la suma de los dos números dividida entre 2

Ejercicio 5. Cálculo de tiempos

Escribir dos funciones que permitan calcular:

a) La cantidad de segundos en un tiempo dado en horas, minutos y segundos. **calcula_segundos ("hh:mm:ss") devuelve segundos**

b) La cantidad de horas, minutos y segundos de un tiempo dado en segundos **calcula_hhmmss (segundos) devuelve "hh:mm:ss"**

Después realiza un programa con un menú de opciones

1. Calcular cantidad de segundos
2. Calcular horas, minutos y segundos

y pide los datos adecuados a la opción elegida y muestra el resultado.

Ejercicio 6. Avanzado. Valida DNI.

Realiza dos funciones relativas a la validación del dni que sirvan para NIFs españoles y extranjeros. Usa la web del ministerio para obtener el método de cálculo (*).

- valida_dni (dni) : Boolean
  - Recibe una cadena con un dni (nacional o extranjero) y devuelve True si es vàlido y False si no lo és.
- calcula_letra_dni (dni) : Boolean
  - Recibe una cadena con un dni sin letra final (nacional o extranjero) y devuelve la letra que le corresponde.

Desarrolla un programa para poder probar ambas funciones

(*)

http://www.interior.gob.es/web/servicios-al-ciudadano/dni/calculo-del-digito-de-control-del-nif-nie

Ejercicio 7. Avanzado. Entrada de números

Desarrolla dos funciones para pedir datos numéricos, con control de errores y valores por defecto (opcionales).

- input_int (cadena_texto, valor_por_defecto) : int
  - Realiza una entrada de datos (input), convierte la entrada a ínteger, y si se produce un error devuelve el valor por defecto (si se ha informado) o cero si no se ha informado
- input_float (cadena_texto, valor_por_defecto) : float
  - Realiza una entrada de datos (input), convierte la entrada a float, y si se produce un error devuelve el valor por defecto (si se ha informado) o 0.0 si no se ha informado.

Desarrolla un programa para poder probar ambas funciones

```
# SOLUCION PRACTICA P01

#------------------------------ definiciones
def sumalista_k (nums, k = 1) :
    suma = 0
    for n in nums :
        suma += n * k
    return (suma)

#------------------------------ inicio
print (sumalista_k ((7,6,4,3), 2))
```

Solucion Practica P02

```
def genera_fichas(fichas) :
   fichas.clear()
   for i in range (0, 7) :
      for j in range (0, 7) :
            fichas.append( str(i)+"-"+str(j))


def imprime_tablero (fichas) :
#recibe una lista de fichas y las imprime
   print(fichas)


todas_fichas = []
genera_fichas(todas_fichas)
imprime_tablero(todas_fichas)
```