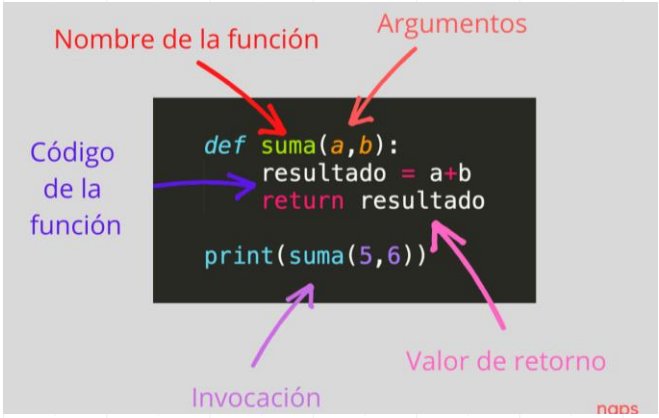
Funciones - Python

Ing. Javier Calli Olvea

¿Qué es una Función?

Una función en Python (y en cualquier otro lenguaje de programación) es un bloque de líneas de código o un conjunto de instrucciones cuya finalidad es realizar una tarea específica. Puede reutilizarse a voluntad para repetir dicha tarea. Las funciones nos ayudan a que el código sea más fácil de leer y entender.

Estructura de una Función



Palabras reservadas

Las palabras reservadas corresponden a órdenes especiales que tiene su propio sentido en Python. Algunas son funciones, otras bucles, otras objetos de tipo booleano.

Las siguientes palabras no pueden usarse para definir otros objetos.

Palabras reservadas

and	elif	if	raise
as	else	import	return
assert	except	in	True
async	e	is	try
await	xec	lambda	while
break	False	Not	with
class	finally	None	yield
continue	for	or	
def	From	pass	
del	global	print	

5

Palabras reservadas

Una función es un bloque de instrucciones agrupadas, que permiten reutilizar partes de un programa . La biblioteca estándar de Python incluye las siguientes funciones de forma predeterminada (es decir, estas funciones siempre están disponibles)

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

6

Motivación

Una primera aplicación de estas estructuras condicionales es la construcción de funciones definidas a trozos. Por ejemplo, l a función signo se define como:

$$\text{signo}\, x = \begin{cases} 1 & \text{si } x > 0, \\ -1 & \text{si } x < 0, \\ 0 & \text{si } x = 0. \end{cases}$$

Funciones con y sin return

```
def repite_conretorno(caracter='-', repite=3):
    return caracter * repite

repite_conretorno()
'---'

x = repite_conretorno()

x
'---'
```

```
def repite_sinretorno(caracter='-', repite=3):
    print(caracter * repite)

repite_sinretorno()


y = repite_sinretorno()


y
```

Cuando tenemos un return, podemos asignar el valor resultante a un nuevo objeto.

Si no tenemos un return, el objeto al que asignamos el valor resultante no guarda ningún valor.

Ejemplo: Función

```
def suma(num1, num2):
        return num1 + num2


# ejecutar
suma(6, 3)
```

```
>> 9.0
```


Ejemplo: Funciones con un número fijo de argumentos

```
def division(num1, num2):
        return num1/num2

# ejecutar
division(num1=12, num2=3)

# ejecutar
division(num2=2, num1=10)
```

```
>> 4.0


>> 5.0
```

Ejemplo: Funciones con un número fijo de argumentos

```
def área_triangulo(base, altura):
        return (base*altura)/2

# ejecutar
área_triangulo(6, 4)

# ejecutar
área_triangulo(6)
```

```
>> 12.0

>> error
```

```
File "<ipython-input-3-4f014e311e94>", line 1, in <module>
  area_triangulo(6)

TypeError: area_triangulo() missing 1 required positional argument:
'altura'
```


Ejemplo: Función

$$\text{signo}\, x = \begin{cases} 1 & \text{si } x > 0, \\ -1 & \text{si } x < 0, \\ 0 & \text{si } x = 0. \end{cases}$$

```
def signo(x):
        if x < 0:
                return -1
        elif x > 0:
                return 1
        else:
                return 0
```

```
Signo(-2.4)

>> -1
```

Ejemplo: Funciones con un número variable

```
def distancia(*tramos):
    ''' Suma distancia de tramos '''
    total = 0
    for distancia in tramos:
        total = total + distancia
    return total
```

El asterisco indica que cada vez podemos especificar un número distinto de valores.

```
distancia(2,3)

distancia(2,3,4)

distancia(2,3,4,5)
```

```
>> 5

>> 9

>> 14
```


Ejemplo: Funciones con parámetros por defecto

```
def pagar(importe, descuento = 5):
    return importe - (importe * descuento/100)

pagar(1000)

pagar(1000,10)
```

```
>> 950.00

>> 900.00
```

Ejemplo: Funciones con parámetros por defecto

```
def repite_caracter(caracter="-", repite=3):
        return caracter * repite

repite_carácter()

repite_caracter('+',20)

repite_caracter(repite=10, carácter='*')
```

```
>> ---

>> ++++++++++++++++++++

>> **********
```

15

Ejemplo: Funciones que devuelven más de un valor

```
def distancia2(*tramos):
    ''' Suma distancia de tramos '''
    total = 0
    for distancia in tramos:
        total = total + distancia
    media = total / len(tramos)
    return total, media

distancia2(2,3,4)
```

```
>> (9, 3.0)
```

16

Ejemplo: Funciones recursivas

```
def cuenta_regresiva(numero):
    numero -= 1
    if numero > 0:
        print(numero)
        cuenta_regresiva(numero)
    else:
        print(numero)

cuenta_regresiva(10)
```

```
9
8
7
6
5
4
3
2
1
```


Ejemplo: Funciones recursivas - tarea

```
def factorial(numero):
    print("Valor inicial ->", numero)
    if numero > 1:
        numero = numero * factorial(numero -1)
    print("valor final ->",numero)
    return numero
```

```
In [2]: factorial(5)
Valor inicial -> 5
Valor inicial -> 4
Valor inicial -> 3
Valor inicial -> 2
Valor inicial -> 1
valor final -> 1
valor final -> 2
```

Ejemplo: Funciones recursivas - tarea

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

fib(5)
```

```
>> 5
```

19

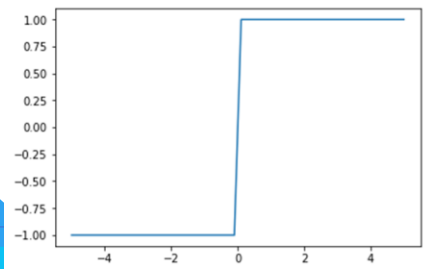
```
In [1]: import numpy as np
   ...: import matplotlib.pyplot as plt

In [2]: def signo(x):
   ...:     if x<0:
   ...:         return -1
   ...:     elif x>0:
   ...:         return 1
   ...:     else:
   ...:         return 0

In [3]: signo = np.vectorize(signo)

In [4]: x = np.linspace(-5,5)

In [5]: plt.plot(x,signo(x))
Out[5]: [<matplotlib.lines.Line2D at 0x1f7f3cf38d0>]
```



20

Bibliografia.

https://uniwebsidad.com/libros/python

Javier Calli Olvea. javier422@gmail.com

Gracias!