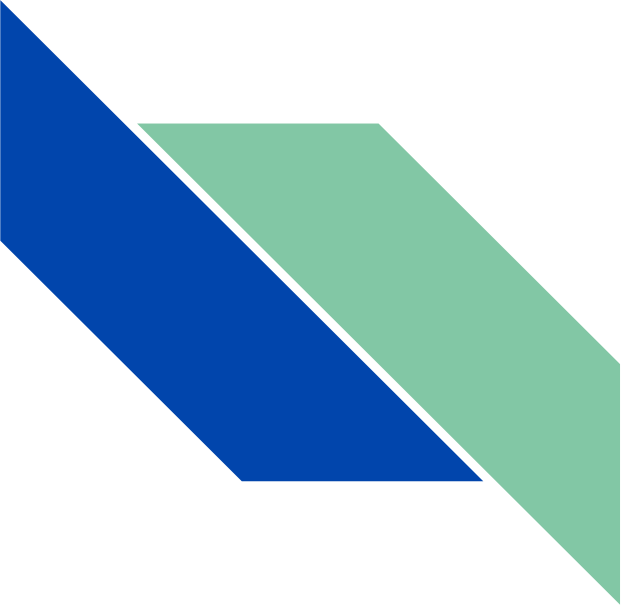

Curso Intensivo de Python - Tema 6

Agustín Arenas

Curso Intensivo de Python - Tema 6

1) Funciones
2) Variables locales
3) Argumentos
4) Módulos

Funciones

- Una función es un grupo de sentencias que existen con un programa para llevar a cabo una tarea específica

- El porqué de usar funciones se basa en **dividir y conquistar**. Esto es el principio de Single responsability

- “Una función o método debe hacer una sola cosa”

Funciones

- Las funciones pueden venir de dos sabores: **void functions y value-returning functions**
- Las del primer tipo, simplemente ejecutan las sentencias que contiene y termina
- Las del segundo tipo, retorna un valor cuando termina de llamarse.

Funciones

- El código para una función se conoce como **definición de la función**
- La ejecución de la misma se realiza cuando se **llama a la función**
- El nombre de la función sigue la misma convención que las variables

Funciones

```
def funcionTipoUno(): #¡Ojo! Aqui van dos puntos y hay que indentar
    #Diremos Tipo 1 a la función que no recive ni devuelve nada
    print('Hola, no recibo ni devuelvo nada')

def funcionTipoDos(mensaje):
    #Diremos Tipo 2 a la función que recive N args ni devuelve nada
    print('Hola, yo recibo {0}'.format(mensaje.upper()))

def funcionTipoTres():
    #Diremos Tipo 3 a la función que no recive nada pero devuelve algo
    print('Voy a devolver algo...')
    return 'algo'

def funcionTipoCuatro(mensaje):
    #Diremos Tipo 4 a la función que recive N args y devuelve algo
    print('Hola, yo recibo {0}'.format(mensaje.upper()))
    return 'algo'.upper()

#Llamadas a las funciones
mensaje = 'Mi MeNsAjE'
funcionTipoUno()
funcionTipoDos(mensaje)
mensaje = funcionTipoTres()
mensaje = funcionTipoCuatro(mensaje)
print(mensaje)
```

Variable local

- Las variables en las funciones tienen una tratativa especial, se denominan **variables locales**. Estas variables son creadas **dentro** de la función

- Y **NO** pueden ser accedidas desde afuera de la función. Diferentes funciones pueden tener los mismos nombres de variables locales, dado que no pueden verse entre sí

Variable local

```
def main():
    obtenerNombre()
    print('Hola {0}'.format(nombre)) #¡Error!

def obtenerNombre():
    nombre = input('¿Como te llamas?')

main()
```

Argumentos

- Algunas veces resulta útil no solo llamar a una función, sino enviar información, como vimos en la función Tipo 2 y Tipo 4

- Un argumento es cualquier pieza de dato que se pasa dentro a una función cuando se la llama. En cambio, un parámetro es una variable que recibe ese argumento pasado

- Se puede pasar más de un argumento a una función. Los argumentos son pasados por asignación, es decir, el parámetro pasado es una referencia a un objeto, pero la referencia es pasada por valor (copia)

Argumentos

- Hay que tener en cuenta que algunos tipos de datos son mutables, pero otros no

- Por lo que si pasamos un objeto mutable a un método, este consigue una referencia al mismo objeto y puede cambiarse. En cambio, si pasamos un objeto inmutable, no podemos modificar la referencia

Argumentos

```
def tratarModificarLista(lista):
    print('Obtuve', lista)
    lista.append(4)
    print('Ahora es', lista)
listaOriginal = list(range(3))
print('Previo a llamarse', listaOriginal)
tratarModificarLista(listaOriginal)
print('Luego de modificarse', listaOriginal)
def usamosParametro(lista):
    print('Obtuve', lista)
    lista = list(range(5))
    print('Ahora es', lista)
    print('Ahora es', listaOriginal)
listaOriginal = list(range(3))
print('Previo a llamarse', listaOriginal)
usamosParametro(listaOriginal)
print('Luego de modificarse', listaOriginal)
```

Módulos

- Las funciones pueden definirse en un módulo, que es simplemente un archivo Python que contiene el código

Módulo (circulo.py)

```
import math as m


def area(radio):
    return m.pi * radio**2


def circunferencia(radio):
    return 2*m.pi*radio
```

Módulo (main.py)

```
import circulo as c


def main():
    for r in range(1, 5):
        print('Circulo de radio {0}, tiene area
{1:.2f} y circunferencia {2:.2f}'.format(
            r,
            c.area(r),
            c.circunferencia(r)
        ))


main()
```

Lambda

- Una función lambda se usa cuando necesitas una función sencilla y de rápido acceso
- La sintaxis de una función lambda es **lambda args: expresión**
- **NO** puedes darle un nombre a una función lambda, ya que estas son anónimas por definición
- Una función lambda puede tener tantos argumento como necesites, pero debe tener una sola expresión

Lambda

```python
def doble(x):
  return x * 2

mi_lista = [1, 2, 3, 4, 5, 6]
mi_lista_2 = [18, -3, 5, 0, -1, 12]

lista_nueva = list(map(doble, mi_lista))

# Una función lambda se define donde se usa
lista_nueva_lambda = list(map(lambda x: x * 2, mi_lista))
lista_nueva_2 = list(filter(lambda x: x > 0, mi_lista))

print(lista_nueva) # [2, 4, 6, 8, 10, 12]
print(lista_nueva_lambda) # [2, 4, 6, 8, 10, 12]
print(lista_nueva_2) # [18, 5, 12]
```

Lambda

```
"""
def multiplicar_por(x):
  def temp (n):
    return x*n
  return temp
"""

def multiplicar_por (n):
  return lambda x: x * n

duplicar = multiplicar_por(2)
triplicar = multiplicar_por(3)
diez_veces = multiplicar_por(10)

print(duplicar(6)) # 12
print(triplicar(5)) # 15
print(diez_veces(12)) # 120
```

¿Preguntas?