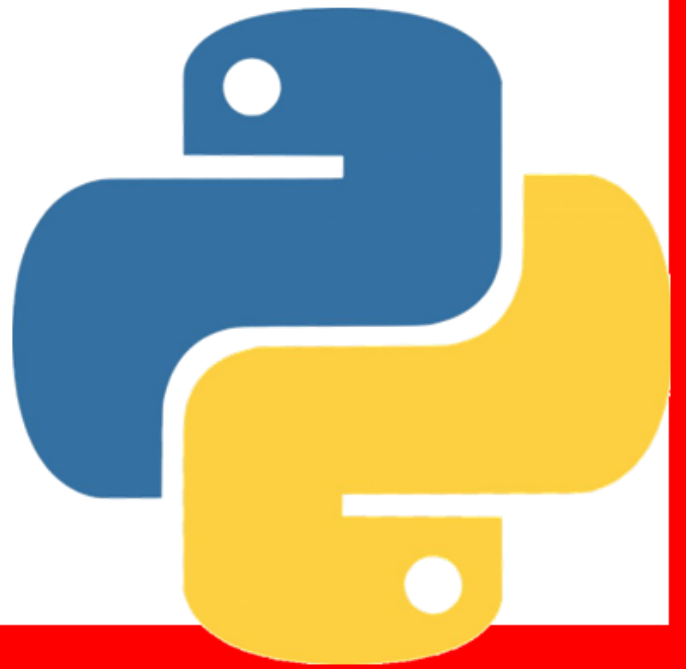
Funciones 2

Recursividad, funciones labda y …

11

Curso Básico

python



Funciones 2 (contenido)

1. Valores de argumentos por omisión (por default)
2. Argumentos de palabra clave (keyword arguments)
3. Funciones con devoluciones múltiples usando tuplas
4. Funciones con número arbitrario de parámetros, usando tuplas *args
5. Funciones con número arbitrario de parámetros, usando diccionarios **kwargs
6. Recursividad
7. Funciones lambda

Valores de argumentos por omisión (por default)

- Algunos argumentos pueden tener un valor por omisión si el usuario los omite al llamar la función.
- Una función puede tener varios argumentos con valores por default, pero una vez que se coloca un algumento pre-determinado, todos los argumentos a la derecha deben ser predeterminados.

```
def saludo(nombre, mensaje = ' buen día'):
    print("Hola ", nombre, mensaje)

def main():
    saludo('Juan')
    saludo('Luis', ' ya despierta')

if __name__ == "__main__":
    main()
```

Argumentos de palabra clave (keyword arguments)

```python
from math import pi

def volCilindro(radio, altura):
    volumen = pi * radio**2 * altura
    print(volumen)

def main():
    #Supongase que el radio=5 y altura=3
    volCilindro(5,3)
    volCilindro(radio=5,altura=3)
    volCilindro(altura=3,radios=5)

    #La llamada que sigue cambia radio por altura
    volCilindro(3,5) #Da un resultado distinto

    |
if __name__ == "__main__":
    main()
```

- Los argumentos en una función pueden ser llamados en **cualquier orden**, siempre que se use el nombre del parámetro y su valor asignado.

```
235.61944901923448
235.61944901923448
235.61944901923448
141.37166941115407
```

Funciones con devoluciones múltiples usando tuplas

```
def getMinMaxMean(datos):
    mi=ma=datos[0]
    su=0
    for a in datos:
        su+=a
        if a<mi:
            mi=a
        if a>ma:
            ma=a
    pro = su/len(datos)
    return (mi,ma,pro)

def main():
    datos=[34,67,89,34,56,78,45,23,67,56]
    men, may, med = getMinMaxMean(datos)
    print("El menor es: ",men)
    print("El mayor es: ",may)
    print("El promedio es: ",med)

if __name__ == "__main__":
    main()
```

```
El menor es:  23
El mayor es:  89
El promedio es:  54.9
```

Funciones con número arbitrario de parámetros, usando tuplas *args

- Cuando no conocemos de antemano el número de argumentos, podeos usar un asterisco antes del nombre del parámetro para indicar que contiene un numero arbitrario de valores.
- Posicionales
- No keywords

```
def adder(*num):
    sum = 0

    for n in num:
        sum = sum + n

    print("Sum:",sum)

adder(3,5)
adder(4,5,6,7)
adder(1,2,3,5,6)
```

```
def saludo1(nombre):
    print("Hola ", nombre)

def saludo2(*nombres):
    for name in nombres:
        print("Hola ", name)

def main():
    saludo1('Efren')
    saludo2('Efren', 'Ana', 'Edgar')


if __name__ == "__main__":
    main()
```

Turbo Código
https://www.youtube.com/user/juarezefren

Funciones con número arbitrario de parámetros, usando diccionarios **kwargs

- Se usan dos asteriscos antes de la variable ****var**, para pasar un número arbitrario de argumentos (de palabra clave, o argumentos nombrados) a una función. Dentro de la función, var es tratado como diccionario.

```
def intro(**data):
    print("\nTipo de dato del argumento:",type(data))
    for key, value in data.items():
        print("Su ", key, " es",value)

intro(Nombre="Marina", Apellido="Medina", Edad=22, Teléfono=1234567890)
intro(Nombre="Luis", Apellido="Lopez", Email="luislop@nomail.com",
      País="Wakanda", Edad=25, Teléfono=9876543210)
```

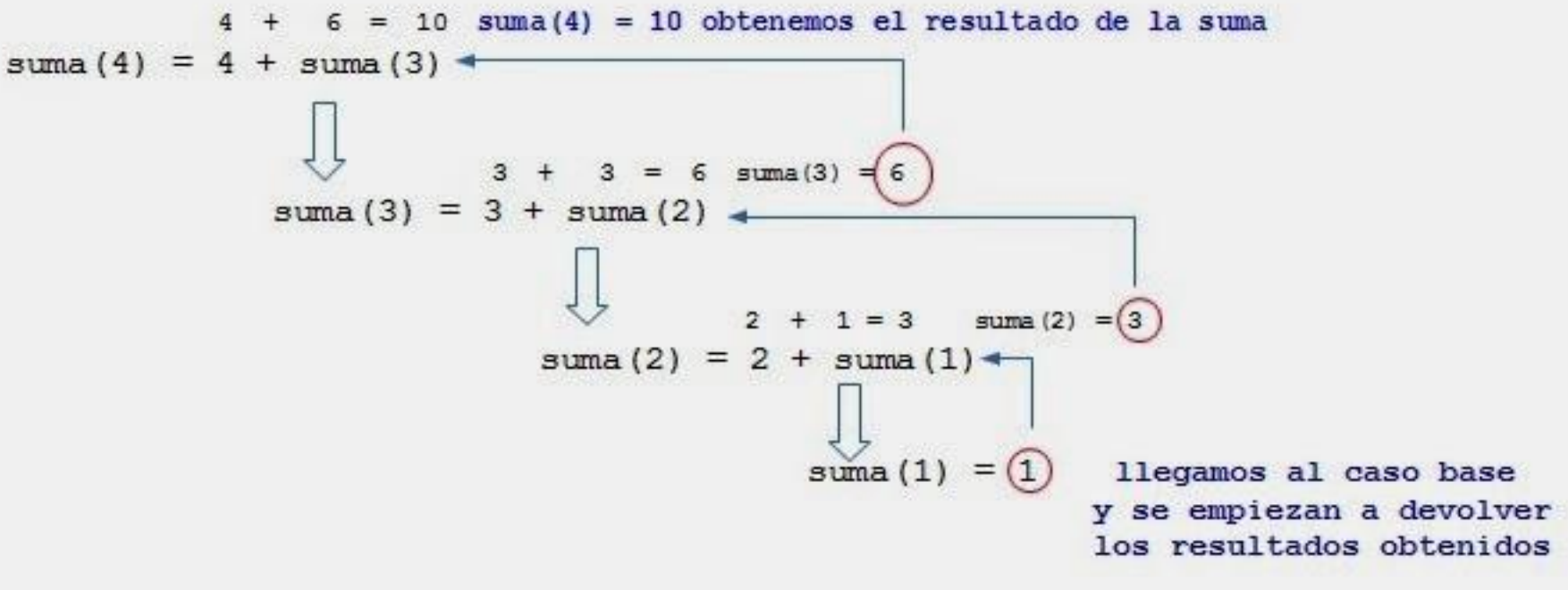
Recursividad

- Una función recursiva es aquella que se invoca a si misma dentro de su propio cuerpo de la función
- Lo anterior hace que se produzca un ciclo o bucle (como el bucle while o for), por tanto debe haber una forma de salir del bucle.
- Toda función recursiva debe contemplar al menos dos casos:
  - Caso recursivo: La función se llama así misma
  - Caso base: En esta situación la función ya no se llama a si misma, este caso detiene el ciclo, sin este caso el ciclo sería infinito.

Recursividad

```
def suma(n):
    if n<=1:
        return 1 #caso base
    else:
        return n + suma(n-1) #Caso recursivo


print( suma(4) )
```



Turbo Código
https://www.youtube.com/user/juarezefren

Recursividad

- Factorial de 5 es igual a:
- 5! = 5x4x3x2x1 = 5 x (4x3x2x1) = 5 x 4!
- 4! = 4x3x2x1 = 4 x 3!
- 3! = 3 x 2!

$$
\begin{aligned}
3! &= 3 \cdot (3-1)! \\
&= 3 \cdot 2! \\
&= 3 \cdot 2 \cdot (2-1)! \\
&= 3 \cdot 2 \cdot 1! \\
&= 3 \cdot 2 \cdot 1 \cdot (1-1)! \\
&= 3 \cdot 2 \cdot 1 \cdot 0! \\
&= 3 \cdot 2 \cdot 1 \cdot 1 \\
&= 6
\end{aligned}
$$

No recursiva

```
def factorial(n):
    f=1
    for i in range(1,n+1):
        f=f*i
    return f
```

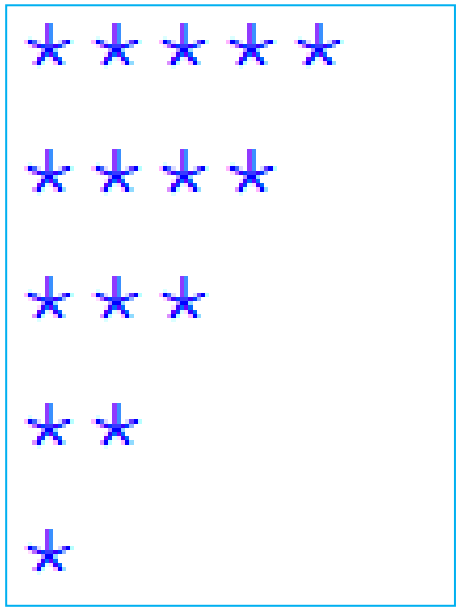
```
def factorial(n):
    if n > 1:
        return n * factorial (n-1) #recursivo
    else:
        return n #base

def main():
    print( factorial(4) )

if __name__ == "__main__":
    main()
```

Turbo Código
https://www.youtube.com/user/juarezefren

Recursividad (sin return)




```
def imprimePir(n):
    if n<=1:
        print('*')
    else:
        print('*' * n)
        imprimePir(n-1)

def main():
    imprimePir(5)

if __name__ == "__main__":
    main()
```

Caso base

Caso recursivo

Funciones lambda

- Pequeñas funciones anónimas pueden ser creadas con la palabra reservada **lambda.**

Syntax

```
lambda arguments : expression
```

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b : a * b
print(x(5, 6))
```

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Porque usar funciones lambda

- El poder de las funciones lambda se muestra cuando las usamos como una función anónima adentro de otra función

```
def myfunc(n):
  return lambda a : a * n
```

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

Porque usar funciones lambda

- El poder de las funciones lambda se muestra cuando las usamos como una función anónima adentro de otra función

Example

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

Funciones 2

Recursividad, funciones labda y …

11

Curso Básico

python