# Funciones

## Llamada a funciones

Una función es una **secuencia de sentencias que realizan una operación y que reciben un nombre.**cuando definimos una fucnción es como si la guardaramos como una variable petro en el interior lleva código.

- llamar a un afunción en el código se llama **invocar una función**.

```
def funcion():
    print("Hola Mundo")
    print("Segundo esta aqui para quedarse")

funcion()
```

```
type(32)
```

## Funciones internas

- El nombre de la función es type. La expresión **entre paréntesis recibe el nombre de argumento de la función.**
- El argumento es un valor o variable que se **pasa a la función como parámetro de entrada.**
- El resultado de la función type es el tipo del argumento.
- Es habitual decir que una función "toma" (o recibe) un argumento y "retorna" (o devuelve) un resultado. **El resultado se llama valor de retorno.** Argumento Esta en la definicion de la funcion. El parametros es el valor real que le pasamos a la función.

```
grande = max("Hola mundo")
print(grande)
min1 = min("Hola Mundo")
print(min1)
```

```
max("Hello World")
#print(max("Hello World!"))
max(1, 3)
type(max(1, 2))
```

## Funciones de conversión de tipos

Python también proporciona funciones internas que convierten valores de un tipo a otro. La función int toma cualquier valor y lo convierte en un entero, si puede, o se queja si no puede

## Tipos de conversiones :

1. **int** puede convertir valores en punto flotante a enteros, pero no los redondea; simplemente corta y descarta la parte decimal:

```
3.9999
ent = int(3.9999)
print(ent)
```

2. **float** convierte enteros y cadenas en números de punto flotante:

```
print(3)
print(type(3))
n1 = float(3)
n2 = float('2.344')
print('2.344')
print(type('2.344'))
print(f"{n1}", type(n1),{n2}, type(n2))
```

3. Finalmente, str convierte su argumento en una cadena

```
print(f"32 -->", type(32), "'32' -->", type(str(32)))
```

## Añadiendo funciones nuevas

**def (Parametros, Parametros,Parametros):**

- Solo es la definición de la función no ejecuta nada.
- Solo se ejeculata cuando la invocamos. ejemplo de que no se ejecuta **def**

```
x = 5
def estribillo():
    print("Hola Mundo!")
    print(":-) Siempre la misma frase.")
#no se ejecuta la función
x = x + 2
print(x)
```

```
x = 5
def estribillo():
    print("Hola Mundo!")
    print(":-) Siempre la misma frase.")
#no se ejecuta la función
x = x + 2
print(x)
estribillo()
```

### Parámetros y Argumentos

#### Parametros

- **Los parámetros se usan al crear la funcion** es como le llamamos
- los parametros son como variables que no existen son como alias.

```
def <nombre_funcion>(variable1, variable2 <--(Estos son parámetros)):
    cuerpo = sentencias
    return valor o variable etc
```

- Se deja un linea en blanco cuando se usa linea de comandos
- En linea de comandos hay que dejar une linea en blanco.

#### Argumentos

- **Los argumentos** son los valores reales que le damos a la funcion.

cuando invocamos le pasamos argumentos

```
def funcion1(variable1, variable2):# aqui son PARÁMETROS
    print(variable1)
    print(variable2)

x = 2
x = x + 4
y = 5
print(x)
funcion1(x, y) # aqui como tiene valor real se llama ARGUMENTOS
```

- En esta funcion se le asigna dos argumentos (x) e (y) al parámetro variable 1 y variable 2

## Funciones productivas y esteriles

- SI producen resultados: las llamaremos **funciones productivas (fruitful functions)** se usa **retrun**
- realizan una acción, pero **no devuelven un valor. No Usan return**. **funciones estériles (void functions).** Ejemplo función productiva :

```
def productiva(): #no tiene porque tener parámetros
    return "Hola"

print(productiva(), "Pepe")
```

```
    def productiva(): #no tiene porque tener parámetros
        return "Hola"

    print(productiva(), "Pepe")
```

```
    def idioma(lang):
        if lang == 'es':
           return "1. Castellano"
        elif lang == 'val':
           return "2. Valenciano"
        else:
            return "No sabemos q idioma es."

    print("Que idioma hablas: ")
    print("Pues yo hablo ", idioma('es'))
    print("Que idioma hablas: ")
    print("Pues yo hablo ", idioma('val'))
    print("Que idioma hablas: ")
    print("Pues yo hablo ", idioma('rr'))
    #aqui pasamo el valor dentro de una variable
    rt = "val"
    print("Que idioma hablas: ")
    print("Pues yo hablo ", idioma(rt))
    print("\tSi intento hacer esto rr = idioma('val') devuelve None\n"
    "\t- ESTA FUUNCION NO DEVUELVE NADA = None\n \t- Luego Nome es lo que se mete en la variable \n\t- Si definimos la variable con
    #idioma()
```

Ejemplos función esteril:

```
def esteril():
    print("Hola esteril")

esteril()
```

```
    def esteril():
        print("Hola esteril")
        # es teteril ya que no devuele ningun valoir solo ejecuta un a acción

    esteril()
```

```
    print("Hola")
    def mi_funcion():
        print("Ya estamos dentro de la funcion esteril")
        print("Esto tambien esta dentro de una funcion esteril")

    x = 5
    x += 5
    print(x)
    mi_funcion()
    print("final del codigo")
```

```
    def idioma(lang):
        if lang == 'es':
            print("1. Castellano")
        elif lang == 'val':
            print("2. Valenciano")
        else:
            print("No sabemos q idioma es.")

    print("Que idioma hablas")
    idioma('es')
    print("Pues yo hablo ")
    idioma('val')
    print("\tSi intento hacer esto rr = idioma('val') devuelve None\n"
    "\t- ESTA FUUNCION NO DEVUELVE NADA = None\n \t- Luego Nome es lo que se mete en la variable \n\t- Si definimos la variable con
    #idioma()
```

## Anidar funciones

en el emplo de abajo vemos que podemos llamar a una funcion dentro de otra : - ptint(type(32)) --> una funcion dentro de otra.

```
def grito():
    return "Tu nombre es "

#print(grito())
grito()
type(grito())

hola = "Segundo"#input("Tu nombre es: ")

def llamada(palabra):
    print("El nonmbre que tenemos BD es :")
    return  grito() + palabra

#pasamos como argumento la variable hola
llamada(hola)
```