

CÓNQUER BLOCKS

PYTHON

FUNCIONES LAMBDA Y DECORADORES

FUNCIONES LAMBDA

¿QUÉ SON? SON FUNCIONES ANONIMAS

¿PARA QUÉ SE USAN? PARA ABREVIAR SINTAXIS Y AHORRARNOS TIEMPO

Todo lo que podemos hacer con una función lambda puede hacerse con una función normal, pero no todo lo que podemos hacer con funciones normales puede hacerse con funciones lambda

FUNCIONES LAMBDA

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

```
triangulo1 = area_triangulo(5,7)  
triangulo2 = area_triangulo(9,6)
```

```
print(triangulo1, triangulo2)
```

✓ 0.0s

17.5 27.0

FUNCIONES LAMBDA

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

```
triangulo1 = area_triangulo(5,7)  
triangulo2 = area_triangulo(9,6)
```

```
print(triangulo1, triangulo2)
```

✓ 0.0s

17.5 27.0

```
area_triangulo=lambda base,altura:(base*altura/2)
```

```
triangulo1 = area_triangulo(5,7)  
triangulo2 = area_triangulo(9,6)
```

```
print(triangulo1, triangulo2)
```

✓ 0.0s

17.5 27.0

FUNCIONES LAMBDA

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

```
triangulo1 = area_triangulo(5,7)  
triangulo2 = area_triangulo(9,6)
```

```
print(triangulo1, triangulo2)
```

✓ 0.0s

17.5 27.0

```
area_triangulo=lambda base,altura:(base*altura/2)
```

```
triangulo1 = area_triangulo(5,7)  
triangulo2 = area_triangulo(9,6)
```

```
print(triangulo1, triangulo2)
```

✓ 0.0s

17.5 27.0

FUNCIONES LAMBDA

NO PUEDE TENER BUCLES O CONDICIONALES. SOLO PUEDE DEVOLVERNOS UN CÁLCULO.

```
triangulo1 = area_triangu(5,7)
triangulo2 = area_triangu(9,6)
```

```
print(triangulo1, triangulo2)
```

✓ 0.0s

17.5 27.0

funciones on demand,
on the go,
online...

```
triangulo1 = area_triangu(5,7)
```

```
triangulo2 = area_triangu(9,6)
```

```
print(triangulo1, triangulo2)
```

17.5 27.0

FUNCIONES LAMBDA

EJEMPLO: POTENCIA

```
#al_cubo=lambda numero:pow(numero,3)  
al_cubo=lambda numero:numero**3  
print(al_cubo(13))
```

✓ 0.0s

2197

FUNCIONES LAMBDA

EJEMPLO: TRABAJO DE FORMATO

```
destacar_valor=lambda comision:"i{}! $.format(comision)
comision_luis = 2300
print(destacar_valor(comision_luis))
```

✓ 0.0s

i2300! \$

FILTROS CON FUNCIONES LAMBDA

EJEMPLO: FILTRAR NUMEROS PARES DE UNA LISTA

```
def numero_par(num):  
    if num % 2==0:  
        return True  
  
numeros=[17,24,7,39,8,51,92]  
print(list(filter(numero_par, numeros)))
```

✓ 0.0s

[24, 8, 92]

FILTROS CON FUNCIONES LAMBDA

EJEMPLO: FILTRAR NUMEROS PARES DE UNA LISTA

```
def numero_par(num):  
    if num % 2==0:  
        return True
```

```
numeros=[17,24,7,39,8,51,92]  
print(list(filter(numero_par, numeros)))
```

✓ 0.0s

[24, 8, 92]

```
numeros=[17,24,7,39,8,51,92]  
print(list(filter(lambda numero_par:numero_par%2==0, numeros)))
```

✓ 0.0s

[24, 8, 92]

FUNCIONES LAMBDA COMO CLAVE

EJEMPLO: SORT

```
autores = ["Isaac Asimov", "Frank Herbert", "Douglas Adams"]  
autores.sort()  
print(autores)
```

✓ 0.0s

```
['Douglas Adams', 'Frank Herbert', 'Isaac Asimov']
```

FUNCIONES LAMBDA COMO CLAVE

EJEMPLO: SORT

```
autores = ["Isaac Asimov", "Frank Herbert", "Douglas Adams"]  
autores.sort()  
print(autores)
```

✓ 0.0s

```
['Douglas Adams', 'Frank Herbert', 'Isaac Asimov']
```

```
autores = ["Isaac Asimov", "Frank Herbert", "Douglas Adams"]  
autores.sort(key=lambda name:name.split(" ")[-1])  
print(autores)
```

✓ 0.0s

```
['Douglas Adams', 'Isaac Asimov', 'Frank Herbert']
```

FUNCIONES DECORADORES

SON FUNCIONES QUE AÑADEN FUNCIONALIDADES A FUNCIONES YA EXISTENTES EN NUESTRO PROGRAMA

ESTRUCTURA:

- ➡ **3 funciones (A, B, C) donde A recibe como parámetro la función B y devuelve la función C.**
- ➡ **Un decorador devuelve una función**

FUNCIONES DECORADORES

SON FUNCIONES QUE AÑADEN FUNCIONALIDADES A FUNCIONES YA EXISTENTES EN NUESTRO PROGRAMA

ESTRUCTURA:

- ➔ 3 funciones (A, B, C) donde A recibe B y devuelve la función C.
- ➔ Un decorador devuelve una función

```
def funcion_decorador(funcion):  
    def funcion_interna():  
        #codigo de funcion interna  
    return funcion_interna
```

FUNCION A **FUNCION B**
FUNCION C

FUNCIONES DECORADORES

EJEMPLO:

```
def suma():  
    print(15+20)
```

```
def resta():  
    print(10-3)
```

```
...
```

```
...
```

```
...
```


FUNCIONES DECORADORES

EJEMPLO:

```
def suma():  
    print(15+20)
```

```
def resta():  
    print(10-3)
```

```
...
```

```
...
```

```
...
```

```
def funcion_decoradora(funcion_parametro):  
    def funcion_interior():  
        # Acciones adicionales que decoran  
        print("Vamos a realizar el calculo: ")  
  
        funcion_parametro()  
  
        # Acciones adicionales que decoran  
        print("Hemos terminado el calculo")  
  
    return(funcion_interior)
```

FUNCIONES DECORADORES

EJEMPLO:

```
def suma():  
    print(15+20)
```

```
def resta():  
    print(10-3)
```

...

...

...

```
def funcion_decoradora(funcion_parametro):  
    def funcion_interior():  
        # Acciones adicionales que decoran  
        print("Vamos a realizar el calculo: ")  
  
        funcion_parametro()  
  
        # Acciones adicionales que decoran  
        print("Hemos terminado el calculo")  
  
    return(funcion_interior)
```

```
@funcion_decoradora  
def suma():  
    print(15+20)
```

```
@funcion_decoradora  
def resta():  
    print(10-3)
```

```
Vamos a realizar el calculo:  
35  
Hemos terminado el calculo  
Vamos a realizar el calculo:  
7  
Hemos terminado el calculo
```

FUNCIONES DECORADORES

```
def funcion_decoradora(funcion_parametro):  
    def funcion_interior(*args):  
        # Acciones adicionales que decoran  
        print("Vamos a realizar el calculo: ")  
  
        funcion_parametro(*args)  
  
        # Acciones adicionales que decoran  
        print("Hemos terminado el calculo")  
  
    return(funcion_interior)
```

```
@funcion_decoradora  
def suma(num1, num2, num3):  
    print(num1+num2+num3)
```

```
@funcion_decoradora  
def resta(num1, num2):  
    print(num1-num2)
```

```
suma(7,5,6)  
resta(4,3)
```

Función interna con argumentos

```
Vamos a realizar el calculo:  
18  
Hemos terminado el calculo  
Vamos a realizar el calculo:  
1  
Hemos terminado el calculo
```

FUNCIONES DECORADORES

```
def funcion_decoradora(funcion_parametro):  
    def funcion_interior(*args):  
        # Acciones adicionales que decoran  
        print("Vamos a realizar el calculo: ")  
  
        funcion_parametro(*args)  
  
        # Acciones adicionales que decoran  
        print("Hemos terminado el calculo")  
  
    return(funcion_interior)
```

```
@funcion_decoradora  
def suma(num1, num2, num3):  
    print(num1+num2+num3)
```

```
@funcion_decoradora  
def resta(num1, num2):  
    print(num1-num2)
```

```
suma(7,5,6)  
resta(4,3)
```

Función interna con
argumentos

Arguments

```
Vamos a realizar el calculo:  
18  
Hemos terminado el calculo  
Vamos a realizar el calculo:  
1  
Hemos terminado el calculo
```

FUNCIONES DECORADORES

```
def funcion_decoradora(funcion_parametro):  
    def funcion_interior(*args, **kwargs):  
        # Acciones adicionales que decoran  
        print("Vamos a realizar el calculo: ")  
  
        funcion_parametro(*args, **kwargs)  
  
        # Acciones adicionales que decoran  
        print("Hemos terminado el calculo")  
  
    return(funcion_interior)
```

```
@funcion_decoradora  
def potencia(base, exponente):  
    print(base**exponente)  
  
potencia(base=4, exponente=3)
```

✓ 0.0s

**Función interna con
argumentos**

```
Vamos a realizar el calculo:  
64  
Hemos terminado el calculo
```


FUNCIONES DECORADORES

```
def funcion_decoradora(funcion_parametro):  
    def funcion_interior(*args, **kwargs):  
        # Acciones adicionales que decoran  
        print("Vamos a realizar el calculo: ")  
  
        funcion_parametro(*args, **kwargs)  
  
        # Acciones adicionales que decoran  
        print("Hemos terminado el calculo")  
  
    return(funcion_interior)
```

```
@funcion_decoradora  
def potencia(base, exponente):  
    print(base**exponente)  
  
potencia(base=4, exponente=3)
```

✓ 0.0s

Función interna con
argumentos

Keyword arguments

```
Vamos a realizar el calculo:  
64  
Hemos terminado el calculo
```

CÔNQUER BLOCKS