





Decoradores

 Autoría	 Ana Raquel Martinez Carballo
 Etiquetas	
 Hora de creación	@5 de abril de 2024 19:48

Switch case

Hasta la versión 3.10, Python nunca tuvo una característica que implementara lo que hace la declaración switch en otros lenguajes de programación.

Por lo tanto, si querías ejecutar múltiples declaraciones condicionales, tendrías que usar la palabra clave `elif` de esta manera:

```
age = 120

if age > 90:
    print("You are too old to party, granny.")
elif age < 0:
    print("You're yet to be born")
elif age >= 18:
    print("You are allowed to party")
else:
    "You're too young to party"

# Output: You are too old to party, granny.
```

Ahora puedes implementar esta característica con las palabras clave `match` y `case`.

```
match term:
    case pattern-1:
        action-1
    case pattern-2:
        action-2
    case pattern-3:
```

```
        action-3
    case _:
        action-default
```

```
lang = input("What's the programming language you want to learn? ")

match lang:
    case "JavaScript":
        print("You can become a web developer.")

    case "Python":
        print("You can become a Data Scientist")

    case "PHP":
        print("You can become a backend developer")

    case "Solidity":
        print("You can become a Blockchain developer")

    case "Java":
        print("You can become a mobile app developer")
    case _:
        print("The language doesn't matter, what matters is you're learning")
```

Decoradores

A través de los decoradores seremos capaces reducir las líneas de código duplicadas, haremos que nuestro código sea legible, fácil de testear, fácil de mantener y sobre todo, tendremos un código mucho más Pythonico.

En Python las funciones son ciudadanos de primera clase, eso quiere decir que una función puede ser asignada a una variable, puede ser utilizada como argumento para otra función, o inclusive puede ser retornada:

```
def saludar():
    print('Hola soy una función')

def super_funcion(funcion):
```

```
funcion()  
  
funcion = saludar # Asignamos la función a una variable!  
  
super_funcion(funcion)
```



Un decorador no es más que una función la cual toma como input una función y a su vez retorna otra función.

Lo que nos debe quedar claro es que al momento de implementar un decorador estaremos trabajando, con por lo menos, 3 funciones. El input, el output y la función principal. Para que nos quede más en claro a mi me gusta nombrar a las funciones como: *a*, *b* y *c*.

```
def funcion_a(funcion_b):  
    def funcion_c():  
        print('Antes de la ejecución de la función a decorar'  
              funcion_b()  
              print('Después de la ejecución de la función a decora  
  
    return funcion_c
```



Un decorador en Python **es una función que recibe otra función como parámetro, le añade cosas y retorna una función diferente.**

Al nosotros utilizar la palabra **decorar** estamos indicando que queremos modificar el comportamiento de una función ya existente, pero sin tener que modificar su código. Esto es muy útil, principalmente, cuando queremos extender nuevas funcionalidades a dicha función. De allí el nombre **decorar**.

Para decorar una función basta con colocar, en su parte superior de dicha función, el decorador con el prefijo @.

```
@funcion_a
def saludar():
    print('Hola mundo!!')
```

```
def funcion_a(funcion_b):
    def wrapper(*args, **kwargs):
        print('Antes de la ejecución de la función a decorar')
        result = funcion_b(*args, **kwargs)
        print('Después de la ejecución de la función a decora

        return result

    return wrapper
```

```
@funcion_a
def suma(a, b):
    print("hola")
    return a + b
suma(1,2)
```

```
def measure_time(function):
    def wrapper(*args, **kwargs):
        import time

        start = time.time()
        result = function(*args, **kwargs)
        total = time.time() - start
        print(total, 'seconds' )
        return result

    return wrapper

@measure_time
def suma(a, b):
```

```
import time
time.sleep(1)
return a + b

print(suma(10, 20))
```