EL LIBRO DE PYTHON

CONTENIDO

Assert en Python
assert() en testing
assert() en funciones
assert() con clases
Deshabilitar assert

Assert en Python

El uso de `assert` en Python nos **permite realizar comprobaciones**. Si la expresión contenida dentro del mismo es `False`, se lanzará una excepción, concretamente `AssertionError`. Veamos un ejemplo:

```
assert(1==2)
# AssertionError
```

Es decir, si el contenido existente dentro del `assert` es igual a `False`, se lanzará la excepción. Se podría conseguir el mismo resultado haciendo lo siguiente, pero el uso de `assert()` resulta más cómodo.

```
if condicion:
    raise AssertionError()
```

Podemos también añadir un texto con información relevante acerca del `assert()`.

```
assert False, "El assert falló"
```

Aunque mucho cuidado, ya que la expresión anterior no es equivalente a la siguiente, siendo la misma errónea. Esto se debe a que en realidad se está evaluando `bool((False, "El assert falló"))`, lo que resulta ser siempre `True`. De hecho el siguiente código no lanzaría una excepción, cuando realmente se esperaría que lo hiciera.

```
# INCORRECTO
assert(False, "El assert falló")
```

Por otro lado, también se puede hacer uso del `assert()` sin usar paréntesis como se muestra a continuación.

```
x = "ElLibroDePython"
assert x == "ElLibroDePython"
```

assert() en testing

La función `assert()` puede ser también muy útil para realizar testing de nuestro código, especialmente para test unitarios o *unit tests*. Imagínate que tenemos una función `calcula_media()` que como su nombre indica calcula la media de un conjunto de números.

```
def calcula_media(lista):
    return sum(lista)/len(lista)
```

En el mundo de la programación es muy importante probar o *testear* el software, para asegurarse de que está libre de errores. Gracias al uso de `assert()` podemos realizar estas comprobaciones de manera automática.

```
assert(calcula_media([5, 10, 7.5]) == 7.5)
assert(calcula_media([4, 8]) == 6)
```

Por lo que si hacemos que estas comprobaciones sean parte de nuestro código, podríamos proteger nuestra función, asegurándonos de que nadie la "rompa".

assert() en funciones

Puede resultar útil usar `assert()` cuando queremos realizar alguna comprobación, como podría ser **dentro de una función**. En el siguiente ejemplo tenemos una función `suma()` que sólo suma las variables si son números enteros.

```
# Funcion suma de variables enteras
def suma(a, b):
    assert(type(a) == int)
    assert(type(b) == int)
    return a+b

# Error, ya que las variables no son int
suma(3.0, 5.0)
```

```
# Ok, los argumentos son int
suma(3, 5)
```

assert() con clases

Otro ejemplo podría verificar que un objeto **pertenece a una clase determinada**.

```
class MiClase():
    pass

class MiOtraClase():
    pass

mi_objeto = MiClase()
mi_otro_objeto = MiOtraClase()

# Ok
assert(isinstance(mi_objeto, MiClase))

# Ok
assert(isinstance(mi_otro_objeto, MiOtraClase))

# Error, mi_objeto no pertenece a MiOtraClase
assert(isinstance(mi_objeto, MiOtraClase))

# Error, mi_otro_objeto no pertenece a MiClase
assert(isinstance(mi_otro_objeto, MiClase))
```

Deshabilitar assert

A modo de curiosidad, es posible ejecutar un script de Python deshabilitando el `assert`. Supongamos que tenemos el siguiente código.

```
# ejemplo.py
assert(1==2)
```

Si ejecutamos nuestro script de la siguiente manera, los `assert` se eliminarán, por lo que no se producirá ninguna excepción.

```
$ python -O ejemplo.py
```

Anterior

← Excepciones en Python

Siguiente

Definiendo Excepciones →

Nuestra tienda
Colabora
Canal de telegram

Política de privacidad
Términos y condiciones
Contacta con nosotros