Python ‘==’ e ‘is’. Qual a diferença?

[](https://medium.com/@joaolggross?source=post_page-----73b69dcd22e1----------------------)

[João Gross](https://medium.com/@joaolggross?source=post_page-----73b69dcd22e1----------------------)

[Aug 11](https://medium.com/@joaolggross/python-e-is-qual-a-diferen%C3%A7a-73b69dcd22e1?source=post_page-----73b69dcd22e1----------------------) · 3 min read

Para quem está aprendendo Python e, principalmente, para quem criou sua base de conhecimento em programação em outras linguagens, tais como C e C++ (meu caso), os operadores == e is de Python podem causar bastante confusão.

Image for post



Foto de [Joeyy Lee](https://unsplash.com/@joeyy_anne?utm_source=medium&utm_medium=referral) no[Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

**Alguns exemplos**

Vamos partir direto para alguns exemplos comparativos de uso dos dois operadores para que possamos entender seu comportamento:

Nenhum problema aí, é um comportamento esperado. A princípio estamos comparando se x é igual a y, o que nos parece correto nesse caso.

Aqui já começamos a ter problemas. Por algum motivo o operador is apresenta um resultado diferente do operador ==, embora o conteúdo de x e y seja o mesmo.

Novamente, outro resultado inesperado (a princípio). Se x e y possuem o mesmo conteúdo porque x is y retorna False?

Agora sim! Neste último exemplo parece que finalmente voltou a fazer sentido. Mas o que realmente estão fazendo os operadores == e is?

**O operador '**=='

Este é o operador de **igualdade** em Python. Ele é utilizado para comparar os valores dos objetos em questão. Logo, ao fazer x == y a operação é algo como “O valor de x é igual ao valor de y?”.

Esse é o motivo pelo qual nos exemplos apresentados o operador == sempre retorna True. Em todos os casos os valores de x e y são iguais.

**O operador**'is'

O operador is compara **identidade**em Python. Para obter a identidade de um objeto é possível utilizar a função id(), que retorna um número inteiro único que se mantém constante durante o seu ciclo de vida.

Podemos observar que para cada valor atribuído a x o identificar único se altera. Vemos agora porque nos exemplos anteriores utilizando o operador is alguns resultados foram True enquanto outros False.

Nos dois exemplos acima os ids são iguais quando o valor atribuído às variáveis é 10, porém ao atribuir o valor 10000 os ids são diferentes. Isso ocorre devido à identidade que Python atribui a certos valores inteiros.

Segundo a documentação do Python 3, “[Integer Objects](https://docs.python.org/3/c-api/long.html)”:

*The current implementation keeps an array of integer objects for all integers between -5 and 256, when you create an int in that range you actually just get back a reference to the existing object. So it should be possible to change the value of 1. I suspect the behaviour of Python in this case is undefined. :-)*

Traduzindo, Python possui um array de inteiros para todos os inteiros do intervalo de -5 a 256, mantendo um identificador fixo para cada valor. O mesmo é aplicável ao [Python 2](https://docs.python.org/2/c-api/int.html).

Isso explica o motivo pelo qual para os valores 10 e 10000 o resultado do operador is é diferente.

**Atribuição e identidade**

Outra particularidade de Python é que ao realizar a atribuição de um objeto para outro utilizando =, o que na verdade ocorre é que ambos os objetos ficam com a mesma identidade.

A simples atribuição com = entre objetos pode ser particularmente problemática ao lidar com arrays:

Para evitar esse problema podemos utilizar o método copy() de listas:

Ao utilizar o método copy() uma nova identidade é criada para y. Isso pode evitar muitos problemas e tempo perdido com debugging de comportamentos inesperados.

Os operadores == e is são fundamentais para o desenvolvimento de software e inevitalmente qualquer programador irá utilizá-los em algum momento. Por isso entender estes conceitos é tão importante.