Python __eq__

Summary: in this tutorial, you'll learn how to use the Python `__eq__` method to compare two objects by their values.

Introduction to the Python __eq__ method

Suppose that you have the following `Person` class with three instance attributes: `first_name`, `last_name`, and `age`:

```
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
```

And you create two instances of the `Person` class:

```
john = Person('John', 'Doe', 25)
jane = Person('Jane', 'Doe', 25)
```

In this example, the `john` and `jane` objects are not the same object. And you can check it using the `is` operator (https://www.pythontutorial.net/advanced-python/python-is-operator/) :

```
print(john is jane)  # False
```

Also, when you compare `john` with `jane` using the equal operator (==), you'll get the result of False:

```
print(john == jane) # False
```

Since `john` and `jane` have the same age, you want them to be equal. In other words, you want the following expression to return `True` :

```
john == jane
```

To do it, you can implement the `__eq__` dunder method in the `Person` class.

Python automatically calls the `__eq__` method of a class when you use the == operator to compare the instances of the class. By default, Python uses the `is` operator if you don't provide a specific implementation for the __eq__ method.

The following shows how to implement the `__eq__` method in the `Person` class that returns `True` if two person objects have the same age:

```
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def __eq__(self, other):
        return self.age == other.age
```

Now, if you compare two instances of the `Person` class with the same age, it returns True:

```python
john = Person('John', 'Doe', 25)
jane = Person('Jane', 'Doe', 25)
print(john == jane)  # True
```

And if two instances of the Person class don't have the same age, the == operator returns False:

```python
john = Person('John', 'Doe', 25)
mary = Person('Mary', 'Doe', 27)
print(john == mary)  # False
```

The following compares a `Person` object with an integer:

```python
john = Person('John', 'Doe', 25)
print(john == 20)
```

It returns an error:

```
AttributeError: 'int' object has no attribute 'age'
```

To fix this, you can modify the `__eq__` method to check if the object is an instance of the `Person` class before accessing the `age` attribute.

If the other object isn't an instance of the `Person` class, the `__eq__` method returns `False` , like this:

```python
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def __eq__(self, other):
        if isinstance(other, Person):
            return self.age == other.age
        return False
```

And you can now compare an instance of the `Person` class with an integer or any object of a different type:

```
john = Person('John', 'Doe', 25)
print(john == 20)  # False
```

Putting it all together.

```
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def __eq__(self, other):
        if isinstance(other, Person):
            return self.age == other.age

        return False


john = Person('John', 'Doe', 25)
jane = Person('Jane', 'Doe', 25)
mary = Person('Mary', 'Doe', 27)

print(john == jane)  # True
print(john == mary)  # False
```

```python
john = Person('John', 'Doe', 25)
print(john == 20)  # False
```

Summary

- Implement the Python `__eq__` method to define the equality logic for comparing two objects using the equal operator (`==`)