Python __repr__

Summary: in this tutorial, you'll learn how to use the Python `__repr__` dunder method and the difference between the `__repr__` and `__str__` methods.

Introduction to the Python __repr__ magic method

The `__repr__` dunder method defines behavior when you pass an instance of a class (https://www.pythontutorial.net/python-oop/python-class/) to the `repr()` .

The `__repr__` method returns the string representation of an object. Typically, the `__repr__()` returns a string that can be executed and yield the same value as the object.

In other words, if you pass the returned string of the `object_name.__repr__()` method to the `eval()` function, you'll get the same value as the `object_name` . Let's take a look at an example.

First, define the `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
```

Second, create a new instance of the `Person` class and display its string representation:

```
person = Person('John', 'Doe', 25)
print(repr(person))
```

Output:

```
<__main__.Person object at 0x000001F51B3313A0>
```

By default, the output contains the memory address of the `person` object. To customize the string representation of the object, you can implement the `__repr__` method like this:

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

    def __repr__(self):
        return f'Person("{self.first_name}","{self.last_name}",{self.age})'
```

When you pass an instance of the `Person` class to the `repr()` , Python will call the `__repr__` method automatically. For example:

```
person = Person("John", "Doe", 25)
print(repr(person))
```

Output:

```
Person("John","Doe",25)
```

If you execute the return string `Person("John","Doe",25)` , it'll return the `person` object.

When a class doesn't implement the `__str__` (https://www.pythontutorial.net/python-oop/python-__str__/) method and you pass an instance of that class to the `str()` , Python returns the result of the `__repr__` method because internally the `__str__` method calls the `__repr__` method:

For example:

```
person = Person('John', 'Doe', 25)
print(person)
```

Output:

```
Person("John","Doe",25)
```

If a class implements the `__str__` method, Python will call the `__str__` method when you pass an instance of the class to the `str()` . For example:

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

    def __repr__(self):
        return f'Person("{self.first_name}","{self.last_name}",{self.age})'

    def __str__(self):
        return f'({self.first_name},{self.last_name},{self.age})'
```

```
person = Person('John', 'Doe', 25)
# use str()
print(person)

# use repr()
print(repr(person))
```

Output:

```
(John,Doe,25)
Person("John","Doe",25)
```

__str__ vs __repr__

The main difference between `__str__` and `__repr__` method is intended audiences.

The `__str__` method returns a string representation of an object that is human-readable while the `__repr__` method returns a string representation of an object that is machine-readable.

Summary

- Implement the `__repr__` method to customize the string representation of an object when `repr()` is called on it.
- The `__str__` calls `__repr__` internally by default.