Python None

Summary: in this tutorial, you'll learn about Python `None` and how to use it properly in your code.

Introduction to the Python None value

In Python, `None` is a special object of the `NoneType` class (https://www.pythontutorial.net/python-oop/python-class/) . To use the `None` value, you specify the `None` as follows:

```
None
```

If you use the `type()` function to check the type of the `None` value, you'll get `NoneType` class:

```
print(type(None))
```

Output:

```
<class 'NoneType'>
```

The `None` is a singleton object of the `NoneType` class. It means that Python creates one and only one `None` object at runtime.

Therefore, if you use the equality (`==`) or `is` operator to compare `None` with `None`, you'll get the result of `True` :

```
print(None == None)
print(None is None)
```

Output:

```
True
True
```

It's a good practice to use the `is` (https://www.pythontutorial.net/advanced-python/python-is-operator/) or `is not` operator to compare a value with `None` .

The reason is that the user-defined objects may change the equality operator's behavior by overriding the `__eq__()` method. For example:

```
class Apple:
    def __eq__(self, other):
        return True


apple = Apple()
print(apple == None)
```

Output:

```
True
```

Note that you cannot override the `is` operator behavior like you do with the equality operator (`==`).

It's also important to note that the None object has the following features:

- `None` is not zero (0, 0.0, …).
- `None` is not the same as `False`.
- `None` is not the same as an empty string (`''`).
- Comparing `None` to any value will return `False` except `None` itself.

The applications of the Python None object

Let's take some practical examples of using the `None` object.

1) Using Python None as an initial value for a variable

When a variable doesn't have any meaningful initial value, you can assign `None` to it, like this:

```
state = None
```

Then you can check if the variable is assigned a value or not by checking it with `None` as follows:

```
if state is None:
    state = 'start'
```

2) Using the Python None object to fix the mutable default argument issue

The following function appends a color to a list:

```
def append(color, colors=[]):
    colors.append(color)
    return colors
```

It works as expected if you pass an existing list:

```
colors = ['red', 'green']
append('blue', colors)

print(colors)
```

Output:

```
['red', 'green', 'blue']
```

However, the problem arises when you use the default value of the second parameter. For example:

```
hsl = append('hue')
print(hsl)

rgb = append('red')
print(rgb)
```

Output:

```
['hue']
['hue', 'red']
```

The issue is that the function creates the list once defined and uses the same list in each successive call.

To fix this issue, you can use the `None` value as a default parameter as follows:

```
def append(color, colors=None):
    if colors is None:
        colors = []

    colors.append(color)
    return colors
```

```
hsl = append('hue')
print(hsl)


rgb = append('red')
print(rgb)
```

Output:

```
['hue']
['red']
```

3) Using the Python None object as a return value of a function

When a function doesn't have a return value, it returns `None` by default. For example:

```
def say(something):
    print(something)


result = say('Hello')
print(result)
```

The `say()` function doesn't return anything; therefore, it returns `None` .

Summary

- `None` is a singleton object of the `NoneType` class.
- `None` is not equal to anything except itself.
- Use `is` or `is not` operator to compare `None` with other values.