Python __bool__

Summary: in this tutorial, you will learn how to implement the Python `__bool__` method to return boolean values for objects of a custom class.

Introduction to the Python __bool__ method

An object of a custom class (https://www.pythontutorial.net/python-oop/python-class/) is associated with a boolean value. By default, it evaluates to `True`. For example:

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


if __name__ == '__main__':
    person = Person('John', 25)
```

In this example, we define the `Person` class, instantiate an object, and show its boolean value. As expected, the `person` object is `True`.

To override this default behavior, you implement the `__bool__` special method. The `__bool__` method must return a boolean value, `True` or `False` .

For example, suppose that you want the `person` object to evaluate `False` if the age of a person is under 18 or above 65:

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

    def __bool__(self):
        if self.age < 18 or self.age > 65:
            return False
        return True


if __name__ == '__main__':
    person = Person('Jane', 16)
    print(bool(person))  # False
```

In this example, the `__bool__` method returns `False` if the age is less than 18 or greater than 65. Otherwise, it returns `True` . The person object has the age value of 16 therefore it returns False in this case.

The __len__ method

If a custom class doesn't have the `__bool__` method, Python will look for the `__len__()` method. If the `__len__` is zero, the object is `False` . Otherwise, it's `True` .

If a class doesn't implement the `__bool__` and `__len__` methods, the objects of the class will evaluate to `True` .

The following defines a `Payroll` class that doesn't implement `__bool__` but the `__len__` method:

```
class Payroll:
    def __init__(self, length):
        self.length = length

    def __len__(self):
        print('len was called...')
        return self.length


if __name__ == '__main__':
    payroll = Payroll(0)
    print(bool(payroll))  # False

    payroll.length = 10
    print(bool(payroll))  # True
```

Since the `Payroll` class doesn't override the `__bool__` method, Python looks for the `__len__` method when evaluating the Payroll's objects to a boolean value.

In the following example payroll's `__len__` returns 0, which is `False`:

```
payroll = Payroll(0)
print(bool(payroll))  # False
```

However, the following example `__len__` returns 10 which is `True`:

```
payroll.length = 10
print(bool(payroll))  # True
```

Summary

- All objects of custom classes return `True` by default.

- Implement the `__bool__` method to override the default. The `__bool__` method must return either `True` or `False` .
- If a class doesn’t implement the `__bool__` method, Python will use the result of the `__len__` method. If the class doesn’t implement both methods, the objects will be `True` by default.