Python __hash__

Summary: in this tutorial, you'll learn about the Python hash() function and how to override the `__hash__` method in a custom class.

Introduction to the Python hash function

Let's start with a simple example.

First, define the `Person` class (https://www.pythontutorial.net/python-oop/python-class/) with the `name` and `age` attributes:

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

Second, create two instances of the `Person` class:

```
p1 = Person('John', 22)
p2 = Person('Jane', 22)
```

Third, show the hashes of the `p1` and `p2` objects:

```
print(hash(p1))
print(hash(p2))
```

Output:

```
110373112736
110373572343
```

The `hash()` function accepts an object and returns the hash value as an integer. When you pass an object to the `hash()` function, Python will execute the `__hash__` special method of the object.

It means that when you pass the `p1` object to the `hash()` function:

```
hash(p1)
```

Python will call the `__hash__` method of the `p1` object:

```
p1.__hash__()
```

By default, the `__hash__` uses the object's identity and the `__eq__` (https://www.pythontutorial.net/python-oop/python-__eq__/) returns `True` if two objects are the same. To override this default behavior, you can implement the `__eq__` and `__hash__` .

If a class overrides the `__eq__` method, the objects of the class become unhashable. This means that you won't able to use the objects in a mapping type. For example, you will not able to use them as keys in a dictionary (https://www.pythontutorial.net/python-basics/python-dictionary/) or elements in a set (https://www.pythontutorial.net/python-basics/python-set/) .

The following `Person` class implements the `__eq__` method:

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

```
        self.name = name
        self.age = age

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

If you attempt to use the `Person` object in a set, you'll get an error. For example:

```
members = {
    Person('John', 22),
    Person('Jane', 22)
}
```

Python issues the following error:

```
TypeError: unhashable type: 'Person'
```

Also, the Person's object loses hashing because if you implement `__eq__`, the `__hash__` is set to `None`. For example:

```
hash(Person('John', 22))
```

Error:

```
TypeError: unhashable type: 'Person'
```

To make the `Person` class hashable, you also need to implement the `__hash__` method:

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

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

    def __hash__(self):
        return hash(self.age)
```

Now, you have the `Person` class that supports equality based on `age` and is hashable.

To make the `Person` work well in data structures like dictionaries, the hash of the class should remain immutable. To do it, you can make the `age` attribute of the `Person` class a read-only property (https://www.pythontutorial.net/python-oop/python-readonly-property/):

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

    @property
    def age(self):
        return self._age

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

    def __hash__(self):
        return hash(self.age)
```

Summary

- By default, `__hash__` uses the id of objects and `__eq__` uses the `is` operator for comparisons.
- If you implement `__eq__`, Python sets `__hash__` to `None` unless you implement `__hash__`.