Python Inheritance

Summary: in this tutorial, you'll learn about Python inheritance and how to use the inheritance to reuse code from an existing class.

Introduction to the Python inheritance

Inheritance allows a class (https://www.pythontutorial.net/python-oop/python-class/) to reuse the logic of an existing class. Suppose you have the following `Person` class:

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

    def greet(self):
        return f"Hi, it's {self.name}"
```

The `Person` class has the `name` attribute and the `greet()` method.

Now, you want to define the `Employee` that is similar to the `Person` class:

```python
class Employee:
    def __init__(self, name, job_title):
        self.name = name
        self.job_title = job_title

    def greet(self):
        return f"Hi, it's {self.name}"
```

The `Employee` class has two attributes `name` and `job_title`. It also has the `greet()` method that is exactly the same as the `greet()` method of the `Person` class.

To reuse the `greet()` method from the `Person` class in the `Employee` class, you can create a relationship between the `Person` and `Employee` classes. To do it, you use inheritance so that the `Employee` class inherits from the `Person` class.

The following redefines the `Employee` class that inherits from the `Person` class:

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

By doing this, the `Employee` class behaves the same as the `Person` class without redefining the greet() method.

This is a **single inheritance** because the `Employee` inherits from a single class (`Person`). Note that Python also supports multiple inheritances where a class inherits from multiple classes.

Since the `Employee` inherits attributes and methods of the `Person` class, you can use the instance of the Employee class as if it were an instance of the `Person` class.

For example, the following creates a new instance of the `Employee` class and call the `greet()` method:

```
employee = Employee('John', 'Python Developer')
print(employee.greet())
```

Output:

```
Hi, it's John
```

Inheritance terminology

The `Person` class is the **parent class**, the **base class**, or the **super class** of the `Employee` class. And the `Employee` class is a **child** class, a **derived class**, or a **subclass** of the `Person` class.

The `Employee` class **derives from**, **extends**, or **subclasses** the `Person` class.

The relationship between the `Employee` class and `Person` class is **IS-A** relationship. In other words, an employee **is a** person.

type vs. isinstance

The following shows the type of instances of the `Person` and `Employee` classes:

```
person = Person('Jane')
print(type(person))

employee = Employee('John', 'Python Developer')
print(type(employee))
```

Output:

```
<class '__main__.Person'>
<class '__main__.Employee'>
```

To check if an object is an instance of a class, you use the `isinstance()` method. For example:

```
person = Person('Jane')
print(isinstance(person, Person))  # True

employee = Employee('John', 'Python Developer')
print(isinstance(employee, Person))  # True
print(isinstance(employee, Employee))  # True
print(isinstance(person, Employee))  # False
```

Output:

```
True
True
True
False
```

As clearly shown in the output:

- The `person` is an instance of the `Person` class.
- The `employee` is an instance of the `Employee` class. It's also an instance of the `Person` class.
- The `person` is not an instance of the `Employee` class.

In practice, you'll often use the `isinstance()` function to check whether an object has certain methods. Then, you'll call the methods of that object.

issubclass

To check if a class is a subclass of another class, you use the `issubclass()` function. For example:

```
print(issubclass(Employee, Person)) # True
```

The following defines the `SalesEmployee` class that inherits from the `Employee` class:

```python
class SalesEmployee(Employee):
    pass
```

The `SalesEmployee` is the subclass of the `Employee` class. It's also a subclass of the `Person` class as shown in the following:

```python
print(issubclass(SalesEmployee, Employee)) # True
print(issubclass(SalesEmployee, Person)) # True
```

Note that when you define a class that doesn't inherit from any class, it'll implicitly inherit from the built-in `object` class.

For example, the `Person` class inherits from the `object` class implicitly. Therefore, it is a subclass of the `object` class:

```python
print(issubclass(Person, object)) # True
```

In other words, all classes are subclasses of the `object` class.

Summary

- Inheritance allows a class to reuse existing attributes and methods of another class.
- The class that inherits from another class is called a child class, a subclass, or a derived class.
- The class from which other classes inherit is call a parent class, a super class, or a base class.
- Use `isinstance()` to check if an object is an instance of a class.
- Use `issubclass()` to check if a class is a subclass of another class.