Python Abstract Classes

Summary: in this tutorial, you'll learn about Python Abstract classes and how to use it to create a blueprint for other classes.

Introduction to Python Abstract Classes

In object-oriented programming, an abstract class is a class (https://www.pythontutorial.net/python-oop/python-class/) that cannot be instantiated. However, you can create classes that inherit from an abstract class.

Typically, you use an abstract class to create a blueprint for other classes.

Similarly, an abstract method is an method without an implementation. An abstract class may or may not include abstract methods.

Python doesn't directly support abstract classes. But it does offer a module (https://www.pythontutorial.net/python-basics/python-module/) that allows you to define abstract classes.

To define an abstract class, you use the `abc` (abstract base class) module.

The `abc` module provides you with the infrastructure for defining abstract base classes.

For example:

```
from abc import ABC


class AbstractClassName(ABC):
    pass
```

To define an abstract method, you use the `@abstractmethod` decorator:

```
from abc import ABC, abstractmethod


class AbstractClassName(ABC):
    @abstractmethod
    def abstract_method_name(self):
        pass
```

Python abstract class example

Suppose that you need to develop a payroll program for a company.

The company has two groups of employees: full-time employees and hourly employees. The full-time employees get a fixed salary while the hourly employees get paid by hourly wages for their services.

The payroll program needs to print out a payroll that includes employee names and their monthly salaries.

To model the payroll program in an object-oriented way, you may come up with the following classes: `Employee`, `FulltimeEmployee`, `HourlyEmployee`, and `Payroll`.

To structure the program, we'll use modules (https://www.pythontutorial.net/python-basics/python-module/), where each class is placed in a separate module (or file).

The Employee class

The `Employee` class represents an employee, either full-time or hourly. The `Employee` class should be an abstract class because there're only full-time employees and hourly employees, no general employees exist.

The `Employee` class should have a property that returns the full name of an employee. In addition, it should have a method that calculates salary. The method for calculating salary should be an abstract method.

The following defines the `Employee` abstract class:

```
from abc import ABC, abstractmethod


class Employee(ABC):
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def full_name(self):
        return f"{self.first_name} {self.last_name}"

    @abstractmethod
    def get_salary(self):
        pass
```

The FulltimeEmployee class

The `FulltimeEmployee` class inherits from the `Employee` class. It'll provide the implementation for the `get_salary()` method.

Since full-time employees get fixed salaries, you can initialize the salary in the constructor of the class.

The following illustrates the `FulltimeEmployee` class:

```
class FulltimeEmployee(Employee):
    def __init__(self, first_name, last_name, salary):
        super().__init__(first_name, last_name)
        self.salary = salary

    def get_salary(self):
        return self.salary
```

The HourlyEmployee class

The `HourlyEmployee` also inherits from the `Employee` class. However, hourly employees get paid by working hours and their rates. Therefore, you can initialize this information in the constructor of the class.

To calculate the salary for the hourly employees, you multiply the working hours and rates.

The following shows the `HourlyEmployee` class:

```
class HourlyEmployee(Employee):
    def __init__(self, first_name, last_name, worked_hours, rate):
        super().__init__(first_name, last_name)
        self.worked_hours = worked_hours
        self.rate = rate

    def get_salary(self):
        return self.worked_hours * self.rate
```

The Payroll class

The `Payroll` class will have a method that adds an employee to the employee list and print out the payroll.

Since fulltime and hourly employees share the same interfaces (`full_time` property and `get_salary()` method). Therefore, the Payroll class doesn't need to distinguish them.

The following shows the `Payroll` class:

```
class Payroll:
    def __init__(self):
        self.employee_list = []

    def add(self, employee):
        self.employee_list.append(employee)

    def print(self):
        for e in self.employee_list:
            print(f"{e.full_name} \t ${e.get_salary()}")
```

The main program

The following `app.py` uses the `FulltimeEmployee`, `HourlyEmployee`, and `Payroll` classes to print out the payroll of five employees.

```
from fulltimeemployee import FulltimeEmployee
from hourlyemployee import HourlyEmployee
from payroll import Payroll

payroll = Payroll()

payroll.add(FulltimeEmployee('John', 'Doe', 6000))
payroll.add(FulltimeEmployee('Jane', 'Doe', 6500))
payroll.add(HourlyEmployee('Jenifer', 'Smith', 200, 50))
payroll.add(HourlyEmployee('David', 'Wilson', 150, 100))
payroll.add(HourlyEmployee('Kevin', 'Miller', 100, 150))

payroll.print()
```

Output:

```
John Doe          $6000
Jane Doe          $6500
Jenifer Smith     $10000
David Wilson      $15000
Kevin Miller      $15000
```

When to use abstract classes

In practice, you use abstract classes to share the code among several closely related classes. In the payroll program, all subclasses of the `Employee` class share the same `full_name` property.

Summary

- Abstract classes are classes that you cannot create instances from.
- Use `abc` module to define abstract classes.