ABC
Abstract Base Class

OOP – Spring 2022 (Python)

Abstract Base Class (ABC)

An abstract class is a special class that cannot be instantiated.

Child classes can inherit abstract class.

Mainly abstract classes are used to create a model for other classes.

Bird, Plant, Animal, Player, Student are examples of ABC.

Abstract Base Class (ABC)

Unlike other languages Python has no keyword named abstract. There are two conditions to become an abstract class.

1. An abstract class has to inherit "ABC" class
2. requires (minimum) one abstract method

Python has a module "abc" having class ABC

Abstract Method

Abstract function is to be decorated as "abstractmethod".

1. An abstract class has to inherit "ABC" class
2. requires (minimum) one abstract method

Python has a module "abc" having class ABC

Abstract Class – Example 1

```
from abc import *

class My_ABC(ABC):
    pass

def main():
    my_abc = My_ABC()
    print ('Object of My_ABC is
successfully created')

main()
```

Output:

```
Object of My_ABC is successfully created
```

Abstract Class – Example 2

```
from abc import *

class My_ABC:
  @abstractmethod
  def abstract_function(self):
   print ('I am abstract function')

def main():
  my_abc = My_ABC()
  my_abc.abstract_function()

main()
```

Output:

```
I am abstract function
```

Abstract Class – Example 3

```
from abc import *

class My_ABC(ABC):
  @abstractmethod
  def abstract_function(self):
    print ('I am abstract function')

def main():
  my_abc = My_ABC()
  my_abc.abstract_function()

main()
```

```
from abc import *

class My_ABC(ABC):
  @abstractmethod
  def abstract_function(self):
    print ('I am abstract function')

def main():
  my_abc = My_ABC()
  my_abc.abstract_function()

main()
```

Error Message:

```
Traceback (most recent ca
File "e:\subjects\spring
2022\oop_python\class exe
25\abstract3.py", line 12
main()
File "e:\subjects\spring
2022\oop_python\class exe
25\abstract3.py", line 9,
my_abc = My_ABC()

TypeError: Can't instantiate abstract class My_ABC
with abstract methods abstract_function
```

Abstract Class Solid Example

We 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, we may come up with the following classes: Employee, Full_Time_Employee, Hourly_Employee, and Payroll.

To structure the program, we'll use modules, where each class is placed in a separate module (or file).

Ref: https://www.pythontutorial.net/python-oop/python-abstract-class/

The Employee class - Description

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:

The Employee class - Code

```
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

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

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

The Fulltime_Employee class - Description

The Full_Time_Employee 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 Full_Time_Employee class:

The Fulltime_Employee class - Code

```
from employee import *

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 Hourly_Employee class - Description

The Hourly_Employee 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 Hourly_Employee class:

The Hourly_Employee class - Code

```
from employee import *

class Hourly_Employee(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 - Description

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:

The Payroll class - Code

```
from employee import *

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 Full_Time_Employee, Hourly_Employee, and Payroll classes to print out the payroll of five employees.

The Payroll class - Code

```
from employee import *

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()}")
```

OUTPUT – Payroll System

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