MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Object-Oriented Programming

6. Write a program to read 3 subject marks and display pass or failed using class and object

Object-Oriented Programming

- **Procedural programming:** A procedural program is typically a **list of instructions** that execute one after the other starting from the top of the line.
- On the other hand, **object-oriented programs** are built around well objects.
- **Procedural:** Tasks are treated as step-by-step iterations where **common tasks are placed in functions** that are called as needed.
- This coding style favors **iteration, sequencing, selection, and modularization**. Python excels in implementing this particular paradigm.

Object-Oriented Programming

Procedural coding example

```
def add(any_list):
    sum = 0
    for x in any_list:
        sum += x
    return sum

print(add(my_list))
```

Object-Oriented example

```
class ChangeList(object):
    def __init__(self, any_list):
        self.any_list = any_list
    def do_add(self):
        self.sum = sum(self.any_list)


create_sum = ChangeList(my_list)
create_sum.do_add()
print(create_sum.sum)
```

Object-Oriented Programming

- Python is a **multi-paradigm programming language**. It supports different programming approaches.
- **Classes and objects** are the two main aspects of object oriented programming.
- A **class** creates a new type where **objects** are **instances** of the class.
- An object has two characteristics:
  - attributes
  - behavior

Object-Oriented Programming

- **Objects can store data using** ordinary variables that *belong* to the object.
- **Variables** that belong to an object or class are referred to as **fields**
- **Objects** can also have **functionality** by using functions that *belong* to a class. Such functions are called **methods** of the class.
- **Method:** A special kind of function that is defined in a class definition
- Collectively, the fields and methods can be referred to as the **attributes** of that class

Object-Oriented Programming

- Fields are of two types - they can belong to each **instance/object of the class** or they can **belong to the class itself.**
- They are called **instance variables** and **class variables** respectively.
- A class is created **using the class keyword.** *The **fields and methods** of the class are listed in an indented block.*

```
class Person:
    pass  # An empty block

p = Person()
print(p)
```

an empty block which is indicated using the pass statement.

- create an object/instance of this class using the **name of the class followed by a pair of parentheses**
- It tells us that we have an instance of the Person class in the __main__ module.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Object-Oriented Programming

- The **self**: **Class methods** have only one **specific difference from ordinary functions**
- they must have an extra first name that has to be added to the beginning of the parameter list, but you **do not give a value** for this parameter when you call the method, **Python will provide it.**
- You have a class called **MyClass** and an instance of this class called **myobject**.
- When **you call a method** of this object as **myobject.method(arg1, arg2),**
- this is automatically converted by Python into **MyClass.method(myobject, arg1, arg2) -** this is all the special **self** is about.
- This also means that if **you have a method which takes no arguments**, then you still have to have one argument - the self.

Methods

- classes/objects can have **methods just like functions** except that we have an extra self variable.

```
class Person:
    def say_hi(self):
        print('Hello, how are you?')

p = Person()
p.say_hi()
```

- say_hi method takes no parameters but still has the self in the function definition

The __init__ method

significance of the __init__ method:

- The __init__ method is run **as soon as an object of a class** is instantiated (i.e. **created**).
- The method is **useful to do any initialization** (i.e. passing initial values to your object) you want to do with your object.
- The **double underscores** both at the beginning and at the end of the name.

The __init__ method

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

    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Swaroop')
p.say_hi()
# The previous 2 lines can also be written as
# Person('Swaroop').say_hi()
```

```
class students():                       # classs name
    def __init__(self,name,contact):     # init function ...by default parameter self,
        self.name=name
        self.contact=contact

    def getdata(self):                  # function getdata  by default parameter self,
      self.name = input(" Enter name of student ")
      self.contact= input(" Enter contact number ")

    def printdata(self):                ## function printdata  by default parameter self
        print("\n Name of student is "+self.name, " \n Contact number is "+self.contact)


raj=students("",0)                      # object of class....can access methods of class
raj.getdata()
raj.printdata()
```

The __init__ method

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

    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Swaroop')
p.say_hi()
# The previous 2 lines can also be written as
# Person('Swaroop').say_hi()
```

Inheritance

- Inheritance is the process by which one class takes on the attributes and methods of another.
- Newly formed classes are called **child classes**, and the classes that child classes are derived from are called **parent classes**.
- Child classes can **override or extend** the attributes and methods of parent classes.
- In other words, child classes **inherit all of the parent's attributes and methods**
- …. but can also specify attributes and methods that are unique to themselves.

```
class students():                    # superclass
    def __int__(self,name,contact):
        self.name=name
        self.contact=contact

    def getdata(self):
        print(" Enter student data ")
        self.name= input(" Enter name ofstudents ")
        self.contact= int (input(" Enter contact number "))

    def printdata(self):
        print("\n Studnet Information  ")
        print(" Name "+self.name," Contact is ",self.contact)
```

```
class enggstudents(students):        # inheritance .....
subclass
    def __int__(self,age):
        self.age=age

    def input(self):
        print(" Enter age of students ")
        self.age= int (input(" Enter age of students "))

    def display(self):
        print(" Age is : ",self.age)

abc=enggstudents()
abc.getdata()              # Inherited class can use methods
of base class
abc.printdata()
abc.input()
abc.display()
```

Special Function

Operator	Expression	Internally
Addition	`p1 + p2`	`p1.__add__(p2)`
Subtraction	`p1 - p2`	`p1.__sub__(p2)`
Multiplication	`p1 * p2`	`p1.__mul__(p2)`
Power	`p1 ** p2`	`p1.__pow__(p2)`
Division	`p1 / p2`	`p1.__truediv__(p2)`
Floor Division	`p1 // p2`	`p1.__floordiv__(p2)`
Remainder (modulo)	`p1 % p2`	`p1.__mod__(p2)`
Bitwise Left Shift	`p1 << p2`	`p1.__lshift__(p2)`
Bitwise Right Shift	`p1 >> p2`	`p1.__rshift__(p2)`
Bitwise AND	`p1 & p2`	`p1.__and__(p2)`
Bitwise OR	`p1 | p2`	`p1.__or__(p2)`
Bitwise XOR	`p1 ^ p2`	`p1.__xor__(p2)`

Polymorphism

- polymorphism is the condition of occurrence in different forms.
- Polymorphism is a very important concept in programming.
- It refers to the use of a single type entity (method, operator or object) to represent different types in different scenarios.

+ operator

For integer data types, + operator is used to perform arithmetic addition operation

string data types, + operator is used to perform concatenation

```
num1 = 1
num2 = 2
print(num1+num2)
```

```
str1 = "Python"
str2 = "Programming"
print(str1+" "+str2)
```

Polymorphism

Polymorphic len() function

```
print(len("Programiz"))
print(len(["Python", "Java", "C"]))
print(len({"Name": "John", "Address": "Nepal"}))
```

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

STAY SAFE & HEALTHY !!