Mayank Singh

01, Dec, 2022

What Is Object-Oriented Programming in Python?

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects. Put another way, object-oriented programming is an approach for modeling concrete, real-world things, like cars, as well as relations between things, like companies and employees, students and teachers, and so on. OOP models real-world entities as software objects that have some data associated with them and can perform certain functions.

READ MORE

https://realpython.com/python3-object-oriented-programming/#:~:text=Programming%20with%20Python.-,What%20Is%20Object%2DOriented%20F
(https://realpython.com/python3-object-oriented-programming/#:~:text=Programming%20with%20Python.-,What%20Is%20Object%2DOriented%20F

Classes and Objects

Class is a blueprint/set of rules stating the behaviour of its object. Object of the class follows the set of rules of its class.

Types of Classes in python:

1. Builtin class - List, tuple, dict ect.
2. User defined class - Atm (in our code)

Object is an instance of the class. Object can access all the attributes/properties and behaviour/methods of the class, it belongs to.

Rules of the class:

1. Declare all variables of class inside the constructor ('**init**')
2. Always declare variables with self, ex - self.name = 'Mayank'
3. Only making a class does not provide any ouput, unless its object is not created.

In [1]:

```
# Defining a class without creating object - example
# It will not give any output
class Atm:
    def __init__(self):
        self.pin = ''
        self.balance = 0
        print('Object is not created') # it will not get printed as there is no c
```

syntax to create object

object name = class name()

In [2]:

```
# Defining a class with object - example
# It will give output
class Atm:
    # constructor
    def __init__(self):
        self.pin = ''
        self.balance = 0
        print('Object created and hence printed')
A = Atm() # object created
```

```
Object created and hence printed
```

In [3]:

```
# verify if object "A" is of which class
print(type(A))
```

```
<class '__main__.Atm'>
```

Constructor

In the below code **init**(self) function is termed as a constructor. Constructors are special types of functions inside the class. It has a super power that means constructors need not to be called explicitly like other functions and when an object is created of the class, constructors are called automatically/by default and anything inside construtor will be executed by default.

The constructor is a method that is called when an object is created of a class. The creation of the constructor depends on the programmer, or else Python will automatically generate the default constructor. It can be used in three types - Parameterized Constructor, Non-Parameterized Constructor, Default Constructor.

Benefits of Constructor: It is use to hold those functionalities that should not be dependent on user actions. Mainly all the configurations realted functinalities are kept inside the constructron. In other words the things that should remain out of the control of user are placed inside the constructor.

And hence in above code of lines when object **A** was created, print statement was executed even though we had not called **init** function.

Types of Constructor - Non Parameterized and Parameterized

In [4]:

```python
class Atm:
    # constructor
    def __init__(self):
        self.pin = ''
        self.balance = 0
        # calling one of the function inside the constructor. menu() function is
        #so that when an object is created menu() function is called automaticall
        #the options available
        self.menu()

    def menu(self):
        user_input = input("""
        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        """)
A = Atm() # object created
```

```
        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        6
```

In [5]:

```
class Atm:
    # constructor
    def __init__(self):
        self.pin = ''
        self.balance = 0
        self.menu()

    def menu(self):
        user_input = input("""
        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        """)
        # creating skeleton of our app
        if user_input == '1':
            pass
            #create pin
        elif user_input == '2':
            pass
            # change pin
        elif user_input == '3':
            pass
            # check balance
        elif user_input == '4':
            pass
            # withdraw amount
        else:
            pass
            # exit from the app
A = Atm()
```

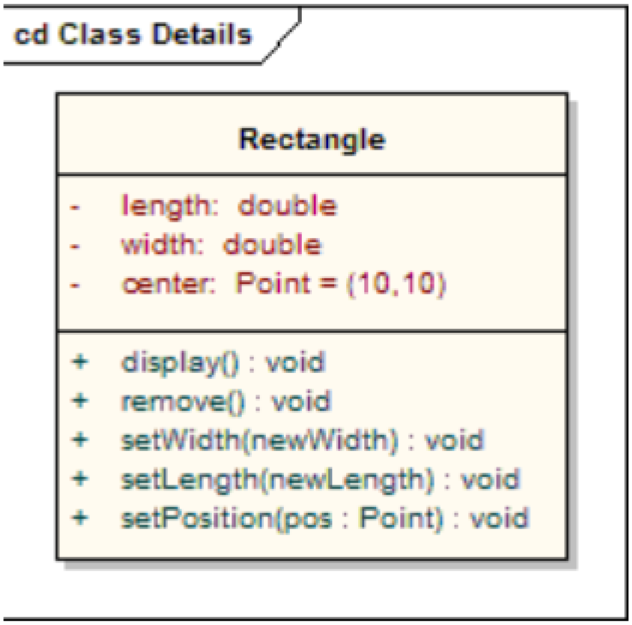
```
        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        6
```

In [6]:

```
class Atm:

 # constructor(special function)->superpower ->
  def __init__(self):
    print(id(self))
    self.pin = ''
    self.balance = 0
    self.menu()

  def menu(self):
    user_input = input("""
    Hi how can I help you?
    1. Press 1 to create pin
    2. Press 2 to change pin
    3. Press 3 to check balance
    4. Press 4 to withdraw
    5. Press anything to exit to exit
    """)
    ## All other functions are called inside the menu function so that whenev
    ## all other functions are called after checking if conditions.
    if user_input == '1':
      self.create_pin()
    elif user_input == '2':
      self.change_pin()
    elif user_input == '3':
      self.check_balance()
    elif user_input == '4':
      self.withdraw()
    else:
      print('Thank you for banking with us')
      exit()

  def create_pin(self):
    user_pin = input('enter your pin')
    self.pin = user_pin

    user_balance = int(input('enter balance'))
    self.balance = user_balance

    print('pin created successfully')
    self.menu()

  def change_pin(self):
    old_pin = input('enter old pin')

    if old_pin == self.pin:
     # let him change the pin
      new_pin = input('enter new pin')
      self.pin = new_pin
      print('pin change successful')
      self.menu()
    else:
      print('nai karne de sakta re baba')
      self.menu()

  def check_balance(self):
        user_pin = input('enter your pin')
        if user_pin == self.pin:
            print('your balance is ',self.balance)
            self.menu()
        else:
            print('chal nikal yahan se')
            self.menu()

    def withdraw(self):
        user_pin = input('enter the pin')
        if user_pin == self.pin:
          # allow to withdraw
            amount = int(input('enter the amount'))
            if amount <= self.balance:
                self.balance = self.balance - amount
                print('withdrawl successful.balance is',self.balance)
                self.menu()
            else:
                print('abe garib')
                self.menu()
        else:
            print('sale chor')
            self.menu()
A = Atm()
```

```
2259603632624

        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Press anything to exit to exit
        6
Thank you for banking with us
```

Class Diagram


cd Class Details
Rectangle
- length: double
- width: double
- center: Point = (10,10)
+ display() : void
+ remove() : void
+ setWidth(newWidth) : void
+ setLength(newLength) : void
+ setPosition(pos : Point) : void

Magic Methods - Dunder Methods

Magic methods in Python are the special methods that start and end with the double underscores. They are also called dunder methods. Magic methods are not meant to be invoked directly by you, but the invocation happens internally from the class on a certain action. For example, when you add two numbers using the + operator, internally, the **add**() method will be called.

Built-in classes in Python define many magic methods. Use the dir() function to see the number of magic methods inherited by a class. For example, the following lists all the attributes and methods defined in the int class.

Unlike other methods, it is not required to be called, it gets automatically called, as soon as object of the class is created **Read_More** [https://www.tutorialsteacher.com/python/magic-methods-in-python (https://www.tutorialsteacher.com/python/magic-methods-in-python)](https://www.tutorialsteacher.com/python/magic-methods-in-python)

```
>>> dir(int)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__',
'__float__', '__floor__', '__floordiv__', '__format__', '__ge__',
'__getattribute__', '__getnewargs__', '__gt__', '__hash__',
'__index__', '__init__', '__init_subclass__', '__int__', '__invert__',
'__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__',
'__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__',
'__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__',
'__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
'__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__',
'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag',
'numerator', 'real', 'to_bytes']
```

In [7]:

```
class Fraction:

# parameterized constructor
    def __init__(self,x,y):
        self.num = x
        self.den = y

    def __str__(self):
        return '{}/{}'.format(self.num,self.den)
    # self contains fr1 - i.e. nem and dem both, similarly other contains fr2, ag
    def __add__(self,other):
        new_num = self.num*other.den + other.num*self.den
        new_den = self.den*other.den

        return '{}/{}'.format(new_num,new_den)

    def __sub__(self,other):
        new_num = self.num*other.den - other.num*self.den
        new_den = self.den*other.den

        return '{}/{}'.format(new_num,new_den)

    def __mul__(self,other):
        new_num = self.num*other.num
        new_den = self.den*other.den

        return '{}/{}'.format(new_num,new_den)

    def __truediv__(self,other):
        new_num = self.num*other.den
        new_den = self.den*other.num

        return '{}/{}'.format(new_num,new_den)

    def convert_to_decimal(self):
        return self.num/self.den
fr1 = Fraction(3,4)
fr2 = Fraction(1,2)
```

In [8]:

```
print(fr2)
# 3/4
```

```
1/2
```

In [9]:

```
print(fr1 + fr2)
print(fr1 - fr2)
print(fr1 * fr2)
print(fr1 / fr2)
```

```
10/8
2/8
3/8
6/4
```

In [ ]:

```
# https://www.youtube.com/watch?v=Trvqq5w7-0Q&t=5348s
```