Mayank Singh

2, Dec, 2022

Power of Object/Object creation

The creation of objects gives object the power to access all the attributes and methods of the class to which object belongs to. Any attribute/method not created in the class can not be accessed and python gives an error. Refer ex below as illustration of the concept

In [1]:
```
class SamplezClass():
    def __init__(self):
        self.name = 'Mayank'
        self.gender = 'Male'
    def testfun(self):
        print('This code is for testing the call of function')
s=SamplezClass() # object created
```

In [2]:
```
## object "s" has the power to access the attributes as well as function of the c
print(s.name)
print(s.gender)
print(s.testfun())
```

```
Mayank
Male
This code is for testing the call of function
None
```

In [3]:
```
# but if we try to access anything not created inside the class, python throws er
s.age
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [3], in <cell line: 2>()
      1 # but if we try to access anything not created inside the class, python
throws error
----> 2 s.age

AttributeError: 'SamplezClass' object has no attribute 'age'
```

Creation of attribute is possible from outside the class

In [4]:
```python
s.age = 20
print(s.age)
```

```
20
```

Reference Variables

- Reference variables hold the objects
- We can create objects without reference variable as well
- An object can have multiple reference variables
- Assigning a new reference variable to an existing object does not create a new object

A - Reference varibale hold the object/s

when creating object of the class for eg. **s = SamplezClass()**, here **s** is not the object. Till now we are thinking **s** to be object, but in fact **s** contains the reference/address of the object created using **SamplezClass**. That's why the heading says ref variable **s** holds the object/s and it **is not** the object

B - We can create create objects without reference variable as well

Object creation is possible without any variable as well. **SamplezClass()** is valid syntax to create object of the class **SamplezClass**

In [5]:
```python
# object without a reference
class SamplezClass():
    def __init__(self):
        self.name = 'Mayank'
        self.gender = 'Male'
    def testfun(self):
        print('This code is for testing the call of function')
SamplezClass() # object created
```

Out[5]:
```
<__main__.SamplezClass at 0x28e21c8f7c0>
```

C - An object can have multiple reference variables

- p = Person() now we can have another object using previous refe var **p**, eg. below: p = q, **q** is a valid ref variable of object created for class **Person**.

In [6]:
```python
class Person:

  def __init__(self):
    self.name = 'nitish'
    self.gender = 'male'

p = Person()
q = p
```

D - Assigning a new reference variable to an existing object does not create a new object

Whne in the above description, creating **p** as new ref varibale, does not mean we have created new object. It only means that we have created multiple ref variables of the same object **SamplezClass**. And that's why **id** of the ref variable is same. Making changes to any of the ref variables, affects the other ref varibale as well.

In [7]:
```python
print(id(p))
print(id(q))
```

```
2809476713920
2809476713920
```

In [8]:
```python
## Making changes to any of the ref variables, affects the other ref varibale as
print(p.name)
print(q.name)
q.name = 'ankit'
print(q.name)
print(p.name)
```

```
nitish
nitish
ankit
ankit
```

Pass/call by reference

Python is neither call-by-value nor call-by-reference. It is call-by-sharing (also known as “call-by-object” or “call-by-object-sharing”) But we can use it as Pass by reference by some means. By reference means that the argument you’re passing to the function is a reference to a variable that already exists in memory rather than an independent copy of that variable. **OR** Pass by reference means that you have to pass the function(reference) to a variable which refers that the variable already exists in memory.

In [13]:
```
class Person:

    def __init__(self,name,gender):
        self.name = name
        self.gender = gender

# outside the class -> Another function
def greet(Person):
    print('Hi my name is',Person.name,'and I am a',Person.gender)
#     p1 = Person('ankit','male')
#     return p1

p = Person('nitish','male') # creating object of class Person
x = greet(p) # passing greet function the object(refernece) of class Person. This
# "Hi my name is nitish and I am a male" as ref (p) passed to greet function.
```

```
Hi my name is nitish and I am a male
```

In [15]:
```
## Important point to be noted - in the above code we passed ___Person__ and in t
## output is same because we are passing reference (address) of the object indire
class Person:

    def __init__(self,name,gender):
        self.name = name
        self.gender = gender

# outside the class -> Another function
def greet(p):
    print('Hi my name is',p.name,'and I am a',p.gender)
p = Person('nitish','male') # creating object of class Person
x = greet(p)
```

```
Hi my name is nitish and I am a male
```

A function can take object as argument, or can retrun object as ouput:

In [16]:
```
class Person:

    def __init__(self,name,gender):
        self.name = name
        self.gender = gender

# outside the class -> function
def greet(person): ## taking object as argument
    print('Hi my name is',person.name,'and I am a',person.gender)
    p1 = Person('ankit','male') ## creating another object of calss inside the fu
    return p1

p = Person('nitish','male')
x = greet(p)
## Returning object creating inside the function
print(x.name)
print(x.gender)
```

```
Hi my name is nitish and I am a male
ankit
male
```

Mutability of objects

In technical programming terms, a mutable object is an object whose state can be modified after it is defined. And in python, objects are mutable.

We can make make them immutable, if we want, but by default they are mutable

In [18]:
```
class Person:

    def __init__(self,name,gender):
        self.name = name
        self.gender = gender

# outside the class -> Another function
def greet(person):
    person.name = 'ankit' ## changing the object value - mutating the object, pro
    return person

## the two ids will be same, as mutable objects points to the same memory address
p = Person('nitish','male')
print(id(p))
p1 = greet(p)
print(id(p1))
```

```
2809476694944
2809476694944
```

Instance Variable

An instance variable is a varibale whose value is different for differnt objects created of the class.

In [23]:
```
## Creating two objects of the same class, changes the values of the varibales, h
class Person:
    def __init__(self,name,gender):
        self.name = name
        self.gender = gender
w1 = Person('Mayank', 20)
w2 = Person('Maya', 21)
```

In [24]:
```
w1.gender
```

Out[24]:
```
20
```

In [25]:
```
w1.name
```

Out[25]:
```
'Mayank'
```

In [26]:
```
w2.gender
```

Out[26]:
```
21
```

In [27]:
```
w2.name
```

Out[27]:
```
'Maya'
```

Encapsulation

Encapsulation is a mechanism of wrapping the data (attributes/instance variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

In simple terms, if the coder wants that attributes of the class and methods of the class should not be visible after creating the object of the class, the coder can encapsulate the attributes and methods.

Encapsulation can be done/implemented by applying __ **(double underscore) before the particular attribute/s and method/s.** for eg.__ **attribute_name**, __ **method_name**. This is known as making attributes and methods **private**.

It is always advisable/best parctice to atleast make the attribute/s private or encapsulate them, so that no other person can change the value of the attributes from outside the class.

But in case, it is required to do so, python provides the option of creating getter and setter method.

getter_method is used/created to see the value of the private attributes.

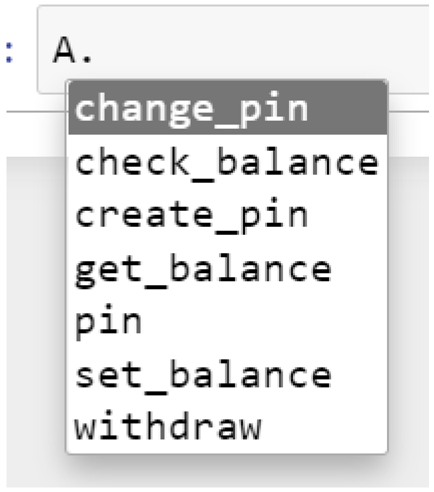
setter_method is used/created to set/change the value private attributes.

In [5]:
```
class Atm:

    # constructor(special function)->superpower ->
    def __init__(self):
        print(id(self))
        self.pin = ''
        self.__balance = 0 ## Making attribute private/encapsulating attribute
        #self.menu()

    def get_balance(self):## getter method
        return self.__balance

    def set_balance(self,new_value): # setter method
        if type(new_value) == int:
            self.__balance = new_value
        else:
            print('beta bahot maarenge')

    def __menu(self): ## making method private/encapsulating mathod
        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
        """)

        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:
            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')

    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')
        else:
            print('nai karne de sakta re baba')

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

    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)
            else:
                print('abe garib')
        else:
            print('sale chor')
```

In [6]:
```
A = Atm() ## created object
```

```
1601418736064
```

And hence now if we try to access the private method/attributes, they are not part of suggestions and no one would dream that class has those attribute/methods.


```
: A.
change_pin
check_balance
create_pin
get_balance
pin
set_balance
withdraw
```

In [7]:
```
## Important point to be noted: In case coder wants to change the value of attrib
## without using setter method, it can be done using the syntax " _class name__at
```

Instance variable v.s. Static variable

1. Instance variables are of object, whereas static varibale are of class.
2. Instance variable are declared inside the constructor, whereas static variable are declared outside all methods within the class
3. Intance variables are called using oject name (self), whereas static variable are called using calss name.

In [8]:
```
class static:

    name = 'Mayank' ## static varibale
    def __init__(self):
        self.name = 'Maya' ## instance variable
        print(self.name)
s = static()
```

```
Maya
```

In [9]:
```
static.name ## calling static varibale using class name
```

Out[9]:
```
'Mayank'
```

In [10]:
```
s.name ## calling intance varibale using object name
```

Out[10]:
```
'Maya'
```

In [22]:
```python
## more example:

class Atm:

    counter = 1 ## static variable

    # constructor(special function)->superpower ->
    def __init__(self):
        print(id(self))
        self.pin = ''
        self.__balance = 0
        ## static variable use inside the class
        self.cid = Atm.counter
        Atm.counter = Atm.counter + 1
        #self.menu()

    def set_balance(self,new_value):
        if type(new_value) == int:
            self.__balance = new_value
        else:
            print('beta bahot maarenge')

    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
        """)

        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:
            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')

    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')
        else:
            print('nai karne de sakta re baba')

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

    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)
            else:
                print('abe garib')
        else:
            print('sale chor')
```

In [23]:
```
c1 = Atm()
```

```
1601418937872
```

In [24]:
```
c2 = Atm()
```

```
1601418939744
```

In [25]:
```
c3 = Atm()
```

```
1601418946608
```

In [26]:
```
c1.cid
```

Out[26]:
```
1
```

In [27]:
```
c2.cid
```

Out[27]:
```
2
```

In [28]:
```
c3.cid
```

Out[28]:
```
3
```

```
In [29]: Atm.counter

Out[29]: 4
```

Applying concept of encapsulation to static variable

In [30]:
```
## more example:

class Atm:

    __counter = 1 ## naking static variable as private

    # constructor(special function)->superpower ->
    def __init__(self):
        print(id(self))
        self.pin = ''
        self.__balance = 0
        self.cid = Atm.__counter
        Atm.__counter = Atm.__counter + 1
        #self.menu()

    # utility functions
    @staticmethod ## decorator - it is known as class method that does not requir
    def get_counter():
        return Atm.__counter


    def get_balance(self):
        return self.__balance

    def set_balance(self,new_value):
        if type(new_value) == int:
            self.__balance = new_value
        else:
            print('beta bahot maarenge')

    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
        """)

        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:
            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')

    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')
        else:
            print('nai karne de sakta re baba')

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

    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)
            else:
                print('abe garib')
        else:
            print('sale chor')
```

In [ ]: