Python OOPS Classes and Objects

Everything in python is an object

Python OOPS gives programmer to wrote their own data types

- class - it is a blueprint of an object. when we create a variable of a class it is called object of that class
- object - it is an instance of class.

In [11]:
```
# syntax to create an object
# objectname = classname()
```

In [13]:
```
# object literal
L = [1,2,3]
# can create object in this way also
L.append(4) # calling method of List class by its object L
L
```

Out[13]:
```
[1, 2, 3, 4]
```

- class : can be built in or user defined
  - Data or property
  - Functions

In [27]:
```
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. 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')
  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)
    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')
    self.menu()
```

In [18]:
```
object=Atm()
```

```
2704975290272
pin created successfully
your balance is  4500
```

In [20]:
```
print(id(object))
```

```
2704975290272
```

Method vs Functions

- Method - functions created inside a class
- functions - created outside class and independent

Class Diagram - Atm class

In [ ]:
```
- +----------------------+
- |       Atm            |
- +----------------------+
- | - pin: str           |
- | - balance: int       |
- +----------------------+
- | + __ init __( )      |
- | + menu()             |
- | + create_pin( )      |
- | + change_pin( )      |
- | + check_balance( )   |
- | + withdraw( )        |
- +----------------------+

+  : public
- : private
```

In [43]:
```
L = [1,2,3]
len(L) # function ->bcos it is outside the list class
L.append(4)# method -> bcos it is inside the list class
```

Magic Methods/Dunder Methods

These are special methods which have some uniqueness . and they will be always in the form of __ nameofmethod __

- 1. Constructor - function that created inside a class using __ init __ and runs explicitly , whenver any object is being created .
- 2. __ str __ - it will print whatever in this method, and we can call it whenever we us print(classname)
- 3. __ add __ - whenver we use + between two object it get triggered and do whatever in the add method
- 4. __ sub __ - whenver we use - between two object it get triggered and do whatever in the add method
- 5. __ mul __ - whenver we use * between two object it get triggered and do whatever in the add method
- 6. __ truediv __ - whenver we use / between two object it get triggered and do whatever in the add method

1. __init__

- Created by itself on creating object of class
- can be use to trigger or execute something in code that does not need user touch
- we cannot change the name of constructor

- self : self is object itself.
- because all variables and function of class can only be accessed by class's object only.
- self helps to call one function by another function in a class rather using object explicitly

In [58]:
```
class Temp:

  def __init__(self):
      print(id(self))
      print('hello')

obj = Temp()
print(id(obj))
```

```
2704974771648
hello
2704974771648
```

In [69]:
```
class Fraction:

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

  def __str__(self):
    return '{}/{}'.format(self.num,self.den)

  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
```

In [71]:
```
fr1 = Fraction(3,4)
fr2 = Fraction(1,2)
```

2. __ str __

- whenever we use to print classname like - print(classname) then it will run str magic method that return whatever in str method

In [75]:
```
print(fr1,fr2)
```

```
3/4 1/2
```

3. __ add __
4. __ sub __
5. __ mul __

6. __ truediv __

- whenever we do + operation between two class it will get triggered and work accordingly

In [80]:
```
print(fr1 + fr2)
print(fr1 - fr2)
print(fr1 * fr2)
print(fr1 / fr2)
```

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

END