Python OOPS Encapsulation & Static keyword

Write OOP classes to handle the following scenarios:

- A user can create and view 2D coordinates
- A user can find out the distance between 2 coordinates
- A user can find find the distance of a coordinate from origin
- A user can check if a point lies on a given line
- A user can find the distance between a given 2D point and a given line

In [32]:
```
class Point:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def __str__(self):
        return f"({self.x},{self.y})"
    def distance(self,other):
        return (((self.x-other.x)**2)+((self.y-other.y)**2))**0.5
    def distance_from_origin(self):
        return ((self.x**2)+(self.y**2))**0.5



class Line:
    def __init__(self,A,B,C):
        self.A=A
        self.B=B
        self.C=C
    def __str__(self):
        return f"{self.A}X+ {self.B}Y+ {self.C}"
    def check(line,point):
        if line.A*point.x+line.B*point.y+line.C==0:
            print("point is on line")
```

```
        else:
            print("point does not lie on line")
    def distance(line,point):
        return abs(line.A*point.x + line.B*point.y + line.C)/(line.A**2 + line.B**2)**0.5
```

In [38]:
```
l1 = Line(1,1,-2)
p1 = Point(1,10)
print(l1)
print(p1)

print(l1.distance(p1))
l1.check(p1)
```

```
1X+ 1Y+ -2
(1,10)
6.363961030678928
point does not lie on line
```

How object access attributes

In [49]:
```
class Person:

  def __init__(self,name_input,country_input):
    self.name = name_input
    self.country = country_input

  def greet(self):
    if self.country == 'india':
      print('Namaste',self.name)
    else:
      print('Hello',self.name)
```

Once we create an object , the object has right to access its variables and methods

In [54]:
```
p=Person("Abhi","india")
```

In [60]:
```
# how to access attributes
```

```
p.name
```

Out[60]: 'Abhi'

In [62]:
```
# how to access methods
p.greet()
```

```
Namaste Abhi
```

In [ ]:
```
# what if i try to access non-existent attributes
# it will give error  - AttributeError: 'Person' object has no attribute 'gender'
p.gender
```

Attribute creation from outside the class

In [70]:
```
p.gender='male'
```

In [72]:
```
p.gender
```

Out[72]: 'male'

In [74]:
```
print(dir(p))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getstat
e__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce_
_', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'country', 'gende
r', 'greet', 'name']
```

In [76]:
```
class MyClass:
    def __init__(self):
        self.existing_attr = "I am defined inside the class."

# Create an instance of MyClass
obj1 = MyClass()

# Add an attribute outside the class definition
obj1.new_attr = "I am defined outside the class."

# Create another instance
```

```
obj2 = MyClass()

# Check attributes
print(obj1.existing_attr)  # Output: I am defined inside the class.
print(obj1.new_attr)       # Output: I am defined outside the class.
print(obj2.existing_attr)  # Output: I am defined inside the class.
print(hasattr(obj2, 'new_attr'))  # Output: False
```

```
I am defined inside the class.
I am defined outside the class.
I am defined inside the class.
False
```

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

In [81]:
```
# object without a reference
class Person:

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

p = Person()
q = p
# p is not an object but a variable that store ref. or address of person object
```

In [83]:
```
# Multiple ref
print(id(p))
print(id(q))
```

```
2476046715312
2476046715312
```

In [85]:
```
# change attribute value with the help of 2nd object
print(p.name)
print(q.name)
q.name = 'ankit'
print(q.name)
print(p.name)
```

```
nitish
nitish
ankit
ankit
```

Pass by reference

In [89]:
```
class Person:

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

# outside the class -> 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')
x = greet(p)
print(x.name)
print(x.gender)
```

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

In [91]:
```
class Person:

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

```
# outside the class -> function
def greet(person):
  print(id(person))
  person.name = 'ankit'
  print(person.name)

p = Person('nitish','male')
print(id(p))
greet(p)
print(p.name)
```

```
2474333760800
2474333760800
ankit
ankit
```

Object Mutability

Object of user defined class in python is mutable

In [96]:
```
class Person:

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

# outside the class -> function
def greet(person):
  person.name = 'ankit'
  return person

p = Person('nitish','male')
print(id(p))
p1 = greet(p)
print(id(p1))
```

```
2474341541952
2474341541952
```

Encapsulation

- Instance variable - variable whose value is different for every object
- To make any variable or methods private use __metho_name
- __balance , __checkbalance()
- In python, we can change value of a private variable from outside

- so we can change values of a variable in python from outside
- so in that case we can change it to private
- suppose we have a variable "balance" and we make it __balance
- but after making it priavte when we write obj.__balance and try to change the value then we are actually creating a new variable outside the class because once we create a var as private its name internally change to __classname__varname
- so now we can still change the value of that private var using __classname__varname
- so there is no proper way to protect it because python is made for adults

In [105…
```python
# instance var -> python tutor
class Person:

  def __init__(self,name_input,country_input):
    self.name = name_input
    self.country = country_input

p1 = Person('nitish','india')
p2 = Person('steve','australia')
```

In [109…
```python
print(p1.name,p2.name)
```

```
nitish steve
```

In [135…
```python
class Atm:

  # constructor(special function)->superpower ->
  def __init__(self):
    print(id(self))
```

```
    self.pin = ''
    self.__balance = 0
    self.__menu()

  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')
    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')

  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')

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

In [137…

```
obj=Atm()
```

```
2474341485600
pin created successfully
```

```
your balance is  3000
withdrawl successful.balance is 2000
your balance is  2000
```

Collection of objects

In [3]:
```python
# list of objects
class Person:

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

p1 = Person('nitish','male')
p2 = Person('ankit','male')
p3 = Person('ankita','female')

L = [p1,p2,p3]

for i in L:
  print(i.name,i.gender)
```

```
nitish male
ankit male
ankita female
```

In [9]:
```python
# dict of objects
# list of objects
class Person:

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

p1 = Person('nitish','male')
p2 = Person('ankit','male')
p3 = Person('ankita','female')

d = {'p1':p1,'p2':p2,'p3':p3}
```

```
for i in d:
  print(i,d[i].gender)
```

```
p1 male
p2 male
p3 female
```

Static Variables Vs Instance Variables

- Static variables are class variables that will be same for every object
- instance variable is object specific

In [4]:
```
class Atm:

  __counter = 1

  # 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
  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')
  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')

  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')

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

In [6]:
```
c1 = Atm()
```

```
2323698470496
```

In [8]:
```
Atm.get_counter()
```

Out[8]: 2

In [10]:
```
c3 = Atm()
```

```
2325424292736
```

In [12]:
```
c3.cid
```

Out[12]: 2

In [18]:
```
Atm.get_counter()
```

Out[18]: 3

Static Methods

Points to remember about static

- Static attributes are created at class level.
- Static attributes are accessed using ClassName.
- Static attributes are object independent. We can access them without creating instance (object) of the class in which they are defined.
- The value stored in static attribute is shared between all instances(objects) of the class in which the static attribute is defined.

In [29]:
```
class Lion:
  __water_source="well in the circus"

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

  def drinks_water(self):
      print(self.__name,
      "drinks water from the",Lion.__water_source)

  @staticmethod
  def get_water_source():
      return Lion.__water_source

simba=Lion("Simba","Male")
simba.drinks_water()
print( "Water source of lions:",Lion.get_water_source())
```

```
Simba drinks water from the well in the circus
Water source of lions: well in the circus
```

END