Code for section "**Object Oriented Programming**"
of the course "Python in Depth" by Deepali Srivastava

Classes and Objects

```
class Person:
      pass

>>>id(Person)
>>>Person()
>>>p1 = Person()
>>>p2 = Person()
>>>type(p1)
>>>type(p2)
>>>id(p1)
>>>id(p2)
>>>p1
>>>p2

class Person:
     def display(self):
         print('I am a person')
     def greet(self):
         print('Hello, how are you doing?')

>>>p1.display()
>>>p1.greet()

>>>p2.display()
>>>p2.greet()

class Person:
      def display(self):
         print('I am a person', self)
      def greet(self):
         print('Hi, how are you doing ? ', self)
>>p1.name = 'Tom'
>>p1.name
>>p2.name

class Person:
      def  set_details(self):
            self.name = 'John'
            self.age = 20
      def display(self):
             print('I am a person', self)
      def greet(self):
             print('Hi, how are you doing ? ', self)
```

```
>>>p1.set_details()
>>>p2.set_details()

>>p1.name
>>p1.age
>>p2.name
>>p2.age

>>p2.name ='Jack'
>>p2.age = 30

>>>p2.name
>>>p2.age
>>>p1.name
>>>p2.age

>>>p1.set_details('Bob',20)
>>>p2.set_details('Ted',90)

>>p1.name
>>p2.name



class Person:
      def  set_details(self):
            self.name = 'John'
            self.age = 20

      def display(self):
         print('I am ', self.name)

      def greet(self):
            if self.age < 80:
                  print('Hi, How are you doing?')
            else:
                  print('Hello, How do you do?')
            self.display()
```

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

      def display(self):
            print('I am ', self.name)

      def greet(self):
            if self.age < 80:
                  print('Hi, How are you doing ?')
            else:
                  print('Hello, How do you do ?')
            self.display()

>>>p1=Person()
>>>p1.set_details('Bob', 20)
>>>p1.greet()
>>>p2=Person()
>>>p2.set_details('Ted', 90)
>>>p2.greet()
```

Classes and Objects …. Continued

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

    def display(self):
         print('I am',self.name)

    def greet(self):
        if self.age < 80:
            print('Hello, how are you doing?')
        else:
            print('Hello, How do you do ?')
        self.display()

   def get_old(self):
        age = 75


p1 = Person()
p2 = Person()

p1.set_details('John',20)
p2.set_details('Jack',90)
```

```
p1.greet()
p2.greet()

p1.get_old()
```

Initializer Method

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

    def display(self):
         print('I am',self.name)

    def greet(self):
        if self.age < 80:
            print('Hello, how are you doing?')
        else:
            print('Hello, How do you do ?')
p1=Person('John',20)
p2=Person('Jack',90)

p1.display()
p1.greet()

p2.display()
p2.greet()
```

Data Hiding

```
class Product:
    def __init__(self):
        self.data1 = 10
        self._data2 = 20

    def methodA(self):
        pass

    def _methodB(self):
        pass

>>>p = Product()
>>>p.data1
>>>p.methodA()
>>>p._data2
>>>p._methodB()
```

```
class Product:
    def __init__(self):
        self.data1 = 10
        self.__data2 = 20

    def methodA(self):
        pass

    def __methodB(self):
        pass

>>>p = Product()
>>>p.__data2
>>>p.__methodB()
>>>p._Product__data2
>>>p._Product__methodB()
```

Property

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)

    @property
    def value(self):
          return self._x

    @value.setter
    def value(self, val):
         self._x = val

    @property
     def y(self):
         return self._y

    @y.setter
     def y(self, val):
         self._y = val



>>>p = Product(12,24)
>>>p.value
```

```
>>>p.value + 2
>>>dir(Product)
>>>p.value = 10
>>>p.value = 20
>>>p.y
>>>p.y = 12


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

    def display(self):
        print(self.name,self.age)

if __name__ == '__main__':
   p = Person('Raj', 30)
   p.age = 100
   p.display()


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

   def display(self):
         print(self.name,self._age)

   def set_age(self, new_age):
       if  20 <new_age< 80:
           self._age = new_age
       else:
           raise ValueError('Age must be between 20 and 80')

   def get_age(self):
       return self._age

>>>p.set_age(100)
>>>p.set_age(12)
>>>p.set_age(25)
>>>p.display()
>>>p.set_age( p.get_age() + 1 )
>>>p.display()
>>>p1 = Person('Dev', 200)
>>p1.display()


    class Person:
```

```
        def __init__(self, name, age):
             self.name = name
             if  20 < age < 80:
                self._age = age
             else:
                 raise ValueError('Age must be between 20 and 80')

        def display(self):
              print(self.name,self._age)

        def set_age(self, new_age):
              if  20 <new_age< 80:
                   self._age = new_age
              else:
                  raise ValueError('Age must be between 20 and 80')

        def get_age(self):
              return self._age


>>>p1 = Person('Dev', 200)

from person import Person
p = Person('Peter', 30)
p.age = 100
print(p.age)


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

        def display(self):
              print(self.name,self.age)

        @property
        def age(self):
            return self._age

        @age.setter
        def age(self, new_age):
             if  20< new_age<80:
               self._age = new_age
             else:
                raise ValueError('Age must be between 20 and 80')


>>p.age
>>p.age = 30
```

```
>>p.age = 200
>>p1 = Person('Dev',200)
>>p.age = p.age +1
>>p.age += 1

class Employee:
      def __init__(self, name, password, salary):
            self._name =  name
            self._password = password
            self._salary = salary

      @property
      def name(self):
           return self._name

      @property
      def password(self):
           raise AttributeError('password not readable')

      @password.setter
      def password(self, new_password):
              self._password =  new_password

      @property
      def salary(self):
           return self._salary

      @salary.setter
      def salary(self, new_salary):
              self._password =  new_salary


>>> e = Employee('Jill', 'feb31', 5000)
>>> e.name
>>>e.name = 'dd'
>>> e.password
>>> e.password = 'feb29'
>>> e.salary
>>> e.salary = 6000
```

```
class Rectangle():
    def __init__(self,length,breadth):
           self.length = length
           self.breadth = breadth
           self.diagonal = (self.length*self.length + self.breadth *
self.breadth)**0.5

    def area(self):
          return self.length * self.breadth

    def perimeter(self):
          return 2*(self.length + self.breadth)

>>>r = Rectangle(2,5)
>>>r.diagonal
>>>r.area()
>>r.perimeter()

>>>r.length = 10
>>>r.diagonal
>>>r.area()
>>>r.perimeter()


class Rectangle():
    def __init__(self,length,breadth):
           self.length = length
           self.breadth = breadth

    def area(self):
          return self.length * self.breadth

    def perimeter(self):
          return 2*(self.length + self.breadth)

    @property
     def diagonal(self):
          return (self.length*self.length + self.breadth *
self.breadth)**0.5

>>>r = Rectangle(2,5)
>>>r.diagonal
>>>r.length = 10
>>>r.diagonal
```

```
class Product:
    def __init__(self,x,y):
        self._x = x
        self._y = y

    def display(self):
        print(self._x, self._y)

    @property
    def value(self):
          return self._x

    @value.setter
    def value(self, val):
         self._x = val

    @value.deleter
    def value(self):
          print('value deleted')

    @property
     def y(self):
         return self._y

    @y.setter
     def y(self, val):
         self._y = val

>>>p = Product(12,24)
>>>del p.value
```

Class Variables

```
class Person:
    species  = 'Homo sapiens'
    def __init__(self,name,age):
        self.name = name
        self.age = age

    def display(self):
         print(f'{self.name} is {self.age} years old')

p1 = Person('John',20)
p2 = Person('Jack',34)
p1.display()
p2.display()
```

```
>>>Person.species
>>>p1.species
>>>p2.species
>>>Person.name
>>>id(p1.species)
>>>id(p2.species)
>>>d(Person.species)
>>>print(f'{self.name} is {self.age} years old {Person.species}')

class Person:
    species  = 'Homo sapiens'
    count = 0

    def __init__(self,name,age):
        self.name = name
        self.age = age
        Person.count+=1

    def display(self):
         print(f'{self.name} is {self.age} years old')

p1 = Person('John',20)
p2 = Person('Jack',34)

p1.display()
p2.display()

>>>Person.count
>>>p3=Person('Jill', 40)
>>> p4=Person('Jane', 35)
>>>Person.count


class BankAccount:
    rate_of_interest = 5
    min_balance = 100
    min_balance_fees = 10

    def __init__(self,account_number, owner_name, balance):
        self.account_number = account_number
        self.owner_name = owner_name
        self.balance = balance

    def withdraw(self,amount):
        self.balance -= amount

    def deposit(self,amount):
        self.balance += amount
```

```
account1 = BankAccount('7348', 'Tom', 50)
account2 = BankAccount('6378', 'Bob', 400)


class Book():
       x = 5
       def __init__(self):
            self.x = 100
       def display(self):
           print(self.x)
           print(Book.x)

b = Book()
b.display()

>>>Book.x
>>>b.x

class Book():
       x = 5
       def __init__(self):
            self.x = 100
       display(self):
           print(self.x)
           print(Book.x)
b = Book()

class Account:
   rate = 5

a1 = Account()
a2 = Account()

>>>Account.rate
>>>a1.rate
>>>a2.rate

>>>Account.rate = 6
>>>Account.rate
>>>a1.rate
>>>a2.rate
>>>a1.rate = 7
>>>Account.rate
>>>a1.rate
>>>a2.rate
>>>id(Account.rate )
>>>id(a1.rate)
>>>id(a2.rate)
```

```
class Account():
       rate = 5
       def some_method(self):
              print(self.rate, Account.rate, id(self.rate),
id(Account.rate))
              self.rate = 10
              print(self.rate, Account.rate, id(self.rate),
id(Account.rate))

a1 = Account()
a2 = Account()
a1.some_method()
```

Class Methods

```
class MyClass():
     a = 5
     def  __init__(self, x):
          self._x = x

     def method1(self):
          print(self.x)

     @classmethod
     def method2(cls):
          print(cls.a)

>>>MyClass.method2()


 class Person:
    species  = 'Homo sapiens'
    count = 0
    def __init__(self,name,age):
        self.name = name
        self.age = age
        Person.count += 1

    def display(self):
         print(f'{self.name} is {self.age} years old')

    @classmethod
    def show_count(cls):
        print(f'There are {cls.count} {cls.species}')

>>>Person.show_count()
>>>p1 = Person('John',20)
>>>p2 = Person('Jack',34)
```

```
>>>Person.show_count()

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

    def display(self):
         print('I am', self.name, self.age, 'years old')

p1 = Person('John',20)
p2 = Person('Jack',34)

s = 'Jim, 23'
d = {'name': 'Jane', 'age':34}


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

    @classmethod
    def from_str(cls,s):
        name,age = s.split(',')
        return cls(name, int(age))

    @classmethod
    def from_dict(cls,d):
        return cls( d['name'], d['age'] )

    def display(self):
         print('I am', self.name, self.age, 'years old')

p1 = Person('John', 20)
p2 = Person('Jim', 35)

s = 'Jack, 23'
d = {'name': 'Jane', 'age':34}

p3 = Person.from_str(s)
p3.display()

p4 = Person.from_dict(d)
p4.display()
```

```
class Employee:
    def __init__(self,first_name, last_name,
name,birth_year,salary):
        self.first_name = first_name
        self.last_name = last_name
        self.birth_year = birth_year
        self.salary = salary

    def show(self):
         print(f'I am {self.first_name} {self.last_name} born in
{self.birth_year}')


from employee import Employee
from datetime import datetime

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

    @classmethod
    def from_str(cls,s):
        name,age = s.split(',')
        return cls(name, int(age))

    @classmethod
    def from_dict(cls,d):
        return cls( d['name'], d['age'] )

    @classmethod
    def from_employee(cls,emp):
        name = emp.first_name + ' ' + emp.last_name
        age = datetime.today().year - emp.birth_year
        return cls(name,age)

    def display(self):
         print('I am', self.name, self.age, 'years old')

e1 = Employee('James', 'Smith', 1990)

p1 = Person.from_employee(e1)
p1.display()
```

Static Methods

```
class MyClass():

     a = 5
     def  __init__(self, x):
          self.x = x

     def method1(self):
          print(self.x)

     @classmethod
     def method2(cls):
          print(cls.a)

     @staticmethod
     def method3(m,n):
          retrun m+n
```

Magic Methods - 1

```
class Fraction:
    def __init__(self,nr,dr=1):
        self.nr = nr
        self.dr = dr
        if self.dr < 0:
            self.nr *= -1
            self.dr *= -1
        self._reduce()

    def show(self):
        print(f'{self.nr}/{self.dr}')

    def add(self,other):
        if isinstance(other,int):
            other = Fraction(other)
        f = Fraction(self.nr * other.dr + other.nr * self.dr,
self.dr * other.dr)
        f._reduce()
        return f

    def multiply(self,other):
        if isinstance(other,int):
            other = Fraction(other)
        f = Fraction(self.nr * other.nr , self.dr * other.dr)
        f._reduce()
        return f
```

```
    def _reduce(self):
        h = Fraction.hcf(self.nr, self.dr)
        if h == 0:
            return

        self.nr //= h
        self.dr //= h

    @staticmethod
    def hcf(x,y):
        x=abs(x)
        y=abs(y)
        smaller = y if x>y else x
        s = smaller
        while s>0:
            if x%s==0 and y%s==0:
                break
            s-=1
        return s



>>>f1 = Fraction(2,3)
>>>f2 = Fraction(3,4)

>>>f3 = f1+f2

>>>f3 = f1*f2


>>>f3 = f1.add(f2)
>>>f3.show()

>>>f3 = f1.multiply(f2)

class Fraction:
    def __init__(self,nr,dr=1):
        self.nr = nr
        self.dr = dr
        if self.dr < 0:
            self.nr *= -1
            self.dr *= -1
        self._reduce()

    def show(self):
        print(f'{self.nr}/{self.dr}')
```

```
    def __add__(self,other):
        if isinstance(other,int):
            other = Fraction(other)
        f = Fraction(self.nr * other.dr + other.nr * self.dr,
self.dr * other.dr)
        f._reduce()
        return f

    def __sub__(self,other):
        if isinstance(other,int):
            other = Fraction(other)
        f = Fraction(self.nr * other.dr - other.nr * self.dr,
self.dr * other.dr)
        f._reduce()
        return f

    def __mul__(self,other):
        if isinstance(other,int):
            other = Fraction(other)
        f = Fraction(self.nr * other.nr , self.dr * other.dr)
        f._reduce()
        return f

    def _reduce(self):
        h = Fraction.hcf(self.nr, self.dr)
        if h == 0:
            return

        self.nr //= h
        self.dr //= h

    @staticmethod
    def hcf(x,y):
        x=abs(x)
        y=abs(y)
        smaller = y if x>y else x
        s = smaller
        while s>0:
            if x%s==0 and y%s==0:
                break
            s-=1
        return s

>>>f3 = f1.__add__(f2)
>>>f3
>>>f3 = f1 + f2
>>>f3
>>>f3 = f1-f2
>>>f3
>>>f3 = f1*f2
```

```
>>f3
>>f3 = f1-2
```

Magic Methods - 2

```
class Fraction:
    def __init__(self,nr,dr=1):
        self.nr = nr
        self.dr = dr
        if self.dr < 0:
            self.nr *= -1
            self.dr *= -1
        self._reduce()

    def show(self):
        print(f'{self.nr}/{self.dr}')

    def add(self,other):
        if isinstance(other,int):
            other = Fraction(other)
        f = Fraction(self.nr * other.dr + other.nr * self.dr,
self.dr * other.dr)
        f._reduce()
        return f

    def multiply(self,other):
        if isinstance(other,int):
            other = Fraction(other)
        f = Fraction(self.nr * other.nr , self.dr * other.dr)
        f._reduce()
        return f

    def __eq__(self,other):
          return (self.nr * other.dr) == (self.dr * other.nr)

    def __lt__(self,other):
          return (self.nr * other.dr) < (self.dr * other.nr)

    def __le__(self,other):
          return (self.nr * other.dr) <= (self.dr * other.nr)

    def __str__(self):
        return f'{self.nr}/{self.dr}'

    def __repr__(self):
        return f'Fraction({self.nr},{self.dr})'

    def _reduce(self):
        h = Fraction.hcf(self.nr, self.dr)
```

```
        if h == 0:
            return

        self.nr //= h
        self.dr //= h

    @staticmethod
    def hcf(x,y):
        x=abs(x)
        y=abs(y)
        smaller = y if x>y else x
        s = smaller
        while s>0:
            if x%s==0 and y%s==0:
                break
            s-=1
        return s
```

```
>>>f1 = Fraction(2,3)
>>>f2 = Fraction(2,3)
>>>f3 = Fraction(4,6)
>>>f1 == f2
>>>f1 == f3
>>>f1 != f2
>>>f1 = Fraction(2,3)
>>>f2 = Fraction(2,3)
>>>f3 = Fraction(1,5)
>>>f1 <  f2
>>>f1 <= f2
>>>f1 < f3
>>>f3 < f1
>>>str(f1)
>>>f1
>>>f1 = Fraction(3,4)
>>>f2 = Fraction(4,5)
>>>f3 = Fraction(1,5)
>>>L = [f1,f2,f3]
>>>print(L)
```

Magic Methods - 3

```
def __radd__(self,other):
      return self.__add__(other)
```

```
>>f2 = f1+3
>>f2 = 3 + f1
```

Inheritance

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

    def greet(self):
        print('Hello I am', self.name)

    def is_adult(self):
        if self.age > 18:
            return True
        else:
            return False

    def contact_details(self):
        print(self.address, self.phone)


class Employee(Person):
   pass


emp = Employee('Jack', 30, 'D4, XYZ Street, Delhi', '994477291')

>>>emp.name
>>>emp.age
>>>emp.address
>>>emp.phone

>>>emp.greet()
>>>emp.is_adult()
>>>emp.contact_details()

>>>isinstance(emp,Employee)  true
>>>isinstance(emp, Person)  true

>>>is subclass(Employee, Person)
>>>is subclass(Person, object)
>>>is subclass(str, object)
>>>is subclass(int, object)
```

```
class Employee(Person):
    def __init__(self, name, age, address, phone, salary,
office_address, office_phone):
        super().__init__(name, age, address, phone)
        self.salary = salary
        self.office_address = office_address
        self.office_phone = office_phone

    def calculate_tax(self):
        if self.salary < 5000:
            return 0
        else:
            return self.salary * 0.05

    def contact_details(self):
        super().contact_details()
        print(self.office_address, self.office_phone)

emp = Employee('Jack', 30, 'D4, XYZ Street', '994477291', 8000, 'ABC
Street', '384923993')
emp.contact_details()
```

Multiple Inheritance

```
class Teacher:
    def greet(self):
        print('I am a Teacher')

class Student:
    def greet(self):
        print('I am a Student')

class TeachingAssistant(Student, Teacher):
     def greet(self):
        print('I am a Teaching Assistant')

x = TeachingAssistant()
x.greet()

>>>TeachingAssistant.__bases__

class Person:
    def greet(self):
        print('I am a Person')
```

```
class Teacher(Person):
    def greet(self):
        print('I am a Teacher')

class Student(Person):
    def greet(self):
        print('I am a Student')

class TeachingAssistant(Student, Teacher):
    def greet(self):
        print('I am a Teaching Assistant')

x = TeachingAssistant()
x.greet()

>>> help(TeachingAssistant)
>>>TeachingAssistant.__mro__
>>> TeachingAssistant.mro()
>>> x.__class__.__mro__
```

MRO and super()

```
class Person:
    def greet(self):
        print('I am a Person')

class Teacher(Person):
    def greet(self):
        Person.greet(self)
        print('I am a Teacher')

class Student(Person):
    def greet(self):
        Person.greet(self)
        print('I am a Student')

class TeachingAssistant(Student, Teacher):
     def greet(self):
         Student.greet(self)
         Teacher.greet(self)
         print('I am a Teaching Assistant')

x = TeachingAssistant()
x.greet()
```

```
class Person:
    def greet(self):
        print('I am a Person')

class Teacher(Person):
    def greet(self):
        super().greet()
        print('I am a Teacher')

class Student(Person):
    def greet(self):
        super().greet()
        print('I am a Student')

class TeachingAssistant(Student, Teacher):
     def greet(self):
         super().greet()
         print('I am a Teaching Assistant')

x = TeachingAssistant()
x.greet()

>>>help(TeachingAssistant)
>>>s = Student()
>>>s.greet()
```

Polymorphism

```
class Car:
   def start(self):
       print('Engine started')
   def move(self):
       print('Car is running')
   def stop(self):
       print('Brakes applied')

class Clock:
   def move(self):
       print('Tick Tick Tick')
   def stop(self):
       print('Clock needles stopped')
```

```
class Person:
    def move(self):
        print('Person walking')

    def stop(self):
        print('Taking rest')

    def talk(self):
        print('Hello')

car = Car()
clock = Clock()
person = Person()

def do_something(x):
   x.move()
   x.stop()

>>do_something(car)
>>do_something(clock)
>>do_something(person)

class Rectangle:
    name = 'Rectangle'
    def __init__(self, length, breadth):
        self.length = length
        self.breadth =  breadth

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

class Triangle:
    name = 'Triangle'
    def __init__(self, s1, s2, s3):
        self.s1 = s1
        self.s2 = s2
        self.s3 = s3

    def area(self):
        sp = (self.s1 + self.s2 + self.s3) / 2
        return ( sp*(sp-self.s1)*(sp-self.s2)*(sp-self.s3) ) ** 0.5

    def perimeter(self):
        return self.s1 + self.s2 + self.s3
```

```
class Circle:
    name = 'Circle'
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

    def perimeter(self):
        return 2 * 3.14 * self.radius

r1 = Rectangle(13,25)
r2 = Rectangle(14,16)
t1 = Triangle(14,17,12)
t2 = Triangle(25,33,52)
c1 = Circle(14)
c2 = Circle(25)

def find_area_perimeter(shape):
    print(shape.name)
    print('Area : ', shape.area() )
    print('Perimeter : ', shape.perimeter() )

>>>find_area_perimeter(t2)
>>>find_area_perimeter(c1)
>>>find_area_perimeter(r2)


shapes = [r1,r2,t1,t2,c1,c2]

total_area = 0
total_perimeter = 0

for shape in shapes:
    total_area += shape.area()
    total_perimeter += shape.perimeter()

print(total_area, total_perimeter)
```