(1) Implementing a New Class

In [ ]:
```
'''
Q1: Implement a class "Circle" that represents circles. The class should support the fo

setSize(diameter): Takes a numeric value as input ans sets the diameter of the circle
perimeter(): Returns the perimeter of the circle
area(): Returns the area of the circle

>>> circle = Circle()
>>> circle.setSize(20)
>>> circle.perimeter()
62.83185307179586
>>> circle.area()
314.1592653589793
'''
```

In [1]:
```
class Circle:

    def setSize(self, diameter):
        self.diam = diameter

    def perimeter(self):
        import math
        return math.pi * self.diam

    def area(self):
        import math
        return math.pi * (self.diam / 2) ** 2
```

In [2]:
```
circle = Circle()
```

In [3]:
```
circle.setSize(20)
```

In [4]:
```
circle.perimeter()
```

Out[4]:
```
62.83185307179586
```

In [5]:
```
circle.area()
```

Out[5]:
```
314.1592653589793
```

In [ ]:
```
'''
Q2: Implement a class "BankAccount" that supports the following methods:

setbalance(amount): Takes an amount as input and set the initial balance
deposit(amount): Takes an amount as input and adds it to the balance
withdraw(amount): Takes an amount as input and withdraws it from the balance
```

```
balance(): Returns the balance of the account

>>> account = BankAccount()
>>> account.setbalance(30)
>>> account.deposit(100)
>>> account.withdraw(30)
>>> account.balance()
100
'''
```

In [6]:
```
class BankAccount:

    def setbalance(self, amount1):
        self.balance1 = amount1

    def deposit(self, amount2):
        self.balance2 = self.balance1 + amount2

    def withdraw(self, amount3):
        self.balance3 = self.balance2 - amount3

    def balance(self):
        return self.balance3
```

In [7]:
```
account = BankAccount()
```

In [8]:
```
account.setbalance(30)
```

In [9]:
```
account.deposit(100)
```

In [10]:
```
account.withdraw(30)
```

In [11]:
```
account.balance()
```

Out[11]:
```
100
```

In [ ]:
```
'''
Q3: Implement a class "Employee" that support methods:

name(firstname): Takes one's first name as input
rate(payrate): Takes one's hourly pay rate as input
changeRate(newpayrate): Takes the new pay rate as input and changes the employee's pay
pay(hours): Takes the number of hours worked as input and returns total payment (i.e. o

>>> e1 = Employee()
>>> e1.name('Jeff')
>>> e1.rate(10)
>>> e1.changeRate(20)
>>> e1.pay(20)
400
'''
```

In [12]:
```
class Employee:

    def name(self, firstname):
        self.nme = firstname

    def rate(self, payrate):
        self.prate = payrate

    def changeRate(self, newpayrate):
        self.nprate = newpayrate

    def pay(self, hours):
        return self.nprate * hours
```

In [13]:
```
e1 = Employee()
```

In [14]:
```
e1.name('Jeff')
```

In [15]:
```
e1.rate(10)
```

In [16]:
```
e1.changeRate(20)
```

In [17]:
```
e1.pay(20)
```

Out[17]: 400

(2) Overloaded Class

In [ ]:
```
'''
Q4: Modify the class "Circle" of Q1 to perform the following:

>>> c1 = Circle(20)
>>> c1.perimeter()
62.83185307179586
>>> c1.area()
314.1592653589793
>>> c2 = Circle()
>>> c2.perimeter()
0.0
>>> c2.area()
0.0
'''
```

In [18]:
```
class Circle:

    def __init__(self, number = 0):
        self.num = number
```

```
    def perimeter(self):
        import math
        return math.pi * self.num

    def area(self):
        import math
        return math.pi * (self.num / 2) ** 2
```

In [19]:
```
c1 = Circle(20)
```

In [20]:
```
c1.perimeter()
```

Out[20]: 62.83185307179586

In [21]:
```
c1.area()
```

Out[21]: 314.1592653589793

In [22]:
```
c2 = Circle()
```

In [23]:
```
c2.perimeter()
```

Out[23]: 0.0

In [24]:
```
c2.area()
```

Out[24]: 0.0

In [ ]:
```
'''
Q5: Modify the class BankAccount of Q2 to perform followings:

>>> a1 = BankAccount(200)
>>> a2 = BankAccount(100)
>>> a1 + a2
300
>>> a1 - a2
100
'''
```

In [25]:
```
class BankAccount:

    def __init__(self, number):
        self.num = number

    def setbalance(self):
        self.balance1 = self.num

    def deposit(self, amount2):
```

```
        self.balance2 = self.balance1 + amount2

    def withdraw(self, amount3):
        self.balance3 = self.balance2 - amount3

    def balance(self):
        return self.balance3
```

In [26]:
```
a1 = BankAccount(200)
```

In [27]:
```
a2 = BankAccount(100)
```

In [30]:
```
a1 + a2
```

```
300
```

In [31]:
```
a1 - a2
```

```
100
```

In [ ]:
```
'''
Q6: Modify the class "Employee" of Q3 to perform followings:

>>> e1 = Employee('Jeff', 20)
>>> e2 = Employee('Grace', 30)
>>> e1.pay()
20
>>> e2.pay()
30
>>> e2.changeRate(20)
>>> e1 == e2
True
'''
```

In [32]:
```
class Employee:

    def __init__(self, firstname, rate):
        self.nme = firstname
        self.r = rate

    def pay(self):
        return self.r

    def changeRate(self, newpayrate):
        self.nprate = newpayrate
        self.r = self.nprate

    def __eq__(self, point):
        return self.r == point.r
```

In [33]:
```
e1 = Employee('Jeff', 20)
```

In [34]:
```
e2 = Employee('Grace', 30)
```

In [35]:
```
e1.pay()
```

Out[35]: 20

In [36]:
```
e2.pay()
```

Out[36]: 30

In [37]:
```
e2.changeRate(20)
```

In [38]:
```
e1 == e2
```

Out[38]: True

(3) Inheritance

In [ ]:
```
'''
Q7: Develop a class "Employee2" as a subclass of "Employee".
    The "Employee2" should overide the inherited operator == to compare firstnames of er

>>> e1 = Employee2('Jeff', 20)
>>> e2 = Employee2('Jeff', 30)
>>> e1 == e2
True
'''
```

In [39]:
```
class Employee2(Employee):
    def __eq__(self, point):
        return self.nme == point.nme
```

In [40]:
```
e1 = Employee2('Jeff', 20)
```

In [41]:
```
e2 = Employee2('Jeff', 30)
```

In [42]:
```
e1 == e2
```

Out[42]: True

In [ ]:
```
'''
Q8: Develop a class "Employee3" as a subclass of "Employee".
    The "Employee3" overloads the inherited operator == to compare firstnames and payra
```

```
>>> e1 = Employee3('Jeff', 20)
>>> e2 = Employee3('Jeff', 30)
>>> e1 == e2
False
>>> e3 = Employee3('Jeff', 20)
>>> e4 = Employee3('Grace', 20)
>>> e3 == e4
False
>>> e5 = Employee3('Jeff', 20)
>>> e6 = Employee3('Jeff', 20)
>>> e5 == e6
True
'''
```

In [43]:
```
class Employee3(Employee):
    def __eq__(self, point):
        return self.nme == point.nme and self.r == point.r
```

In [44]:
```
e1 = Employee3('Jeff', 20)
```

In [45]:
```
e2 = Employee3('Jeff', 30)
```

In [46]:
```
e1 == e2
```

Out[46]: False

In [47]:
```
e3 = Employee3('Jeff', 20)
```

In [48]:
```
e4 = Employee3('Grace', 20)
```

In [49]:
```
e3 == e4
```

Out[49]: False

In [50]:
```
e5 = Employee3('Jeff', 20)
```

In [51]:
```
e6 = Employee3('Jeff', 20)
```

In [52]:
```
e5 == e6
```

Out[52]: True

In [ ]:
```
'''
Q9: Develop a container class "Statistic" that stores a sequence of numbers and
```

```
    provides statistical information about the numbers. It should support an overloaded
    initializes the container and the methods shown below:

>>> s1 = Statistic()
>>> s1.add(2)   # Adds 2 to the "Statistic" container
>>> s1.add(4)
>>> s1.add(6)
>>> s1.add(8)
>>> s1.min()
2
>>> s1.max()
8
'''
```

In [53]:
```
class Statistic:
    def __init__ (self):
        self.lst = []
    def add(self, item):
        self.lst.append(item)
    def min(self):
        return min(self.lst)
    def max(self):
        return max(self.lst)
```

In [54]:
```
s1 = Statistic()
```

In [55]:
```
s1.add(2)
```

In [56]:
```
s1.add(4)
```

In [57]:
```
s1.add(6)
```

In [58]:
```
s1.add(8)
```

In [59]:
```
s1.min()
```

Out[59]: 2

In [60]:
```
s1.max()
```

Out[60]: 8

In [ ]:
```
'''
Q10: Develop a class "AdvancedStat" as a subclass of "Statistic".
     "AdvancedStat" should support the following methods shown below:

>>> s2 = AdvancedStat([2, 4, 6, 8])
>>> s2.add(10)
```

```
>>> s2.min()
2
>>> s2.max()
10
>>> s2.sum()
30
>>> s2.mean()
6.0

'''
```

In [61]:
```
class AdvancedStat(Statistic):
    def add(self):
        self.lst.add()
    def min(self):
        return min(self.lst)
    def max(self):
        return max(self.lst)
    def sum(self):
        return sum(self.lst)
    def mean(self):
        return sum(self.lst) / len(self.lst)
```

In [ ]:
```
s2 = AdvancedStat([2, 4, 6, 8])
```

In [ ]:
```
s2.add(10)
```

In [65]:
```
s2.min()
```

```
2
```

In [72]:
```
s2.max()
```

Out[72]:
```
10
```

In [73]:
```
s2.sum()
```

Out[73]:
```
30
```

In [77]:
```
s2.mean()
```

Out[77]:
```
6.0
```