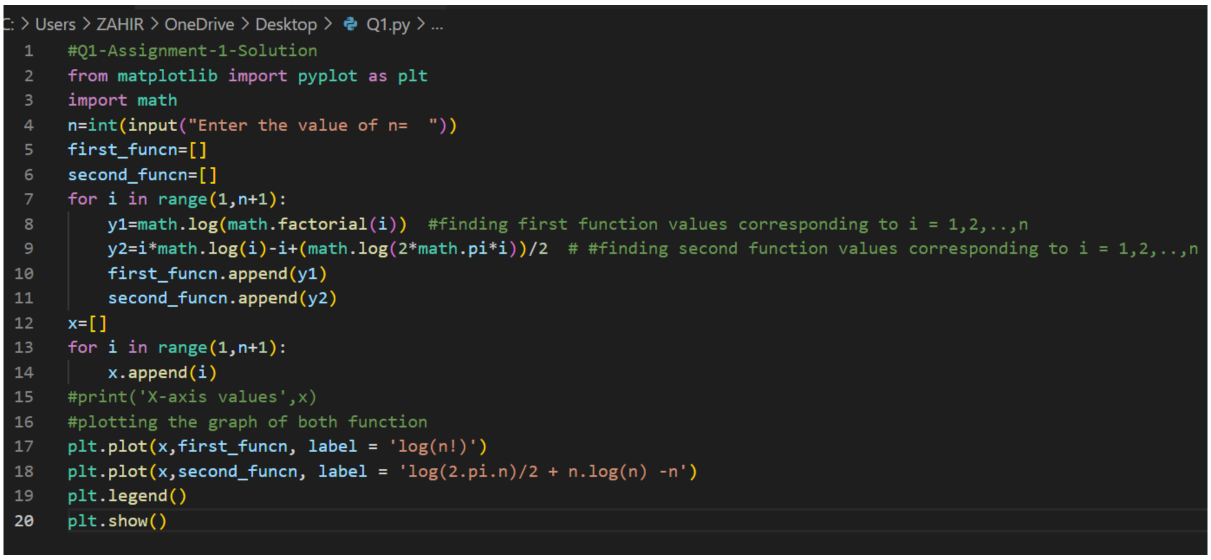
Coding Assignment-1 (codes, output and explainations)

Name :- Zahir Khan.   Roll no & Course:- 112202010 -MTech CaM.

Q.1) Solution:-



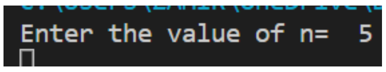
C: > Users > ZAHIR > OneDrive > Desktop > Q1.py > ...

```python
#Q1-Assignment-1-Solution
from matplotlib import pyplot as plt
import math
n=int(input("Enter the value of n=  "))
first_funcn=[]
second_funcn=[]
for i in range(1,n+1):
    y1=math.log(math.factorial(i))  #finding first function values corresponding to i = 1,2,..,n
    y2=i*math.log(i)-i+(math.log(2*math.pi*i))/2  # #finding second function values corresponding to i = 1,2,..,n
    first_funcn.append(y1)
    second_funcn.append(y2)
x=[]
for i in range(1,n+1):
    x.append(i)
#print('X-axis values',x)
#plotting the graph of both function
plt.plot(x,first_funcn, label = 'log(n!)')
plt.plot(x,second_funcn, label = 'log(2.pi.n)/2 + n.log(n) -n')
plt.legend()
plt.show()
```

Steps and Explanations:

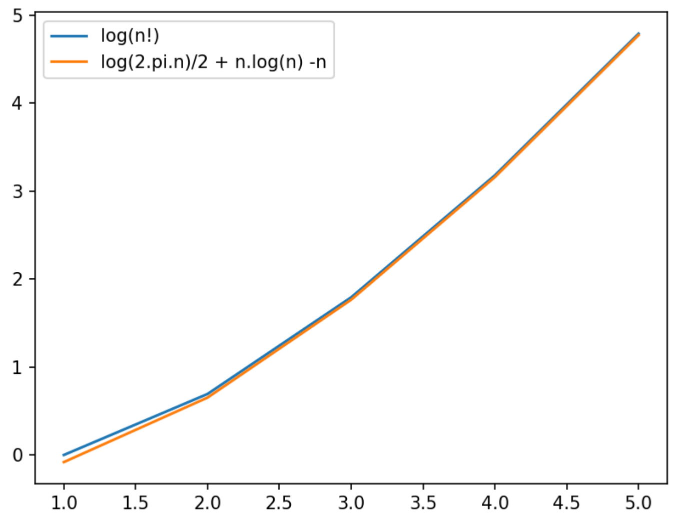
- We have imported ‘matplotlib’ library for plotting the functions in graph and imported ‘math’ for the mathematical expression of the Stirling’s Formula.
- To reduce the running time and the complexity, we have taken ‘log’ in both side of the Stirling’s approximation (n! ≈ $\sqrt{2\pi n}\,(\frac{n}{e})^n$). i.e, log n! ≈ (log 2πn)/2 + n log(n) -n.

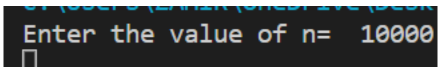
Output 1:



```
Enter the value of n=  5
```

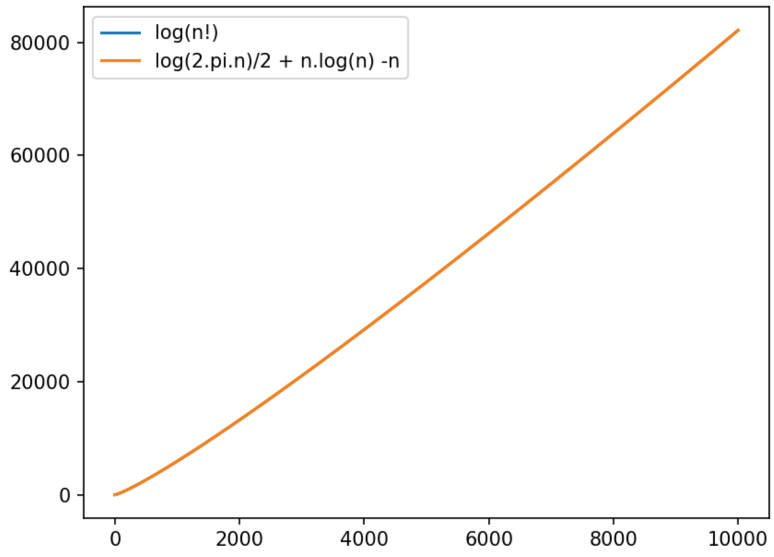
(At first, taking very small value of n, to observe the difference between two expressions)



Output 2: 

```
Enter the value of n=  10000
```

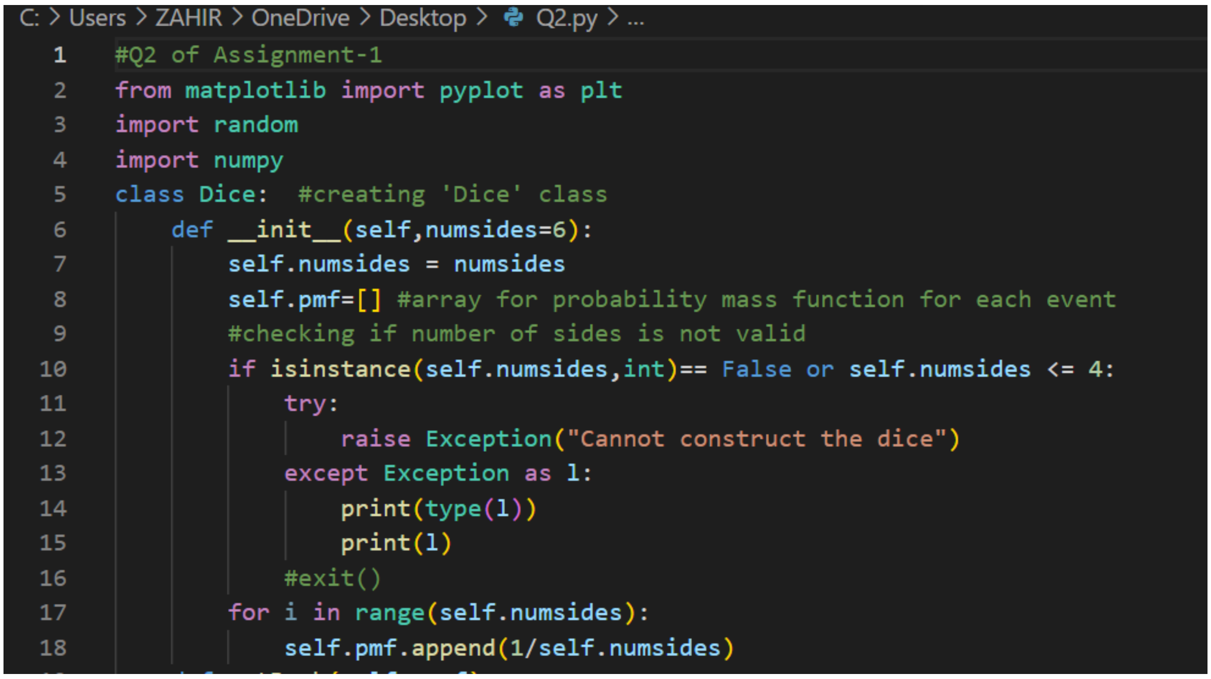
(taking large value of n (10000), to check how much better approximation we are getting)



Observation: So, we are seeing that, more large value we give to n, more better approximation we get.

Q.2) Solution:-

Part1:



C: > Users > ZAHIR > OneDrive > Desktop > Q2.py > ...

```python
1   #Q2 of Assignment-1
2   from matplotlib import pyplot as plt
3   import random
4   import numpy
5   class Dice:  #creating 'Dice' class
6       def __init__(self,numsides=6):
7           self.numsides = numsides
8           self.pmf=[] #array for probability mass function for each event
9           #checking if number of sides is not valid
10          if isinstance(self.numsides,int)== False or self.numsides <= 4:
11              try:
12                  raise Exception("Cannot construct the dice")
13              except Exception as l:
14                  print(type(l))
15                  print(l)
16              #exit()
17          for i in range(self.numsides):
18              self.pmf.append(1/self.numsides)
```

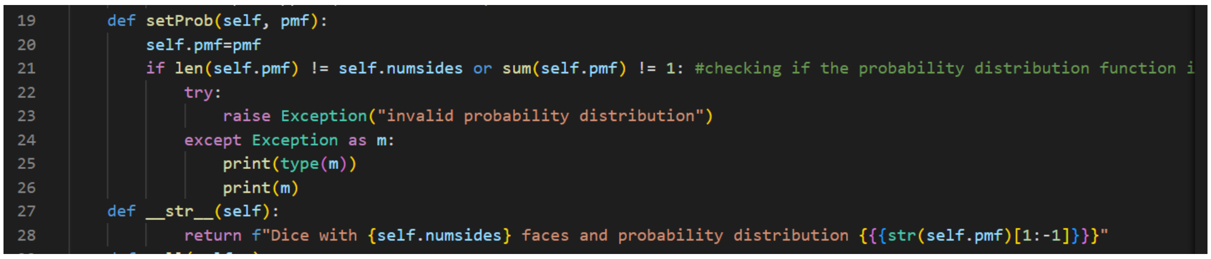
Explanation of part1:

In this part we have defined the construction of the dice with the number of side. In line 10, it is being checked that number of side given by user is integer or not and number of side is greater than 4 or not. If it is not integer or number of side is not greater than 4 then it will raise a exception as “Cannot construction the dice”.

If no input value of n is given by the user then it will take default value 6.

If the user don’t set the probability values of each side of dice then in line 18 , it will calculate uniform probability mass function (for n input, the probability of each value will be 1/n).

Part2:-

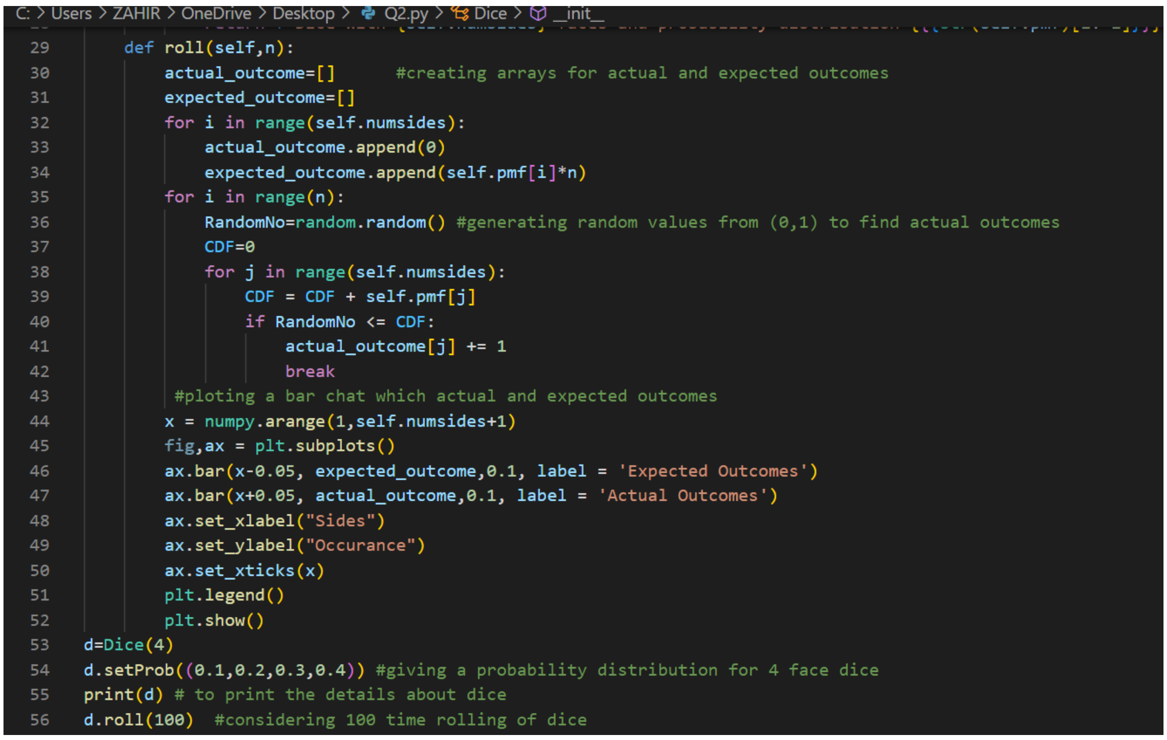


```
19      def setProb(self, pmf):
20          self.pmf=pmf
21          if len(self.pmf) != self.numsides or sum(self.pmf) != 1: #checking if the probability distribution function i
22              try:
23                  raise Exception("invalid probability distribution")
24              except Exception as m:
25                  print(type(m))
26                  print(m)
27      def __str__(self):
28              return f"Dice with {self.numsides} faces and probability distribution {{{str(self.pmf)[1:-1]}}}"
```

Explanation of part2:-

This part mainly checks that default or given probability distribution function is valid or not. If total sum of the probability values is 1 then it is valid. If it is invalid then by line 21-26 it will throw a exception as “invalid probability distribution”. And if it is valid , by line 28 a line will be printed like “ Dice with n faces and probability distribution { a1,a2,…,an}”.

Part3:-


C: > Users > ZAHIR > OneDrive > Desktop > Q2.py > Dice > __init__

```
29      def roll(self,n):
30          actual_outcome=[]      #creating arrays for actual and expected outcomes
31          expected_outcome=[]
32          for i in range(self.numsides):
33              actual_outcome.append(0)
34              expected_outcome.append(self.pmf[i]*n)
35          for i in range(n):
36              RandomNo=random.random() #generating random values from (0,1) to find actual outcomes
37              CDF=0
38              for j in range(self.numsides):
39                  CDF = CDF + self.pmf[j]
40                  if RandomNo <= CDF:
41                      actual_outcome[j] += 1
42                      break
43           #ploting a bar chat which actual and expected outcomes
44          x = numpy.arange(1,self.numsides+1)
45          fig,ax = plt.subplots()
46          ax.bar(x-0.05, expected_outcome,0.1, label = 'Expected Outcomes')
47          ax.bar(x+0.05, actual_outcome,0.1, label = 'Actual Outcomes')
48          ax.set_xlabel("Sides")
49          ax.set_ylabel("Occurance")
50          ax.set_xticks(x)
51          plt.legend()
52          plt.show()
53  d=Dice(4)
54  d.setProb((0.1,0.2,0.3,0.4)) #giving a probability distribution for 4 face dice
55  print(d) # to print the details about dice
56  d.roll(100)  #considering 100 time rolling of dice
```

Explanation of part3:-

In this part we simulate n throws of the dice by calling the function **roll** and it displays a bar chart

showing the expected and actual number of occurrences of each face when the dice is thrown n times.

Calculation of expected-outcome :- Expected outcome is n.p(i) , where n is number of throws and p(x) is the probability value of i-th face. This calculation is done in line 34.

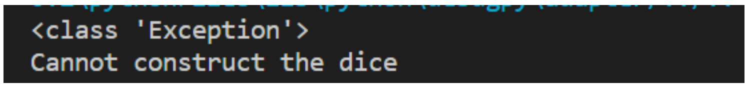
Calculation of actual-outcome:- Actual outcome is calculated in following below steps,

- Initially we take 0 value for each faces in the array **actual_outcome** of length k (number of faces) in line 33 .
- For each throw, initially we take CDF (cumulative distribution function) as 0 in line 37 and choose a random value named **RandomNo** between 0 and 1 (in line 36).
- After that we adding up the probability distribution values one by one (in range 1 to number of sides or faces) with cumulative distribution function (CDF) until we have **RandomNo** <= CDF and then stop.
- If the last probability distribution value (probability mass function) added is corresponding to j-th face then we add 1 in the j-th position of the array **actual_outcome**.

Thus after n throws we get actual outcome number for each faces.

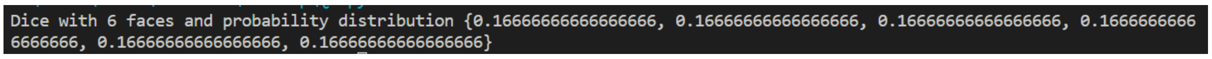
Outputs:-

➢ d= Dice(3)
print(d)



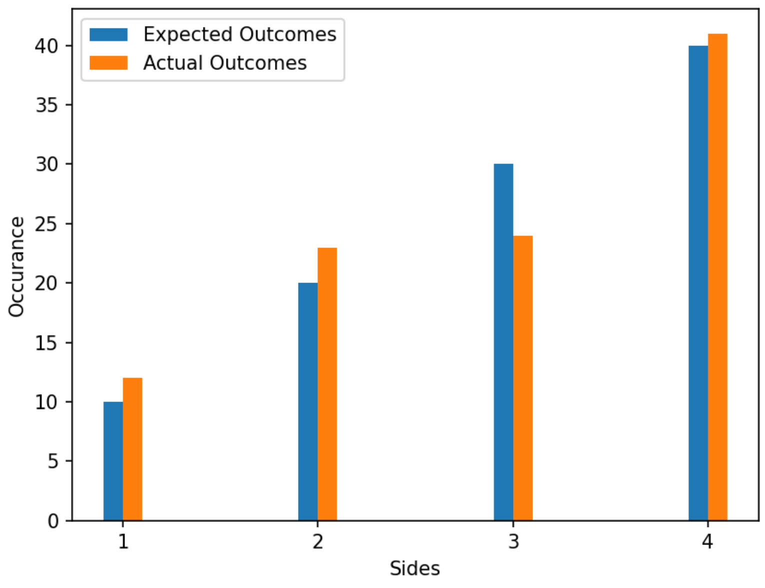
```
<class 'Exception'>
Cannot construct the dice
```

➢ d=Dice()
print(d)

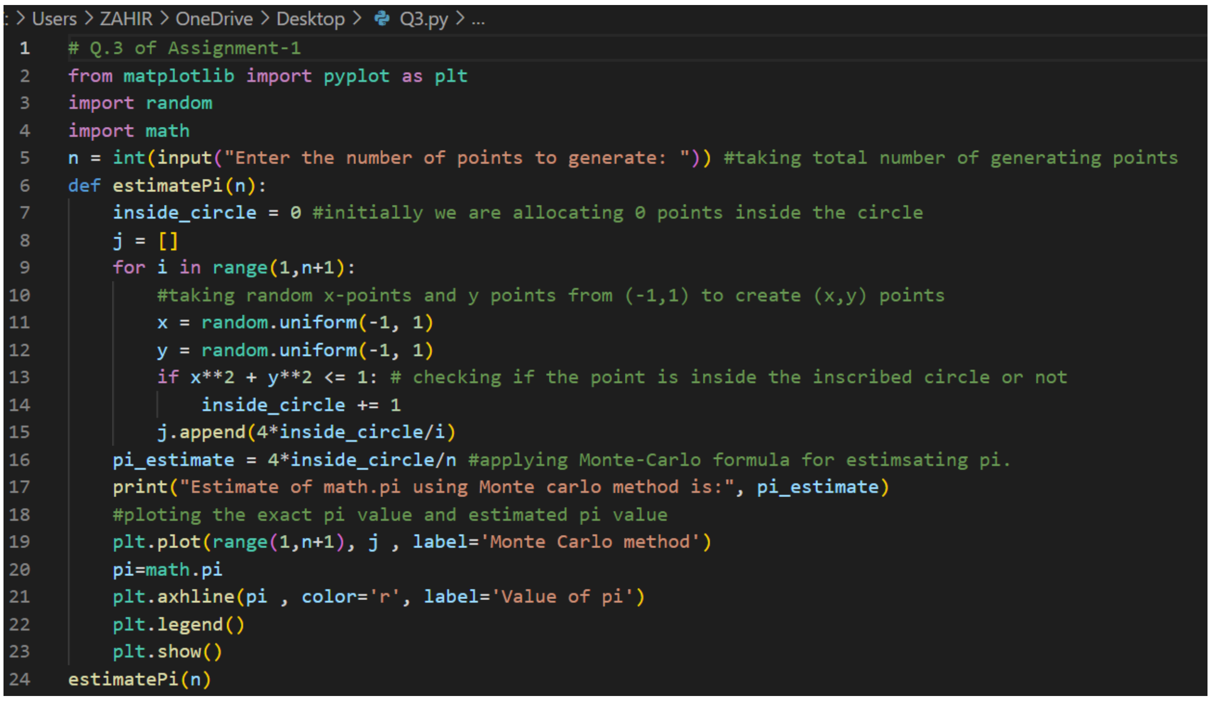


```
Dice with 6 faces and probability distribution {0.16666666666666666, 0.16666666666666666, 0.16666666666666666, 0.1666666666
6666666, 0.16666666666666666, 0.16666666666666666}
```

➢ d=Dice(4)
d.setProb((0.1,0.2,0.3,0.4))
d.roll(100)

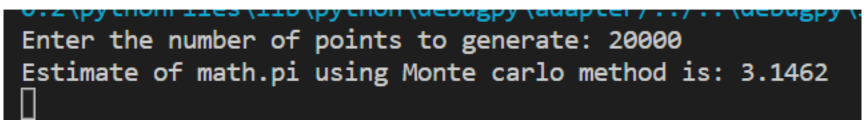


Q.3) Solution:-

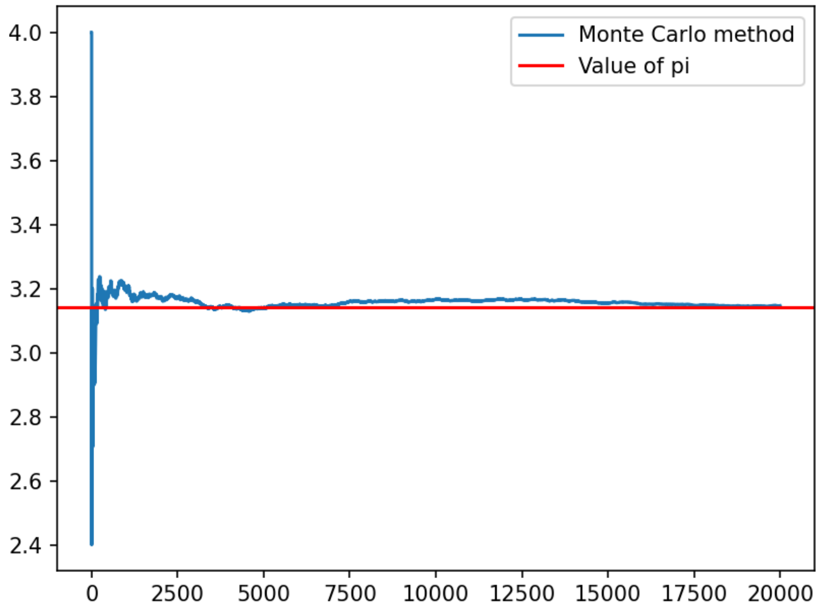


```
: > Users > ZAHIR > OneDrive > Desktop > Q3.py > ...
# Q.3 of Assignment-1
from matplotlib import pyplot as plt
import random
import math
n = int(input("Enter the number of points to generate: ")) #taking total number of generating points
def estimatePi(n):
    inside_circle = 0 #initially we are allocating 0 points inside the circle
    j = []
    for i in range(1,n+1):
        #taking random x-points and y points from (-1,1) to create (x,y) points
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)
        if x**2 + y**2 <= 1: # checking if the point is inside the inscribed circle or not
            inside_circle += 1
        j.append(4*inside_circle/i)
    pi_estimate = 4*inside_circle/n #applying Monte-Carlo formula for estimsating pi.
    print("Estimate of math.pi using Monte carlo method is:", pi_estimate)
    #ploting the exact pi value and estimated pi value
    plt.plot(range(1,n+1), j , label='Monte Carlo method')
    pi=math.pi
    plt.axhline(pi , color='r', label='Value of pi')
    plt.legend()
    plt.show()
estimatePi(n)
```

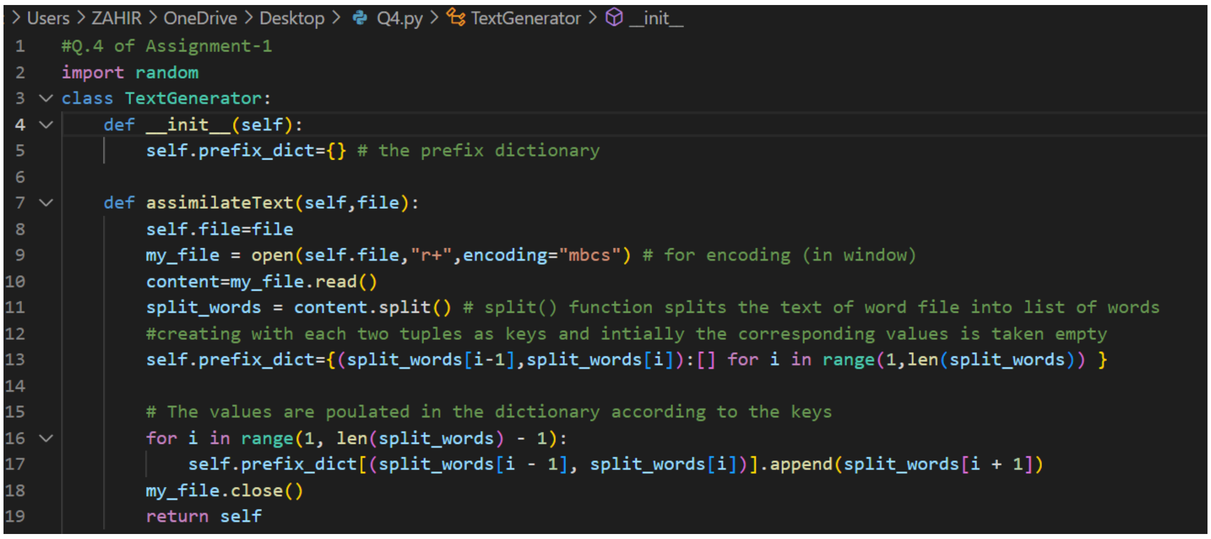
Output:-



```
Enter the number of points to generate: 20000
Estimate of math.pi using Monte carlo method is: 3.1462
```



Q.4) Solution:-

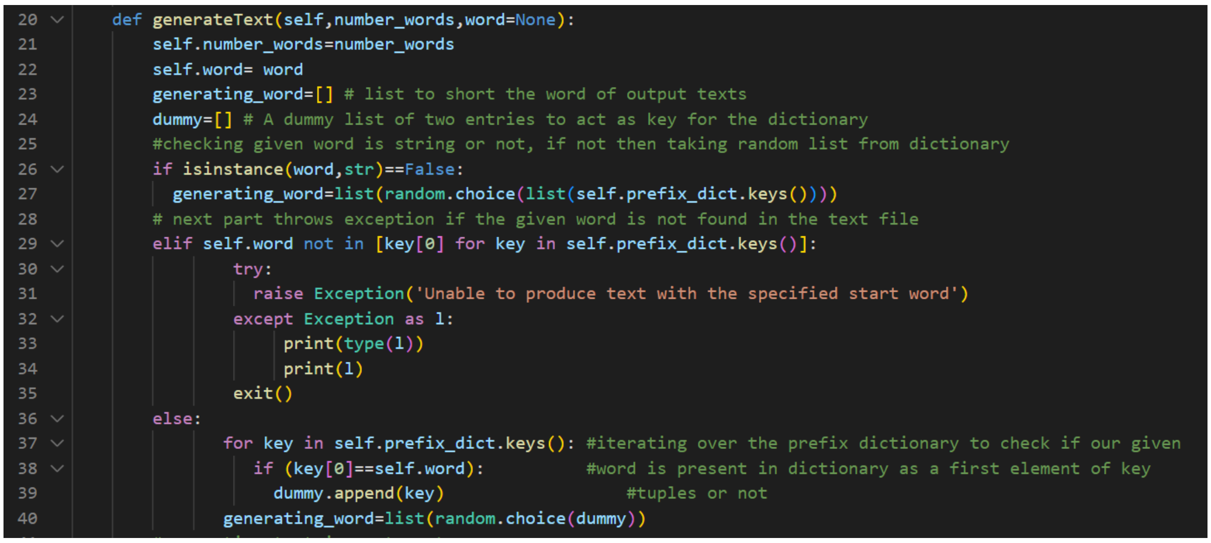

> Users > ZAHIR > OneDrive > Desktop > Q4.py > TextGenerator > __init__

```python
#Q.4 of Assignment-1
import random
class TextGenerator:
    def __init__(self):
        self.prefix_dict={} # the prefix dictionary

    def assimilateText(self,file):
        self.file=file
        my_file = open(self.file,"r+",encoding="mbcs") # for encoding (in window)
        content=my_file.read()
        split_words = content.split() # split() function splits the text of word file into list of words
        #creating with each two tuples as keys and intially the corresponding values is taken empty
        self.prefix_dict={(split_words[i-1],split_words[i]):[] for i in range(1,len(split_words)) }

        # The values are poulated in the dictionary according to the keys
        for i in range(1, len(split_words) - 1):
            self.prefix_dict[(split_words[i - 1], split_words[i])].append(split_words[i + 1])
        my_file.close()
        return self
```

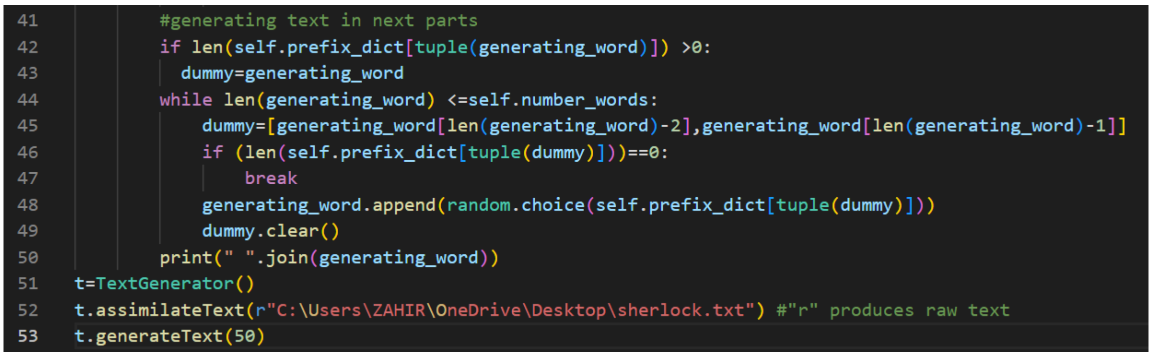
Explanation:- In the function **assimilateText** , we read the text file, then split it into list of words after that we create prefix dictionary.



```python
    def generateText(self,number_words,word=None):
        self.number_words=number_words
        self.word= word
        generating_word=[] # list to short the word of output texts
        dummy=[] # A dummy list of two entries to act as key for the dictionary
        #checking given word is string or not, if not then taking random list from dictionary
        if isinstance(word,str)==False:
            generating_word=list(random.choice(list(self.prefix_dict.keys())))
        # next part throws exception if the given word is not found in the text file
        elif self.word not in [key[0] for key in self.prefix_dict.keys()]:
                try:
                    raise Exception('Unable to produce text with the specified start word')
                except Exception as l:
                        print(type(l))
                        print(l)
                exit()
        else:
                for key in self.prefix_dict.keys(): #iterating over the prefix dictionary to check if our given
                    if (key[0]==self.word):         #word is present in dictionary as a first element of key
                        dummy.append(key)               #tuples or not
                generating_word=list(random.choice(dummy))
```

Explanation:-

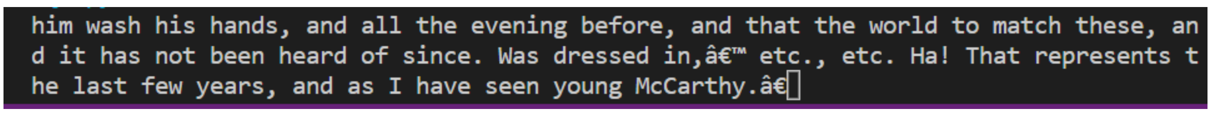
In **generateText** , the list variable **generating_word** is used to generate text and **dummy** works as an iterative key to generate the random words using random.choice(). If no word is given by user then **generating_word** choice a random key from prefix dictionary. If word is given, the key is used in the prefix dictionary to find matches and then begins process to generate words randomly.



```python
            #generating text in next parts
            if len(self.prefix_dict[tuple(generating_word)]) >0:
                dummy=generating_word
            while len(generating_word) <=self.number_words:
                dummy=[generating_word[len(generating_word)-2],generating_word[len(generating_word)-1]]
                if (len(self.prefix_dict[tuple(dummy)]))==0:
                    break
                generating_word.append(random.choice(self.prefix_dict[tuple(dummy)]))
                dummy.clear()
            print(" ".join(generating_word))
t=TextGenerator()
t.assimilateText(r"C:\Users\ZAHIR\OneDrive\Desktop\sherlock.txt") #"r" produces raw text
t.generateText(50)
```

Explanation:- This part generates the desired text by taking words one by one from the list and joining them with a gap of space.

Output:- t.generateText(50)



```
him wash his hands, and all the evening before, and that the world to match these, an
d it has not been heard of since. Was dressed in,â€™ etc., etc. Ha! That represents t
he last few years, and as I have seen young McCarthy.â€
```

t