INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

COMPUTER SCIENCE AND ENGINEERING
Indian Institute of Technology, Palakkad
CS5016: Computational Methods and Applications
Assignment 1: Monte Carlo Method

24 Feb, 2021

Max points: 100

A few instructions

- Codes should be compatible with *Python3* and should run on Ubuntu.
- Code for each question should be placed in a separate file (*Q1.py, Q2.py, Q3.py and Q4.py*).
- Codes should be well-commented.

1. **Stirling's approximation (or Stirling's formula)** is an approximation for factorials. It is a good approximation, leading to accurate results even for small values of n[1]. The approximation is as follows

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^{e}$$

Write a code to visualize the above result for values of n upto 10^6. [20]

2. Create a class `Dice` with attributes `numSides` such that [30]

- It should be possible to create an object of this type by specifying the number of sides of the dice. If number of sides is not mentioned, create a 6 sided dice.

```
//The following code should create a 6 faced dice
d = Dice()
```

```
//The following code should create a 10 faced dice
d = Dice(10)
```

- Number of sides should be an integer greater than 4. Otherwise, an **exception** should be thrown.

```
d = Dice(3)
```

Expected output:

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

[1] `https://en.wikipedia.org/wiki/Stirling's_approximation`

Question 2 continues on the next page...

```
d = Dice(4.5)
```

Expected output:

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

```
d = Dice('5')
```

Expected output:

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

- By default, each face of the dice should have equal probability of occurrence. However, by using the function `setProb` one should be able to set probability of occurrence of each face. This function should take a tuple as its argument

```
//The following code should create a 4 faced dice with probability
//distribution {0.1, 0.2, 0.3, 0.4}
d = Dice(4)
d.setProb((0.1, 0.2, 0.3, 0.4))
```

- If the tuple passed as argument to `setProb` is not a valid probability distribution for the dice, an **exception** should be thrown. A examples are as follows

```
d = Dice(4)
d.setProb((0.1, 0.2, 0.3))
```

Expected output:

```
<class 'Exception'>
Invalid probability distribution
```

```
d = Dice(4)
d.setProb((0.5, 0.2, 0.3, 0.4))
```

Expected output:

```
<class 'Exception'>
Invalid probability distribution
```

- It should be possible to `print` an object of type `Dice`.

```
d = Dice(5)
print(d)
```

Question 2 continues on the next page...

Expected output:

```
Dice with 5 faces and probability distribution {0.2, 0.2, 0.2, 0.2, 0.2}
```

```
d = Dice(4)
d.setProb((0.1, 0.2, 0.3, 0.4))
print(d)
```

Expected output:

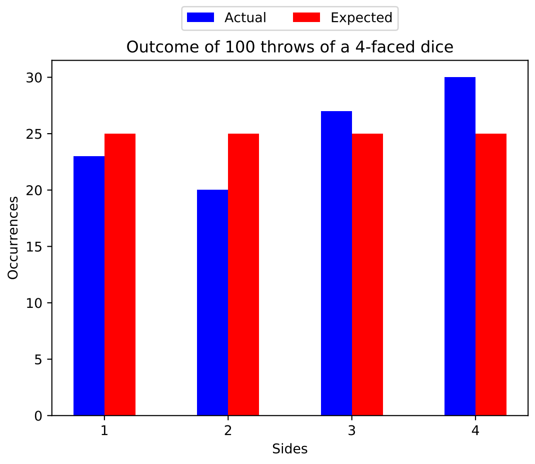
```
Dice with 4 faces and probability distribution {0.1, 0.2, 0.3, 0.4}
```

- It should be possible to simulate n throws of the dice by calling the function `roll`. This function should take n, the number of throws, as its arguments and displays a bar chart showing the expected and actual number of occurrences of each face when the dice is thrown n times.

  NOTE: Only `matplotlib` and `random` modules are to be imported. From `random` module, you are only allowed to use the `random` function.

```
d = Dice(4)
d.roll(100)
```

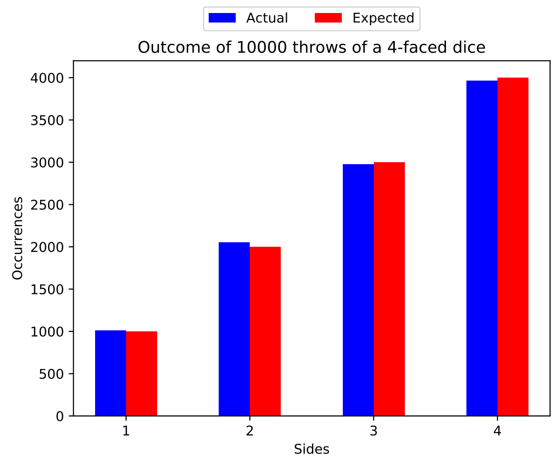
Expected output:



```
d = Dice(4)
d.setProb((0.1, 0.2, 0.3, 0.4))
d.roll(10000)
```

Expected output:

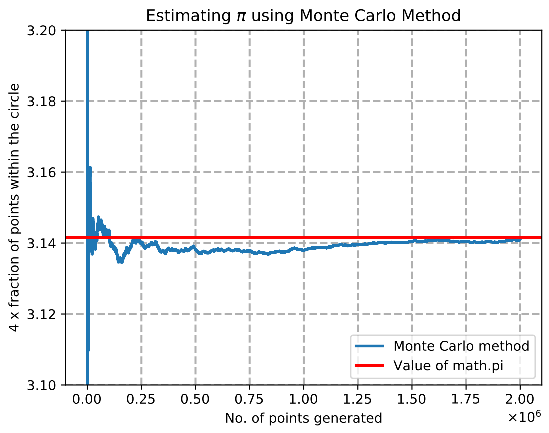
Continued on next page...



3. Write a function **`estimatePi`** that estimates π using the Monte Carlo method. This function should take as argument a positive integer n that denotes the total number of points generated in the simulation.

```
estimatePi(2000000)
```

Expected output:



NOTE: Plot generated by your code should be similar to the above plot.

4. Create a class `TextGenerator` such that

   - It has a function `assimilateText` that takes a file name as its argument. It then read all the text in the file and creates a prefix dictionary that maps a pair (2-tuple) of words to a list of words which follow that pair in the text.

```
t = TextGenerator()
t.assimilateText('sherlock,txt')
```

Question 4 continues on the next page…

- It has a function `generateText` that creates random text based on the triplets contained in the prefix dictionary. This function has a mandatory argument that let it know the number of words to be produced in this random manner.

```
t = TextGenerator()
t.assimilateText('sherlock,txt')
t.generateText(100)
```

Expected output:

```
smelling of iodoform, with a bright, quick face, freckled like a sheep
in a single branch office, and the schemer falls into the stable lane.
So long was he doing there at all? If his purpose were innocent, why
did you first how I employed my morning, or the grace and kindliness
with which I beg pardon. As to Miss Turner. On the next evening I would
find that all was dark in the air. "May I see it?" But I think that the
gas is not so much public attention has now definitely announced his
approaching marriage with so
```

- It should be possible to invoke the function `generateText` with an additional argument that fixes the first word in the random text it produces.

```
t = TextGenerator()
t.assimilateText('sherlock,txt')
t.generateText(50, 'London')
```

Expected output:

```
London for the grace and kindliness with which I beg that you have
described. You must find your own theory as to come to the altar faced
round to his credit at the pool I heard the rush of constables with an
inspector, all on fire to the salesman just
```

- If `generateText` is not able to produce random text with the specifed start word, it should throw an **exception**.

```
t = TextGenerator()
t.assimilateText('sherlock,txt')
t.generateText(50, 'Wedge')
```

Expected output:

```
<class 'Exception'>
Unable to produce text with the specified start word.
```

NOTE: Since the text generated by `generateText` is random in nature, text produced by your code need not match the expected output shown above.