**Problem 1: Rolling a single “fair” and “un-fair” die**

Probability theory uses the metaphor of rolling a “fair die” to illustrate the concept that, upon each roll of the die, the probability is equal for scoring any of the numbers 1-6 (i.e., 1/6 chance of rolling a 3). Computers are calculation machines where the very idea of randomness runs counter to the orderly workings of binary and hexadecimal mathematics. Nevertheless, we want to write a simple program that utilizes a random number generator to simulate the behavior of a “fair die” and an “un-fair die” (i.e., the probability of one or more of the sides is slightly more probable than the others) and test these dice over a large number of rolls to see if the probabilities we predict are in fact simulated with our program.

**To Do:**

1. Open the Python file called Die.py in PyCharm.
2. Complete the function called rollFairDie() that takes no arguments and returns an integer between 1 and 6 inclusively, but at random with an equal probability each time the function is called. This means that each roll must be independent of the past or future rolls.

a. You must import the module random and call the function random() to produce a floating point number between 0.0 and 1.0. While the random module that can produce integers between 1 and 6 randomly, I want you to produce a floating point number between 0.0 and 1.0.

b. Convert the floating point number to an integer (1-6) with equal probability by dividing the range 0-1 into six equal bins and using an if-then structure to return the appropriate integer. For example, we might break the range between 0-1 into six equal pieces like:

x = random()

if (x ≤ 1.0/6):

return 1

else if (x ≤ 2.0/6):

return 2

etc.

1. In a separate file called HW1SP25\_Prob1.py:
   1. Import from your module Die the function rollFairDie with an alias rfd
   2. Write and call a main() function that calls rfd() 1000 times and computes the fraction of rolls that yield 1, 2, 3, etc.
   3. Print the observed probability of each of the possible numbers to the cli (command line interface) (i.e., the screen) as formatted text:

After rolling the die 1000 times:

Probability of rolling a 1: 0.1667

Probability of rolling a 2: 0.1668

Etc.

* 1. Do these probabilities match the theory? Calculate the probabilities if we roll the die 10,000 times by writing and calling a main2() function that calls rollFairDie() 10,000 times and computes the fraction of rolls that yield 1, 2, 3, etc.
  2. Output the probability of each of the possible numbers to the screen as formatted text:

After rolling the die 10,000 times:

Probability of rolling a 1: 0.1667

Probability of rolling a 2: 0.1668

Etc.

1. Modify your Die.py file to include a function called rollUnFairDie() where the die has been modified to roll a 1 with a probability of 0.2. Write and call a main3() function that calls the rollUnFairDie() 10000 times and outputs the results as in step 3. Note: your unfair die can be made by enlarging one of the bins in part 2b of the To Do: list and shrinking the remaining 5.

**Problem 2: Rolling dice**

Now that we can roll a single die, we want to see the probabilities of rolling any possible number when we have **N** dice rolled simultaneously.

**To Do:**

1. Write a new python file called Dice.py that imports rollFairDie and rollUnFairDie from Die.py and includes a function called rollDice(**N**=1), where **N** is a keyword argument that specifies the number of dice to be rolled. rollDice should call rollFairDie N times and return the total score by summing the score of each die.
2. In a separate file called HW1SP25\_Prob2.py, write and call a main() function that specifies the number of dice (n) and calls the function rollDice(**N**=n) 1000 times and then outputs the probability for each possible score to the screen as:

Probability of rolling a 3: 0.xxx

Etc.

(note: for 3 dice, min score =3, max score =18, for 4 dice, min score = 4, max score = 24, etc.)

1. Create a function called main2() that rolls 5 unfair dice 1000 times and output the probabilities as in 2.

**Problem 3: A continuous random variable**

**Description:** To simulate real-world behavior of systems, we often must produce normally distributed values for a property of a system (e.g., the average mass of a pebble in a pile of gravel). Here, you must produce a program that yields list of data that is *normally distributed* around a specified mean (μ) and standard deviation (σ). (Note: if I use a random number generator to produce a uniformly distributed value between 0 and 1, we can use this value as the dependent variable on a cumulative distribution function and find the independent variable value (x) corresponding to this probability. The values of x will be normally distributed.)

Your program should output an array of size N=1000 drawn from normally distributed population. After this sample of data has been generated, you should calculate the sample mean and standard deviation and print the result of these calculations to the screen. You should verify that the data is normally distributed by computing the 1, 2 and 3 standard deviation criteria. You should do this work in a file called HW1SP25\_Prob3.py

Note: there is a function in the module random called normalvariate() that returns numbers from a normal distribution of given mean and standard deviation.

**NOTE:** For every function you write, you should include a docstring that explains what the function does, what the arguments mean and what is returned from the function. Failure to include docstrings will result in an automatic 5 point deduction from your score.

A graph of a function

Description automatically generated with medium confidence