Bloomberg STARTUP

Week 4 In-class Exercise

1) While Loop Practice

Create a new file called 'while_loop_practice.py'. Make sure Python 3 is selected in the top right corner of the output window where the code runs.

Write and test the following functions:

a.)
```
def print_numbers(min, max):
# prints all numbers between min and max in ascending order
```

b.)
```
def print_numbers_desc(min, max):
# prints all numbers between min and max in descending order
# (start with max and count down)
```

c.)
```
def print_by_25(min, max):
# prints numbers between min and max, counting up by 25
```

Testing Your Functions

Test each function as you go, testing your functions with hard coded numbers:

```
print_numbers(0,100)
print_numbers(-23,-5.234)
```

- Do your functions work with negative numbers?
- Decimal numbers?
- What happens if max < min?

2) Dice Probabilities

Create a new file called 'dice_rolls.py'.

We can use the 'random' module to simulate dice rolls by picking a random number between 1 and 6:

```
    dice_roll = random.randint(1,6)
```

2a)

First write a function 'roll_die(num_rolls)' that rolls a die and prints out the value however many times specified by the parameter 'num_rolls'.

```
    roll_die(6)                 =>   5
                                     6
                                     1
                                     3
                                     5
                                     6
```

2b)

Next, write a new function 'roll_two_dice(num_rolls)' that does the same thing as function 'roll_dice' only it rolls two separate dice, and adds them together before printing.

```
    roll_two_dice(4)            =>   5
                                     12
                                     7
                                     9
```

2c)

Now you can leverage the speed of computers to approximate probabilities. Rather than doing math to calculate the probability of rolling snake-eyes, we can have a computer simulate thousands of dice rolls and observe how many snake-eyes occur.

Write a new function 'count_snake_eyes(num_rolls)' that rolls two dice 'num_rolls' times, just like before.

If the sum of the two dice is 2, it is a snake-eyes. Keep a separate count of the number of snake-eyes that occur.

After all the dice rolls are done, print out the number of snake eyes that occurred out of the total number of dice rolls, along with the percentage.

```
    count_snake_eyes(1000)
        =>   "23 snake-eyes found out of 1000 rolls - 2.3%"
```

Fill out the following table with the output from your count_snake_eyes function:

Num Rolls	% Snake Eyes
10	
100	
5000	
50000	
100000	
1000000	

2d)

Now copy/paste the 'count_snake_eyes' function and rename the copied function 'count_n'. This new function should take an additional parameter, 'n'. It should roll two dice 'num_rolls' times, just like before but instead of counting snake-eyes, it should count how many times the sum of the two dice equal the second parameter, n.

```
def count_n(num_rolls, n):
# your code here
```

Reminder: 'count_n(100000, 2)' should do exactly the same thing that 'count_snake_eyes(100000)' did.

Fill out the following table using your new count_n function:

Num Rolls	n	% Found
100000	2	
100000	6	
100000	7	
100000	8	
100000	11	
100000	12	
100000	13	(Hopefully 0!)

3) Dice Game (Challenge)

Create a dice game with rules as follows:

In each round, you are trying to maximize your score.
For your turn, you roll 5 dice. You see the value of each die, and have the option to re-roll twice.

After your optional additional rolls, your score is the sum of the most recently rolled 5 dice.

The computer then rolls 5 dice and its score is the sum of the 5 dice. The computer does not get to re-roll its dice.

If your score is higher than the computers you win this round, otherwise the computer wins this round.

One complete game consists of 5 rounds. Whoever wins the most rounds wins the game.

3b Challenge of challenges

Add some AI to the computer's behavior so that the computer can take advantage of optional re-roll just like the user. Devise a strategy for when the computer should re-roll. This should make it harder to beat the computer.