UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 7

Recursion

Date Due: Friday, November 8, 2019, 6:00pm **Total Marks: 21**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- **Read every question carefully.** All of it. These questions cannot be described in a few sentences, and except for a little bit of fun context, all parts of the question have important information crucial to your success.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M (yes, sometimes typos happen – do not be confused). Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you. Do not submit folders, or zip files, even if you think it will help.
- **Programs must be written in Python 3.** Marks will be deducted with severity if you submit code for Python 2.
- **Assignments must be submitted to Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Questions are annotated use descriptors like "easy" "moderate" and "tricky". All students should be able to obtain perfect grades on "easy" problems. Most students should obtain perfect grades on "moderate" problems. The problems marked "tricky" may require significantly more time, and only the top students should expect to get perfect grades on these. We use these annotations to help students be aware of the differences, and also to help students allocate their time wisely. Partial credit will be given as appropriate, so hand in all attempts.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (6 points):

Purpose: To practice recursion with a simple example.

Degree of Difficulty: Easy.

The evil Team Rocket has invented an incredible new space-ship! The ship works as follows:

- If the distance to the destination is greater than 1 meter, the ship will "fold space" and "jump" to a position exactly half way to the destination. It takes exactly **one minute** to perform this process.
- Any distance of **1 mete**r or less will take exactly **one minute**, using normal impulse rockets.

For example, if the ship has to travel 10 meters, it will jump 5 meters after the first minute, 2.5 meters the second minute, 1.25 meters the 3rd minute, and 0.625 meters the fourth minute. Finally, the remaining 0.625 meters takes one more minute. Thus the total time to travel 10 meters is 5 minutes. (That seems very slow, and it is for small distances; the true value of this method only reveals itself for large distances.)

Write a recursive function called `spaceTime()` that calculates the time needed for Team Rocket's new ship to travel a given distance (in meters). The distance to travel must be a parameter to the function, and the function itself must not do any console input.

Test your function out on all of the following examples:

- Distance from Team Rocket's base to the nearest Poke-Stop : 37 meters.
- Distance for a round-trip around the entire planet earth: 40075000 meters.
- Average distance between our earth and our sun: 1.49e11 meters (about 150 million km).
- Approximate distance between the sun and the closest star: 4.0e16 meters (about 4 light-years).
- Size of the observable universe: 8.8e26 meters (about 93 Giga-light-years)

Sample Run

Here is an example execution of the required program.

```
Team Rocket's drive requires:
- 7 minutes to travel 37 meters to the nearest Poke-Stop
- 27 minutes to travel 40075000.0 meters for a round-trip around the earth
- 39 minutes to travel 149000000000.0 meters from our earth to our sun
- 57 minutes to travel 4e+16 meters from our sun to the nearest star
- 91 minutes to travel 8.8e+26 meters across the observable universe

WOW, IS TEAM ROCKET EVER BLASTING OFF AGAIN!!
```

Your console output doesn't have to look exactly like the above. Copy/paste your output for all of the examples above into a document called `a7q1_output.txt` for submission.

What to Hand In

- A document called `a7q1.py` containing your finished program, as described above.
- A document called `a7q1_output.txt` (rtf and pdf formats are also allowable) with the output of your program copy/pasted into it, to demonstrate that your program works.

Evaluation

- **-1 mark** for lack of name, nsid, student number and instructor in the solution
- 1 mark: The function has a parameter for the distance to be travelled.
- 1 mark: The base case is correct
- 2 marks: The recursive case is correct
- -2 marks: The function uses global variables, or otherwise tries to sidestep a nicely recursive solution.
- 1 mark: The recursive function has an adequate docstring
- 1 mark: You submitted a document showing the output of your program on all of the examples given above.

Question 2 (9 points):

Purpose: To practice recursion on a problem that can be split into parts

Degree of Difficulty: Moderate

Aspiring pokemon trainer Ash Ketchum has caught a lot of pokemon. To keep them healthy, he regularly feeds them vitamins. However, the vitamins are bigger than what a pokemon can swallow in one bite, and so when Ash's pokemon team is given a vitamin, they will break it apart into smaller pieces.

Sometimes, though, the vitamin is SO big, that the pokemon have to break it apart more than once! Here is how the process works:

- If the vitamin's weight is less than or equal to 0.1 grams, it is small enough to eat, and doesn't need to be broken. Thus, there is 1 edible piece for 1 pokemon.
- Otherwise, the pokemon will team up to smash the vitamin into pieces of equal weight. The number of new pieces seems to be random (the pokemon can be a bit excitable), either 2, 3, or 4, but the weight of each new piece is the same (one-half, one-third or one-quarter of the previous piece). The pokemon will then break each of these pieces again until they are small enough to swallow.

For example, suppose the initial vitamin weighed 0.6 grams. The pokemon break it once, and we randomly determine that the vitamin breaks into 2 parts (each part weighs 0.3 grams). The pokemon break the first 0.3 gram piece, which breaks into 3 pieces. Since these pieces will each weigh 0.1 grams, they are now edible, so we have 3 edible pieces so far. The pokemon will then break the second 0.3 gram piece; this time, it breaks into 2 parts, each of which weighs 0.15 grams, which are still too large to eat. The pokemon break each of these parts again; the first breaks into 3 parts, each weighing 0.05 grams, so we have 3 more pieces. The second breaks into 4 parts, each weighing 0.0375 grams, so that's another 4 pieces. There are no longer any pieces larger than 0.1 grams, so the total number of edible pieces is 3 + 3 + 4 = 10 pieces. So in this case, 10 different pokemon are able to get their vitamin dosage from a single, original 0.6 gram vitamin.

Tracing through the problem like in the paragraph above is exhausting! But if you bring yourself to trust in the power of recursion, solving this problem is not hard at all.

For this question, your task is to write a recursive program that will calculate how many edible vitamin pieces are made whenever a pokemon team is given a vitamin that weighs W grams.

Program Design

(a) Write a recursive function that simulates the breaking of a single vitamin. The weight of the vitamin (as a float) should be a parameter to your function. The function should return an integer, indicating the number of edible pieces produced from breaking the vitamin.

To write this function, you will need to use random numbers for when the vitamin is broken into parts. If you first `import random as rand`, then the expression `rand.randint(a,b)` will give you a random number in the range from a to b (including a and b). **For this question, you ARE allowed to use a loop in your recursive function if you like; however, recursion should still do the "real work"**. For instance, you might want to use a loop to iterate over the number of vitamin pieces created from a single smash.

(b) In the 'main' part of your program, write a few lines of code that asks the user for the size of a vitamin, and uses a loop that will **call your piece-counting recursive function 1000 times**. Use the results of those 1000 simulations to report the average number of edible pieces produced from a vitamin.

An example of your program running might look like this:

```
How big is the vitamin, in grams? 1.0
On average, a pokemon team can get 18.774 bite-sized pieces from a 1 gram vitamin!
```

Note that because of the randomness involved you might never get this exact result with an input of 1.0 grams but it should be pretty close.

Run your program using vitamins of weight 5, 10 and 100 grams. Copy/paste your output for all of the examples above into a document called `a7q2_output.txt` for submission.

What to Hand In

- A document entitled `a7q2.py` containing your finished program, as described above.
- A document called `a7q2_output.txt` (rtf and pdf formats are also allowable) with the output of your program copy/pasted into it, to demonstrate that your program works.

Be sure to include your name, NSID, student number, course number and lecture section and laboratory section at the top of all files.

Evaluation

- **-1 mark** for lack of name, nsid, student number and instructor in the solution
- 1 mark: The function has a parameter for the size of the vitamin.
- 1 mark: The base case is correct.
- 3 marks: The recursive case is correct.
- -2 marks: The function uses global variables, or otherwise tries to sidestep a nicely recursive solution.
- 1 mark: The recursive function has an adequate docstring
- 2 marks: You program correctly takes an average count of edible pieces, using 1000 trials.
- 1 mark: You submitted a document showing the output of your program on all of the examples given above.

Question 3 (6 points):

Purpose: To practice recursion on a classic problem

Degree of Difficulty: Tricky.

Nim is a 2-player game where the players take turns picking up sticks from a pile. On their turn, a player can pick up either 1, 2, or 3 sticks. The player who picks up the last stick loses.

Your job is to write a **recursive function** that determines, for a given number of sticks, the winner of the game assuming that both players play optimally. Your function should take **exactly 2** parameters: an integer indicating the current number of sticks, and a boolean indicating which player is currently to move. It should return the boolean value `True` if player 1 (the player that took the very first turn) will win and `False` otherwise.

For example, if there are 4 sticks left and player 1 is to move, then player 1 can win by taking 3 sticks. This will leave only 1 stick left, which player 2 will have to pick up, thus losing the game.

This might sound a little difficult, especially the assumption that players will play optimally, but with the power of recursion, this is not a complicated problem. Your general approach should be to use recursion to simply **try all possible moves** (i.e. make 1 recursive call for each amount of sticks that can be removed) in each position.

Your function should be **no longer than 12 lines of code** (not counting comments) and possibly less (ours is 9). If you find your solution is getting any longer than that, you are overthinking it!

Sample Run

The output for your program might look something like this:

```
Enter the number of sticks for nim: 9
Will player1 win?
False
```

Extra Notes

Nim is a well-known problem, and it is of course possible to come up with a mathematical rule that will tell you which player will win. You won't get any marks for doing this though, as the point of the assignment is to practice using recursion.

What to Hand In

- A document entitled `a7q3.py` containing your finished program, as described above.

Evaluation

- **-1 mark** for lack of name, nsid, student number and instructor in the solution
- 1 mark: The recursive function is called correctly using appropriate arguments
- 1 mark: The base case(s) are correct
- 3 marks: The recursive case is correct
- -2 marks: The function uses global variables, or otherwise tries to sidestep a nicely recursive solution.
- 1 mark: The recursive function has an adequate docstring