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 8

Recursion

Date Due: 5:00pm Friday Nov 22 **Total Marks: 28**

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.
- 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.
- 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. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.5+
- You may not, in part or in whole, submit any code that was written by generative AI tools, such as ChatGPT
- **Assignments must be submitted to Canvas (the course website).**
- **Canvas will not let you submit work after the assignment hand-in closes.** 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.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (4 points):

Purpose: To practice recursion with a simple example.

Moosetopia's most famous rocket scientist, Zeno, has invented a new kind of space ship which will enable interstellar travel across the large distances of space very quickly. The technology is very complicated, and involves a lot of tedious messing about in hyperspace, but the actual motion of the ship is quite simple:

- If the distance to the destination is greater than 1 meter, the ship will "fold space" and "jump" to a position half way to the destination. It takes exactly one minute for the ship's computer to do the calculations prior to the jump. The jump itself is instantaneous.
- Any distance of 1 meter 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 Zeno's ship to travel a given distance. In the same file, write a program that asks the user for a distance in meters (console input) then call your `spaceTime()` function to calculate the time required to travel the given distance.

Test your function out on all of the following examples:

- Average distance to the nearest coffee shop: 537 meters (about a typical Moosetopia city block).
- Average distance from Moosetopia to its biggest moon: 3.84e8 meters (about 400 thousand km).
- 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

Your program output might look something like the following:

```
Zeno's drive requires:
- 11 minutes to travel 537 meters to Zeno's nearest coffee shop
- 30 minutes to travel 384000000.0 meters from Moosetopia to its biggest moon
- 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
```

What to Hand In

- A document called `a8q1.py` containing your finished program
- A document called `a8q1_output.txt` (rtf and pdf formats are also allowable) with your program output

Evaluation

- 3 marks for a correct and useful use of recursion
- 1 mark for submitting your program output

Question 2 (4 points):

Purpose: To practice recursion with a wrapper function

In the lecture slides, you may have attempted an exercise to print out all the characters in a string in reverse using recursion. For this question, you'll tackle a similar task: printing out all of the WORDS in a string in reverse. You can assume that separate words in a string are always separated by at least one space.

Sample Run

Assuming our original sentence was `DO I CHOOSE YOU PIKACHU`, your function should produce the following output [1]:

```
PIKACHU YOU CHOOSE I DO
```

All of the words should be printed on the same line. It is okay if you end up having a trailing space after the last word.

Program Design

When you write programs to solve problems using loops, very often the code doesn't jump right into a loop first thing. Often, there's a bit of set-up that happens first. The same can be true of recursion.

To solve this problem, you should write TWO functions. The **first** function should be called something like `reverse_phrase()` and **must** have a SINGLE parameter: the string that represents the sentence to be reversed. This function should not itself be recursive. It simply does any necessary set-up before calling your second (recursive) function, which is where the real work will be done.

Your **second** function should be called something like `reverse_phrase_recursive()`. It can have any number of parameters that you think you need, and those parameters can be of any data type that you think will be easiest to work with. This function must be recursive and is not allowed to use loops in any way.

To test your program, the "main" part of your program should simply call `reverse_phrase()` with the string you want to reverse as an argument.

What to Hand In

- A document called `a8q2.py` containing your finished program, as described above.

Evaluation

- 1 mark for a wrapper function that usefully sets up the recursion
- 3 marks for correctly and usefully using recursion to solve the problem (with NO global variables)

[1] If Jedi master Yoda played Pokemon, he would need a tool like your program.

Question 3 (6 points):

Purpose: To practice recursion on an engaging problem

Detective Pikachu has discovered an ancient ruin that is filled with fabulous treasure! But Pikachu's bag will only hold up to 100kg of items. What are the best treasures for him to take, given his limited space?

To answer this question, you will write a **recursive** function.

The input for this problem consists of a file listing the available treasures in a room. Each line of the file lists the info for one treasure, consisting of the treasure's name, its value (in dollars) and its weight (in kg).

Solve Room 1 by Hand

Open the file `room1.txt` and look at the treasures there. Then figure out for yourself, just by hand without writing any code, the best selection of treasures that Detective Pikachu can take. Remember that Pikachu's bag can hold a maximum of 100kg worth of treasure. In a written document (that is separate from the code you will write later), submit your list of treasures he should take, along with the total dollar value of those treasures.

Write your Recursive Function

Now it's time to write your recursive function that will calculate the best **total value** of treasure that Pikachu can take from any given room (your function won't actually list WHICH treasures should be taken; it wouldn't be too hard to add that feature as well, but we'll keep things as simple as possible).

A starter file has been provided that reads the data from a room file into a **list of records**. Read the function's docstring and run it to see a sample of what the resulting list of treasures will look like.

Then, write your recursive function. The function should take exactly two parameters:

- the list of treasures being examined
- an integer indicating the amount of remaining space in Pikachu's bag

The function should return the **best possible value** that Pikachu can get from selecting the optimal treasures to fill his bag. Do NOT add any additional parameters to the function, nor should you try to use global variables in **ANY** way. None are needed.

In the main portion of your program, you'll call your function using the list of treasures produced by `read_data()` and a value of 100 for the space in the bag.

Gather Results

Once your function is complete and you're fairly confident it's correct (remember, you solved Room 1 by hand, so your program should give you the same answer there), run your program on all of the provided room files (you can do this by just changing the value of the `file` variable in the provided code). In your written document, report the best treasure value that your program finds for each room.

Finally, call your function one last time on `room3`, but this time, use a bag size of 300 instead of 100. Report the value that your function returns for this bigger bag size. Do you notice anything "funny" compared to the previous times you ran your program? If you do, report that as well.

Extra Notes

You might be tempted to "help" Pikachu by trying to find some rules about how to fill the bag. For instance, you might think you can try the lightest treasures first, or else sort the treasures by their value-to-weight ratio or something like that.

It turns out that NONE of that will work (at least in the general case). The only way to solve this problem is to look at each treasure one-by-one and ask: "should I include this one or not?" You can simulate this question in your code by making TWO recursive calls in the recursive branch of your function: one call for taking the treasure you're looking at, and one for ignoring it. Then simply return the result for whichever choice worked out better!

What to Hand In

- A document entitled `a8q3.py` containing your recursive function (and the starter code for reading the file)
- A written document entitled `a8q3.pdf` (or .txt, .rtf, .docx) listing the solution that you found by hand for room 1, the results your program finds for all the rooms, and your observations from trying a bigger bag (of size 300) on room 3.

Evaluation

- 3 marks: The recursive function is well written and solves the problem using recursion
  - 0 marks if: The function uses global variables, or otherwise avoids good recursion
- 1 mark: Your written document lists the solution you found by hand for the treasures to take and their total value for room 1
- 1 mark: Your program reports the right answer for all 3 rooms
- 1 mark: Your program reports the right answer for the bigger bag, and you included your obsevations

Question 4 (5 points):

Purpose: To practice recursion on a more interesting problem.

Pokemon are fantastic creatures that can evolve[2] into other, usually stronger, creatures. In their secret laboratory, the evil Team Rocket has been trying to evolve Pokemon in terrifying new ways! Might it now be possible for a meek Magikarp to evolve all the way into the mighty Mewtwo? Your job is to write a program that uses the power of recursion to find out.

For this problem, you will be given a pokemon evolution book as a `dictionary`. The `keys` to this dictionary are pokemon names, and the `value` for each key is a list of pokemon that can be evolved into. The following is a sample of what this dictionary might look like:

```
book = {
    "squirtle" : ["wartortle"],
    "wartortle" : ["blastoise"],
    "blastoise" : [],
    "eevee" : ["flareon", "jolteon", "vaporeon"]
}
```

In the example above, **squirtle** can (eventually) evolve into a **blastoise** by first evolving into a **wartortle**, but never into an **eevee**. On the other hand, **eevee** can evolve into any one of **flareon**, **jolteon** or **vaporeon**. Note that a pokemon that can no longer evolve might be in the book and associated with an empty list (like **blastoise** above) or might not be in the book at all. Either case is fine and your program needs to handle both cases correctly.

You are guaranteed, however, that the pokemon book will be structured such that a pokemon cannot "devolve". Thus, if pokemon A can, either directly or indirectly, evolve into pokemon B, then it will not be possible for B to evolve back into A.

Your task is to write a **recursive function** that will answer the question of whether a `source` pokemon can eventually evolve into a given `target` pokemon. Your function should have 3 parameters: the `source`, the `target` and the pokemon dictionary to use to check for possible evolutions. Your function should return `True` if it is possible for `source` to become the `target` and `False` otherwise.

This might sound a little difficult, but with the power of recursion, this is not a complicated problem. Your function should be **no longer than 12 lines of code** (not counting comments) and possibly less (ours is 8). If you find your solution is getting any longer than that, you are overthinking it!

Note: For this question, you ARE allowed to use loops as part of your recursive function! You will likely use a loop to iterate over the lists that are in the evolution book. But your program should still use recursion to do the "real work" (in fact, this problem would be MUCH more difficult to solve without using recursion at all!!).

Provided Files

We are providing you with 3 sample pokemon books to use; you can find these on the course website. You can just copy/paste the dictionary literals into your code, or just rename the provided file and put your own code within.

[2] The Pokemon franchise was in fact extremely scientifically irresponsible to use this term. Evolution occurs at the population level, not at the individual level. A better word would have been "mutate." But oh well.

Sample Run

If you test your program with the sample code provided, you should be able to get output that looks like this:

```
Using book1, can charmeleon evolve to blastoise?
False
Using book2, can eevee evolve to articuno?
True
Using book3, can magikarp evolve to mewtwo?
True
```

Part 2: Analysis

In the problem description above, we gave you a guarantee about the structure of the pokemon evolution book: that it was not possible for a pokemon to "devolve". What if you DIDN'T have this guarantee? Write a short paragraph (2-to-4 sentences is enough) describing the problem(s) this would cause for your program and how you might possibly attempt to address these problems.

What to Hand In

- A document entitled `a8q4.py` containing your finished program, as described above.
- A file called `a8q4_part2.txt` (.rtf and .pdf are also okay) in which you include your answer to the analysis question above.

Evaluation

- 5 marks for correctly using recursion to solve the problem (with NO global variables)

Question 5 (4 points):

Purpose: To employ effective testing on a challenging problem

Write a **test driver** that includes tests for the `can_evolve()` function that you wrote for the previous question. Use a variety of test inputs for which you can calculate the correct expected output by hand.

Do **not** bother including tests for invalid data types, or for Pokemon evolution books that are improperly formatted. Instead, focus on valid cases that are logically different and that cover a variety of special cases and possible outcomes.

Remember that the evolution book to use is one of the inputs to your function. You will probably want to carefully craft different books for each test case that exhibit the property you're wanting to test at the moment. Where possible, make them simple! Simpler tests are better, because then it is easier for you to be confident that you didn't make any mistakes when crafting the test!

What to Hand In

- Your test driver in `a8q5.py`. It's probably best if this file IMPORTS the functions you are testing, i.e `import a8q4 as PK`

Note that you can get full marks for this question **even if you did not complete the** `pokemon_evolution()` **function.** Indeed, showing when a function does NOT work is the main goal of testing!

Evaluation

- 4 marks: Test cases are adequate and show reasonable thought

Question 6 (3 points):

All programs that you submit for this assignment will be assessed with regard to their **style and readability**. Make sure that all of your code:

- includes your name, nsid, and student number as comments
- uses informative variable names
- makes effective use of formatting and white space
- includes a `docstring` for all **function definitions** in the program
- for longer programs, uses helpful and concise single-line comments where appropriate

What to Hand In

There is nothing to hand in for this question. It's simply describing how the rest of your work will be evaluated with regard to style and readability. It is possible to earn these marks even if you don't hand in every other question in this assignment, but you must hand in enough work such that your style and readability can be adequately assessed.

Evaluation

- 3 marks for consistently following the principles listed above

Question 7 (2 points):

In your submission for this assignment, you have an opportunity to earn something we'll call "the wow factor". To earn the wow factor, you must do something creative that goes beyond the expectations laid out in the assignment. This could be an extra feature for one of the previous questions, or it could be an extra little program that you submit that builds on the skills used in the assignment.

You should not aim to do a lot of work for this, but should instead demonstrate in a simple and effective way the depth of your understanding. In short, take ownership of your work and "impress us".

Telling you exactly what to do to earn "the wow factor" would defeat the entire purpose, but here are some very general ideas.

- improve the aesthetic appeal of your program(s)
- improve the user-friendliness of your program(s)
- remove a simplifying assumption and handle the resulting added complexity
- compare two different ways of doing the same task, and submitting data/analysis on which was better
- just doing something generally cool

One quick warning: using fancy extra code libraries in order to AVOID a key learning objective from the assignment is not impressive, it's actually anti-impressive. Focus instead on solving problems using simple clean foundational programming skills. Using extra libraries is fine if they let you do something BEYOND what the assignment asked, but not if they're replacing a core assigned task.

What to Hand In

A plain text file called `wow.txt` describing what you did that you think merits the wow factor. One or two sentences is enough. If you create any additional code files, hand those in too.

Evaluation

- 2 marks for something impressive