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 3

Conditional Branching and Simple Repetition

Date Due: 5:00pm Friday Oct 11 **Total Marks: 22**

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 (3 points):

Purpose: To practice slicing and using string methods

An **email address** is a sequence of characters with the following structure:

- it contains exactly one @ character
- there are 1 or more characters before the @. We call these characters the **username**
- There is at least one period (`.`) SOMEWHERE after the @
- IMMEDIATELY after the @ there are one or more characters that are NOT periods. We call all the characters in between the @ and the first-period-after-the-@ the **domain**

For this question, you will write a program that lets the user type in a valid email address, and reports the **username** and **domain** components of that address. Your program may assume that the user always types a correctly-formatted email address.

Hint: Tools such as slicing and the string method `.find()` will probably be the most useful here.

Sample Runs

Here is one possible sample run. Green text was entered by the user, and the blue text is based on that input.

```
Enter email address: ash@pokemon.kanto.jp
Username: ash
Domain: pokemon
```

Here is another one.

```
Enter email address: jeff.long@usask.ca
Username: jeff.long
Domain: usask
```

What to Hand In

Hand in your solution in a file called `a3q1.py`.

Evaluation

- 3 marks for correctness

Question 2 (5 points):

Purpose: To practice using modules and simple conditionals

For this question, you'll write a program that checks whether a randomly generated integer is a perfect square. Recall that a perfect square is an integer for which the square root is also an integer.

First, your program should import both the `random` and `math` modules. Make sure to use the import syntax as shown in the course readings.

Then, your program should generate a random number between 1 and 10000 (inclusive).

Finally, your program should display that number to the console. If the number is a perfect square, an additional message should be displayed to say so. Use a conditional statement along with any functions from the `math` module that you think you need to do this.

Sample Run

Here is a possible sample run (output value of a variable is shown in blue text):

```
Your random number is: 5062
```

Here is another one.

```
Your random number is: 961
Amazing!  961 is a perfect square!
```

What to Hand In

- A file called `a3q2.py` containing your finished program, as described above.

Evaluation

- 5 marks for:
  - correct import statements
  - random number generation in the correct range
  - correctly identifying perfect squares

Question 3 (6 points):

Purpose: To practice chained if-elif-else constructs.

Pokemon are fantastic creatures that often battle other pokemon. Furthermore, pokemon are categorized into "types", which has a big impact on their effectiveness in battle. A "fire" pokemon will work very well against a "grass" pokemon, but not so well against a "water" pokemon.

There are a lot of different types of pokemon, so keeping track of when each type is effective can be confusing! For this question, we'll write a program to help us determine when a pokemon is effective. We'll limit ourselves to the 6 pokemon types shown in this chart.[1]

Each row in the chart matches the type of the Attacking pokemon, while the columns match the type of the Defending pokemon. The attack is considered "super effective" wherever we seen a green 2 in the chart. For example, Fire is "super effective" against Grass. Wherever we see a red $1/2$, the attack is considered "not very effective". For example, Water is "not very effective" against Grass. If the cell is blank, then we will simply call the attack "okay". For example, Electric is "okay" against Ice, and Normal is "okay" against everything.

Effectiveness Function

First, write a function to encode the information represented in the type effectiveness table. Your function should have two parameters: the type of the attacking pokemon and the type of the defending pokemon. The function should return one of three different strings:

- `"super effective"`
- `"okay"`
- `"not very effective"`

Use conditional statements to implement the rules in the table in order to return the correct string.

A word of warning: if you find yourself writing 36 different if-statements (one for each attack-defense pair), stop. There is a better way. Use nesting and the `else` keyword to take advantage of patterns or structure in the table, and you will end up with much simpler and more readable code.

Main program

Your main program should ask the user for the types of the two pokemon, call your function from the previous section to get a result, and print out a message indicating that result. Also, if the attack is `"super effective"`, the message should end with a bunch of exclamation points (because super effective attacks are exciting!).

[1] This chart is incomplete and there are many more types, but it is already big enough for the purpose of an exercise.

Sample Run

Here is a sample run, with console input shown in green text.

```
Type of Attacking Pokemon: fire
Type of Defending Pokemon: water
Using fire against water is not very effective.
```

Here is another.

```
Type of Attacking Pokemon: electric
Type of Defending Pokemon: water
Using electric against water is super effective!!!
```

What to Hand In

- A file called `a3q3.py` containing your finished program, as described above.

Evaluation

- 4 marks for correct behaviour (determining correct outcome, printing message with exclamations if needed)
- 1 mark for correct design (a correct function interface and interaction with the main program)
- 1 mark for identifying reasonable logical patterns to craft a good conditoinal

Question 4 (3 points):

Purpose: To practice a simple loop and simple conditionals

For this question, you'll write a simple program that makes the user try to guess a secret number repeatedly until they get it right.

First, your program should select a random number between 1 and 10 (inclusive). You can use the `randint()` method from the `random` module to do this. This is the secret number that the user is trying to guess.

Then, your program should allow the user to input numbers as many times as needed in order to guess the secret number. If the user gets it wrong, the program should give a hint by telling the user whether their guess was too high or too low. If the user gets it right, the program should tell them so and then terminate.

Sample Run

Sample input and output (input typed by the user is shown in green text):

```
Guess a number from 1 to 10! 5
Too low!
Guess again: 7
You got it, the secret number was 7
```

What to Hand In

- A file called `a3q4.py` containing your finished program, as described above.

Evaluation

- 3 marks for:
  - Generating a number in the correct range
  - Correct loop behaviour allowing re-guessing
  - Correct hints

Question 5 (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 6 (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