CMSC 201 Section 40

Homework 2 - Conditionals

Due Date: Monday, February 28th, 2022 by 11:59:59 PM
Value: 40 points

This assignment falls under the standard cmsc201 academic integrity policy. This means you should not discuss/show/copy/distribute your solutions, your code or your main ideas for the solutions to any other student. Also, you should not post these problems on any forum, internet solutions website, etc.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
"""
File:     FILENAME.py
Author:   YOUR NAME
Date:     THE DATE
Section:  YOUR DISCUSSION SECTION NUMBER
E-mail:   YOUR_EMAIL@umbc.edu
Description:
  DESCRIPTION OF WHAT THE PROGRAM DOES
"""
```

Creating Your HW2 Directory

```
linux3[1]% cd cmsc201/Homeworks
linux3[2]% mkdir hw2
linux3[3]% cd hw2
linux3[4]%
```

Submission Details

Submit the files under the following titles:
(These are case sensitive as usual.)
submit cmsc201 HW2 {files go here}

Problem 1 - Will it Float	will_it_float.py
Problem 2 - Harry Potter	harry_potter.py
Problem 3 - Leap Year	leap_year.py
Problem 4 - Knobs and Switches	knobs_and_switches.py
Problem 5 - Which Month	which_month.py

For example you would do:

```
linux1[4]% submit cmsc201 HW2 will_it_float.py
harry_potter.py leap_year.py knobs_and_switches.py
which_month.py
Submitting will_it_float.py...OK
Submitting harry_potter.py...OK
Submitting leap_year.py...OK
Submitting knobs_and_switches.py...OK
Submitting which_month.py...OK
linux1[5]%
```

From here, you can use **`emacs`** to start creating and writing your different Homework 2 Python programs.

You don't need to and should not make a separate folder for each file. You should store all of the Homework 2 files in the same **`hw2`** folder.

Coding Standards

Coding standards for CMSC 201 can be found here.

For now, you should pay special attention to the sections about:

- Naming Conventions
- Use of Whitespace
- Comments (specifically, File Header Comments)
- Variable names.

Input Validation

For this assignment, you do **not** need to worry about **SOME** “input validation.”

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

Unlike in HW1 you didn't have to worry about any input validation since you didn't have if statements, but now you do, so you can worry a little about it. You can try to prevent invalid input, but only when that invalid input is of the correct type.

For example, if your program asks the user to enter a whole number, it is acceptable if your program crashes if they enter something else like “dog” or “twenty” or “88.2” instead.

But it's a good idea to try to catch a zero division error for instance, or entering negative numbers when only positive numbers are allowed.

Here is what that error might look like:

```
Please enter a number: twenty
Traceback (most recent call last):
  File "test_file.py", line 10, in <module>
    num = int(input("Please enter a number: "))
ValueError: invalid literal for int() with base 10: 'twenty'
```

Allowed Built-ins/Methods/etc

- Declaring and assigning variables, ints, floats, bools, strings.
- Casting int(x), str(x), float(x), (technically bool(x))
- Using +, -, *, /, //, %, **; +=, -=, *=, /=, //=, %=, **= where appropriate
- Print, with string formatting, with end= or sep=:
    - '{}'.format(var), '%d' % some_int, f-strings
    - Really the point is that we don't care how you format strings in Python
    - Ord, chr, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.
- Using the functions provided to you in the starter code.
- Comparisons ==, <=, >=, >, <, !=, in
- Logical and, or, not
- if/elif/else, nested if statements
- Using import with libraries and specific functions **as allowed** by the project/homework.
- String Operations:
    - upper(), lower()
    - concatenation +, +=

Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

- For loops, both *for i* and *for each* type.
- While loops
    - sentinel values, boolean flags to terminate while loops
- Lists, list(), indexing, i.e. my_list[i] or my_list[3]
    - 2d-lists if you want them/need them my_2d[i][j]
    - Append, remove
    - **list slicing**
- If you have read this section, then you know the secret word is: catenary.
- String operations, split(), strip(), join(), isupper(), islower()
    - **string slicing**
- **Dictionaries**
    - creation using dict(), or {}, copying using dict(other_dict)
    - .get(value, not_found_value) method
    - accessing, inserting elements, removing elements.
- break, continue
- methods outside those permitted within allowed types
    - for instance str.endswith
    - list.index, list.count, etc.
- Keywords you definitely don't need: await, as, assert, async, class, except, finally, global, lambda, nonlocal, raise, try, yield
- The *is* keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as ==).
- built in functions: any, all, breakpoint, callable, classmethod, compile, exec, delattr, divmod, enumerate, filter, map, max, min, isinstance, issubclass, iter, locals, oct, next, memoryview, property, repr, reversed, round, set, setattr, sorted, staticmethod, sum, super, type, vars, zip

- If you have read this section, then you know the secret word is: cowboy.
- exit() or quit()
- If something is not on the allowed list, not on this list, then it is probably forbidden.
- The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.

Problem 1 - Will it Float

A pressing question is whether an object will float when put into water. The density of water is 1000 kg/m^3, which means anything more dense than this will sink, anything less dense will float, and anything of equal density will have neutral buoyancy.

Write a program in a file called will_it_float.py which determines whether an object will float, sink, or have neutral buoyancy.

1) Ask the user first what the name of the object is.
2) Then ask for the weight of the object.
3) Ask the user for the volume of the object.
4) Test whether the object's density (weight/volume) is less, greater, or equal to 1000.
5) Tell the user whether the object will float, have neutral buoyancy, or sink.

Sample Output:

```
linux3[9]% python3 will_it_float.py
What object are we putting in the water today? rubber
ducky
What is the weight of the object? 0.01
What is the volume of the object? 0.00001
rubber ducky will float

linux3[10]% python3 will_it_float.py
What object are we putting in the water today?
concrete block
What is the weight of the object? 2400
What is the volume of the object? 1
concrete block will sink.

linux3[11]% python3 will_it_float.py
What object are we putting in the water today?
Submarine
What is the weight of the object? 10000
What is the volume of the object? 10
Submarine has neutral buoyancy.
```

Problem 2 - Harry Potter

There are far too many characters in Harry Potter to come close in a single homework problem, but we'll try to make a 20-questions-like program in a file called harry_potter.py which will determine which character you are from a very restricted group.

Use the following questions to determine which character 'you are.'

1) Ask if the person is a student at Hogwarts.
    a) If yes, then ask if they have a scar?
        i) If yes, then they are Harry Potter.
    b) If no, then ask if they have red hair.
        i) If yes, then they are a Weasley.
        ii) If no, then they are Hermoine Granger.
2) If not then ask if they have a beard.
    a) If yes, then ask if they are part giant.
        i) If yes, then they are Rubius Hagrid.
        ii) If no, then they are Albus Dumbledore
    b) If no, then ask if they give off evil vibes.
        i) If yes, then they are Severus Snape.
        ii) If no, Minerva McGonagall.
3) If no again, then they are Sirius Black.

All answers to the questions should be "yes" or "no." You are permitted to use .lower() to ensure that "Yes" "YES" "No" "nO" would all be admissible but it isn't necessary. We will only test with "yes" and "no."

Sample Runs:

```
linux4[116]% python3 harry_potter.py
Are you a student at Hogwarts? yes
Do you have a scar? no
Do you have red hair? yes
Red hair, vacant expression... You must be a Weasley.

linux4[117]% python3 harry_potter.py
Are you a student at Hogwarts? no
Are you or have you ever been a teacher at Hogwarts? yes
Do you have a beard? no
Do you give off evil vibes, but have a good heart in the
end? yes
You are Severus Snape.

linux4[118]% python3 harry_potter.py
Are you a student at Hogwarts? no
Are you or have you ever been a teacher at Hogwarts? no
You are Sirius Black.

linux4[119]% python3 harry_potter.py
Are you a student at Hogwarts? yes
Do you have a scar? yes
You are Harry Potter

linux4[120]% python3 harry_potter.py
Are you a student at Hogwarts? no
Are you or have you ever been a teacher at Hogwarts? yes
Do you have a beard? yes
Are you part giant? no
You are Albus Dumbledore
```

Problem 3 - Leap Year

The rules for leap years work this way:

1) If the year is divisible by 4, then it is a leap year, unless:
2) If the year is divisible by 100, then it is not a leap year, unless:
3) If the year is divisible by 400, then it is again a leap year.
4) If the year is not divisible by 4, it is not a leap year.

For example, 1996 is a leap year, as is 2004, but 1900 wasn't. 2000, 2400, 1600 are all leap years because they are divisible by 400. Years like 2009, 2011, 1933 are all not leap years.

Write a program in a file called leap_year.py which determines if a given year (as an integer) is a leap year.

You don't have to worry about negative years.

Sample output:

User input is generally colored **blue** to help distinguish it from the rest of the text.

```
linux4[120]% python3 leap_year.py
Enter a year: 2000
It is a leap year.

linux4[121]% python3 leap_year.py
Enter a year: 2005
It is not a leap year.

linux4[122]% python3 leap_year.py
Enter a year: 2100
It is not a leap year.

linux4[123]% python3 leap_year.py
Enter a year: 1996
It is a leap year.
```

Problem 4 - Knobs and Switches

Name your file knobs_and_switches.py.

You are playing a role playing game (RPG) on your computer and find out that there's a puzzle to be solved. There is a door with two knobs and a switch.

There are two knobs with 12 positions each (1... 12) and a switch which can be toggled "up" or "down".

- In order to open the door, the first knob must be an even position, and the second knob must be in an odd position. The switch must be flipped to up. If this is all done, then print "The door opens, you get all the loot."
- If one of the knobs is flipped to its correct position (even or odd), or both knobs are correct but the switch is down, then "The door clanks but does not open, try again."
- If none of the knobs are correct, then you print "The handle doesn't budge."

Make sure you take in the knob positions first, and then the switch. Consider error conditions like the last sample output, where the user enters jibberish or invalid input.
(Keep in mind that all of the input will be of the correct type, that is to say, if you ask for an integer position, we may enter -17 but we will not enter
"rabbit".)

```
linux3[6]% python3 knobs_and_switches.py
What is the position of the first knob? (Enter 1-12) 3
What is the position of the second knob? (Enter 1-12) 5
What is the position of the switch? (Enter up or down)
down
The door clanks but does not open, try again.

linux3[7]% python3 knobs_and_switches.py
What is the position of the first knob? (Enter 1-12) 4
What is the position of the second knob? (Enter 1-12) 7
What is the position of the switch? (Enter up or down) up
The door opens, you get all the loot.

linux3[8]% python3 knobs_and_switches.py
What is the position of the first knob? (Enter 1-12) -2
What is the position of the second knob? (Enter 1-12) 3
What is the position of the switch? (Enter up or down) up
Knob 1 needs to be set to 1 - 12

linux3[9]% python3 knobs_and_switches.py
What is the position of the first knob? (Enter 1-12) 5
What is the position of the second knob? (Enter 1-12) 2
What is the position of the switch? (Enter up or down)
down
The handle doesn't budge.
```

Problem 5 - Which Month

Name your file which_month.py

Month	Number
January	1
February	2
March	3
April	4
May	5
June	6
July	7
August	8
September	9
October	10
November	11
December	12

First ask what month it is "now" (m) and then ask how many months into the future you want to go (n). These should both be integers. Then display what month it is in the future n months after m.

Display the answer as the actual name of the month. The number of months after the start can be more than 12. [Hint: use mod]

Check to see if the first input is between 1 and 12 before continuing.

Sample Output:

```
linux3[14]% python3 which_month.py
What month are we starting in (enter as an int)? 11
How many months in the future should we go? 24
The month will be November

linux3[15]% python3 which_month.py
What month are we starting in (enter as an int)? 5
How many months in the future should we go? 1
The month will be June

linux3[16]% python3 which_month.py
What month are we starting in (enter as an int)? 8
How many months in the future should we go? 71
The month will be July

linux3[17]% python3 which_month.py
What month are we starting in (enter as an int)? 3
How many months in the future should we go? 9
The month will be December

linux3[18]% python3 which_month.py
What month are we starting in (enter as an int)? 3
How many months in the future should we go? 10
The month will be January

linux3[19]% python3 which_month.py
What month are we starting in (enter as an int)? 17
That is not a month between 1 and 12.
```

More Submission Details

How to submit is covered in detail in Homework 0.

Once each or all of your files are complete, it is time to turn them in with the **submit** command. (You may also turn in individual files as you complete them. To do so, only **submit** those files that are complete.)

You must be logged into your account on GL, and you must be in the same directory as your Homework 2 Python files. To double-check you are in the directory with the correct files, you can type **ls**.

```
linux1[3]% ls
leap_year.py                will_it_float.py
knobs_and_switches.py       harry_potter.py
which_month.py
linux1[4]%
```

You can see what files were successfully submitted by running:
`submitls cmsc201 HW2`

For more information on how to read the output from **`submitls`**, consult with Homework 0. Double-check that you submitted your homework correctly, since **empty files will result in a grade of zero.**

Advice for Submitting Early and Often:

You can also submit one or two files at a time, which means you can submit a file without submitting the rest of the assignment. It's better to miss one part than an entire assignment. Do not wait to submit everything until five minutes before the due date.

```
linux1[4]% submit cmsc201 HW2 monster_battles.py
circuit_diagram.py
Submitting monster_battles.py...OK
Submitting circuit_diagram.py...OK
linux1[5]%
```

If you're re-submitting, the system will ask that you confirm you want to overwrite each file; make sure that you confirm by typing "y" and hitting enter if you want to do so.

```
linux1[5]% submit cmsc201 HW2 monster_battles.py
It seems you have already submitted a file named
monster_battle.py.
Do you wish to overwrite? (y/n):
y

linux1[6]%
```