Homework 2 - Conditionals

Due Date: Monday, September 26, 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 - Wind Speed - Part 1	wind1.py
Problem 2 - Wind Speed - Part 2	wind2.py
Problem 3 - Grading - Part 1	grades1.py
Problem 4 - Grading - Part 2	grades2.py
Problem 5 - Get my bone$!	bones.py
Problem 6 - Which Month	which_month.py

For example you would do:

```
linux1[4]% submit cmsc201 HW2 wind1.py wind2.py grades1.py
grades2.py which_month.py
Submitting wind1.py...OK
Submitting wind2.py...OK
Submitting grades1.py...OK
Submitting grades2.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 - Wind Speed - Part 1

The National Weather Service has 5 categories to describe hurricanes, shown in the following chart:

Category	Wind Speed
1 – Minimum	74 - 95 mph
2 – Moderate	96 - 110 mph
3 – Extensive	111 - 130 mph
4 – Extreme	131 - 155 mph
5 - Catastrophic	greater than 155 mph

Write a program in a file called "wind1.py" that will get the current ***INTEGER*** wind speed value from the user and will print out if this is a hurricane and which category number it is.

1. Ask the user first for the current wind speed
2. Program then displays the category.

Sample Output:

```
linux3[9]% python3 wind1.py
What is the current wind speed? 123
The wind's speed category is Extensive

linux3[10]% python3 wind1.py
What is the current wind speed? 95
The wind's speed category is Minimum
```

Problem 2 - Wind Speed - Part 2

Using the same info from Part 1, write a program in a file called "wind2.py" that will get the current ***FLOAT*** wind speed value from the user and will print out if this is a hurricane and which category number it is. Note: This time the chart would have some "holes" for legitimate values that could be entered! For example, what would happen if a user inputs 95.5?

Category	Wind Speed
1 – Minimum	74 < 96 mph
2 – Moderate	96 < 111 mph
3 – Extensive	111 < 131 mph
4 – Extreme	131 < 155 mph
5 - Catastrophic	greater than 155 mph

1. Ask the user first for the current wind speed
2. Program then displays the category.

Sample Runs:

```
linux3[93]% python3 wind2.py
What is the current wind speed? 123.23
The wind's speed category is Extensive

linux3[94]% python3 wind2.py
What is the current wind speed? 95.9
The wind's speed category is Minimum
```

Problem 3 - Grading - Part 1

There are some universities in this country that use the +/- system for final grades. Write a program "grades1.py" that accepts a numeric grade (float) from (0 –100), then displays if that grade would be an:

A+	96 – 100
A	93 – 95.999999
A-	90 – 92.999999
B+	86 – 89.999999
B	83 – 85.999999
B-	80 – 82.999999
C+	76 – 79.999999
C	73 – 75.999999
C-	70 – 72.999999
D+	66 – 69.999999
D	63 – 65.999999
D-	60 – 62.999999
F	59.99999 and less

MAKE SURE YOU USE IF/ELSE statements!!! NOT ALL Ifs.

Sample output:

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

```
linux4[120]% python3 grades1.py
Please enter the grade: 79.6
The grade would be: C+

linux4[121]% python3 grades1.py
Please enter the grade: 41.6
The grade would be: F
```

Problem 4 - Grading - Part 2

Validation is important. The topic will come up later this semester. Copy and paste grades1.py code and name it grades2.py. Using the same code, add code that will simply display “Bad user input”, if the user types in a value greater than 100, and less than 0.

AGAIN, MAKE SURE YOU USE IF/ELSE statements!!! NOT ALL Ifs.

Sample output:

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

```
linux4[120]% python3 grades2.py
Please enter the grade: 79.6
The grade would be: C+

linux4[121]% python3 grades2.py
Please enter the grade: 41.6
The grade would be: F

linux4[122]% python3 grades2.py
Please enter the grade: -45.6
Bad user input

linux4[123]% python3 grades2.py
Please enter the grade: 141.6
Bad user input
```

Problem 5 - Get my bone$!

Write a program “bones.py” to compute and display a person’s weekly salary:

If the hours (float) worked are less than or equal to 40, the person receives $12.50 per hour; otherwise, the person receives $500.00 plus $18.75 for each hour worked over 40 hours.

The program should request the hours worked as input and display the salary as output. Notice your output should be formatted and should display 2 decimal places.

```
linux4[12]% python3 bones.py
Please enter the hours worked this week: 20
Your paycheck for this week will be: $250.00

linux4[13]% python3 bones.py
Please enter the hours worked this week: 40
Your paycheck for this week will be: $500.00

linux4[14]% python3 bones.py
Please enter the hours worked this week: 45
Your paycheck for this week will be: $593.75
```

Problem 6 - 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
wind1.py            grades1.py      wind2.py
grades2.py          bones.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]%
```