CMSC 201, Section 40
Spring 2020
Homework 2 – Decisions

Assignment: Homework 2 – Decisions
Due Date: Monday, February 17, 2020 by 11:59:59 PM
Value: 40 points

Collaboration: For Homework 2, collaboration is **NOT** allowed.

File Header Comment: Make sure that you have a **complete file header comment at the top of <u>each file</u>**, and that all the information is correctly filled in.

```
"""
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
"""
```

Instructions

For each of the questions below, you are given a program that you must solve or a task you must complete. For this exercise, you will need to use concepts previously practiced in Homework 1, as well as concepts covered in class during the last week.

You should already be familiar with variables, expressions, **`input()`**, casting to an integer or float, and **`print()`**. You will also need to use one-way and two-way decision structures, as well as nested decision structures. You may also need to use multi-way decision structures.

Think carefully about what the overall goal of the algorithm is before you begin coding.

At the end, your Homework 2 files must run without any errors.

Additional Instructions – Creating the hw2 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

Just as you did for Homework 1, you should create a directory to store your Homework 2 files. We recommend calling it **`hw2`** and creating it inside the **`Homeworks`** directory inside your **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1. (You don't need to make a separate folder for each file. You should store all Homework 2 files in the same **`hw2`** folder.)

Coding Standards

Prior to this assignment, you should read the Coding Standards linked on the course website on the "Assignments" page.

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

- Naming Conventions
- Use of Whitespace
- Comments (specifically, File Header Comments)
- Line Length

Additional Specifications

For this assignment, **you must use** `if __name__ == '__main__':` as discussed in class.

For this assignment, you do not need to worry about any "input validation."

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

If the user enters "bogus" data (for example: a negative value when asked for a positive number), this is acceptable. (Your program does not need to worry about correcting the value or fixing it in any way.)

For most of the problems below, we do ask you to handle "bogus" data gracefully by displaying something like "invalid option".

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.

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'
```

Questions

Each question is worth the indicated number of points. Following the coding standards is worth 4 points. If you do not have complete file header comments and correctly named files, you will lose points.

Question 1:

`hw2_part1.py` **(Worth 8 points)**

You are writing a botanical classification system. It will ask questions about a plant and determine what genus that plant is.

Asteraceae: Has over 7 petals, multiple flowers, and are <= 300 cm tall
Helianthus annuus: Has over 7 petals, has multiple flowers, and are taller than 300 cm
Iris: Has 3 petals and a single flower
Orchidaceae: Has 5 petals and a single flower
Nepenthes: Eats bugs, uses a pitfall trap
Dionaea muscipula: Eats bugs, uses snap jaws

Your program should NOT have to ask the user more than four questions to pick the answer. If the user puts information for a flower not mentioned above, your program will say that it doesn't know about that flower.

Your output does not need to match the provided output exactly, but you should include the plant name in the answer when it is given.

```
linux3[14]% python3 hw2_part1.py
Does the plant eat bugs? no
How many petals does this thing have? 4
I don't know what that is...

linux3[15]% python3 hw2_part1.py
Does the plant eat bugs? yes
Does the plant use a pitfall trap or snap jaws? snap
jaws
Watch those fingers!  You got a Dionaea muscipula: the
venus fly trap!
```

```
linux3[15]% python3 hw2_part1.py
Does the plant eat bugs? yes
Does the plant use a pitfall trap or snap jaws?
pitfall trap
Oh, interesting.  That is a Nepenthes.  Don't drink
from a pitcher plant.

linux3[15]% python3 hw2_part1.py
Does the plant eat bugs? no
How many petals does this thing have? 18
Does it have multiple flowers?yes
How many cm tall is it? 25
Hmm, that is some kind of Asteracea.  Premium specimen
of photosynthetic evolution!

linux3[15]% python3 hw2_part1.py
Does the plant eat bugs? no
How many petals does this thing have? 3
Does it have multiple flowers? no
Irises are nice, and that's what you have.

linux3[15]% python3 hw2_part1.py
Does the plant eat bugs? no
How many petals does this thing have? 5
Does it have multiple flowers? no
Oh, you have an Orchidaceae, known by its friends as
an Orchid.

linux3[15]% python3 hw2_part1.py
Does the plant eat bugs? no
How many petals does this thing have? 12
Does it have multiple flowers? yes
How many cm tall is it? 402
Oh, that is a Helianthus annuus, or sunflower for the
uninitiated.
```

Question 2:

`hw2_part2.py` **(Worth 5 points)**

For this program, create a (very simplified) day of the week calculator. Ask the user to enter the day of the month and respond with the correct day of the week.

The program will assume that the month starts on Tuesday and has 31 days (just like the month of October in 2019). The program

- Can assume that the number entered will be an integer
- Cannot assume that the number will be valid!

If the day of the month the user entered is not a valid day of the month (less than 1 or greater than 31), simply print a short error message to the user. Otherwise, print the day of the week that day falls on. For instance, the 2nd would be a Wednesday, the 11th would be a Friday, etc.

IMPORTANT: Do not write a case for each day of the month. If your program uses dozens of individual **if**, **elif**, or **else** statements, you will lose significant points.

(HINT: There is a mathematical operator in Python that will allow you to write the program without the need to have dozens of individual decision statements.)

(See next page for output.)

Here is some sample output, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
linux3[16]% python3 hw2_part2.py
Please enter a day of the month: 12
Today is a Saturday!

linux3[17]% python3 hw2_part2.py
Please enter a day of the month: 1
Today is a Tuesday!

linux3[18]% python3 hw2_part2.py
Please enter a day of the month: 20
Today is a Sunday!

linux3[19]% python3 hw2_part2.py
Please enter a day of the month: 0
The date 0 is an invalid day.

linux3[20]% python3 hw2_part2.py
Please enter a day of the month: 28
Today is a Monday!
```

Question 3:

`hw2_part3.py` **(Worth 8 points)**

This program plays a (very) simple game that tries to guess elements of the periodic table based on data from the user. For simplicity's sake, there are only 5 possible elements.

(WARNING: *This part of the homework is the most challenging, so budget plenty of time and brain power. And read the instructions carefully!)*

The program can ask the player about four characteristics. It should ask the ***minimum*** number of questions needed to guess the element. *(HINT: It should ask* ***no less*** *than two questions and* ***no more*** *than three questions to find the right element)*.

- Are there more than 4 valence electrons?
- Does it have an s orbital?
- Is it solid at room temperature?
- Is it a noble gas?

For these inputs, the program can assume the following:

- The user will only ever enter either lowercase **y** (for "yes") or lowercase **n** (for "no").

Based on the user's responses, the program must select the correct element and print it out to the screen. Here are the possible elements:

- Element has less than 4 valence electrons and has an s orbital: Helium
- Element has less than 4 valence electrons, does *not* have an s orbital: Aluminum
- Element has more than 4 valence electrons and is a noble gas: Argon
- Element has more than 4 valence electrons and is *not* a noble gas, and is a gas at room temperature : Nitrogen
- Element has more than 4 valence electrons, is *not* a noble gas, and is a solid at room temperature: Potassium

(*See next page for sample output.)*

Here is some sample output, with the user input in **blue**.

(Yours does not have to match this word for word, but it should be similar.)

```
linux3[21]% python3 hw2_part3.py
Does it have more than 4 valence electrons? y
Is it a noble gas? y
It must be Argon!

linux3[22]% python3 hw2_part3.py
Does it have more than 4 valence electrons? y
Is it a noble gas? n
Is it a solid at room temperature? n
It must be Nitrogen!

linux3[23]% python3 hw2_part3.py
Does it have more than 4 valence electrons? y
Is it a noble gas? n
Is it a solid at room temperature? y
It must be Potassium!

linux3[24]% python3 hw2_part3.py
Does it have more than 4 valence electrons? n
Does it have an s orbital? n
It must be Aluminum!

linux3[25]% python3 hw2_part3.py
Does it have more than 4 valence electrons? n
Does it have an s orbital? y
It must be Helium!
```

Question 4:

`hw2_part4.py` **(Worth 8 points)**

This program will combine elements (it magically knows how many of each element) that the user provides to create a new compound. The only issue is that this program only knows how to combine 3 elements: ***carbon***, ***hydrogen,*** and ***oxygen***.

Depending on the elements the user provides, print out one of three responses:

- If they entered the same element twice:
  - Print out That's boring, you can't mix <ELEMENT> with itself!
    (*where <ELEMENT> is the element they entered*)
- If they entered one of the known elements (hydrogen, carbon, oxygen):
  - Print out Mixing <ELEMENT1> and <ELEMENT2> makes <MIXED>.
    (*where <MIXED> is the compound created*)
    - Carbon and oxygen make carbon dioxide
    - Hydrogen and oxygen make hydrogen peroxide
    - Hydrogen and carbon make carbon-hydrogen
- If they enter anything else:
  - Print out Mixing <ELEMENT1> and <ELEMENT2> makes an unknown compound.

Your program only needs to work with lowercase letters; you do not need to worry about handling capitalization.

*(HINT: Do **not** start coding this part without having a plan! The order the elements are entered in should not make a difference, so think carefully about what your conditionals should be and how they should be nested.)*

(See next page for sample output.)

Here is some sample output, with the user input in **blue**.

(Yours does not have to match this word for word, but it should be similar.)

```
linux3[7]% python3 hw2_part2.py
Please enter the first element: hydrogen
Please enter the second element: hydrogen
That's boring, you can't mix hydrogen with itself!

linux3[8]% python3 hw2_part2.py
Please enter the first element: hydrogen
Please enter the second element: oxygen
Mixing hydrogen and oxygen make water.

linux3[9]% python3 hw2_part2.py
Please enter the first element: oxygen
Please enter the second element: hydrogen
Mixing oxygen and hydrogen make water.

linux3[10]% python3 hw2_part2.py
Please enter the first element: carbon
Please enter the second element: helium
Mixing carbon and helium make an unknown compound.
```

Question 5

`hw2_part5.py` **(Worth 7 points)**

This program simulates a very simple calculator, which only performs multiplication, division, integer division, and modulus.

The user first enters two integers to use, then selects which operation to perform. The options are "***multiply***", "***divide***", "***int divide***", and "***mod***".

When done, the program prints the mathematical equation, along with the answer. For example, if the user enters integers 2 and 3, then chooses "***multiply***", the program would print out **2 * 3 = 6**.

However, regardless of the order the numbers are entered in, the program always:

- Divides the smaller number by the larger number
  - Creating a result between 0 and 1
- Int division divides the larger number by the smaller number
  - If it were the other way around, the result would always be 0

If the user selects an invalid choice of operation, they receive an error.

(See next page for sample output.)

Here is some sample output, with the user input in **blue**.
(Yours does not have to match this word for word, but it should be similar.)

```
linux3[11]% python3 hw2_part3.py
Please enter the first number: 34
Please enter the second number: 5
The options are multiply, divide, int divide and mod.
What would you like to do? multiply
34 * 5 = 170

linux3[12]% python3 hw2_part3.py
Please enter the first number: 15
Please enter the second number: 80
The options are multiply, divide, int divide and mod.
What would you like to do? divide
15 / 80 = 0.1875

linux3[13]% python3 hw2_part3.py
Please enter the first number: 15
Please enter the second number: 80
The options are multiply, divide, int divide and mod.
What would you like to do? int divide
80 // 15 = 5

linux3[14]% python3 hw2_part3.py
Please enter the first number: 3019301
Please enter the second number: 34
The options are multiply, divide, int divide and mod.
What would you like to do? mod
3019301 % 34 = 33

linux3[15]% python3 hw2_part3.py
Please enter the first number: 4
Please enter the second number: 193
The options are multiply, divide, int divide and mod.
What would you like to do? add
Invalid option...
```

Submitting

Once your **hw2_part1.py**, **hw2_part2.py**, **hw2_part3.py**, **hw2_part4.py**, and **hw2_part5.py** 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**.

```
[mckris2@linux2 hw2]ls
hw2_part1.py  hw2_part2.py  hw2_part3.py  hw2_part4.py  hw2_part5.py
```

To submit your Homework 2 Python files, use the **submit** command, where the class is **cs201s**, and the assignment is **HW2**. Type in (all on one line, even if it wraps around the screen) **submit cs201s HW2 hw2_part1.py hw2_part2.py hw2_part3.py hw2_part4.py hw2_part5.py** and press enter.

```
[[mckris2@linux2 hw2]submit cs201s HW2 hw2_part1.py hw2_part2.py hw2_part3.py
hw2_part4.py hw2_part5.py
Submitting hw2_part1.py...OK
Submitting hw2_part2.py...OK
Submitting hw2_part3.py...OK
Submitting hw2_part4.py...OK
Submitting hw2_part5.py...OK
[mckris2@linux2 hw2]
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can check that your homework was submitted by following the directions in Homework 0 regarding the **submitls** command. Double-check that you submitted your homework correctly, since **an empty file will result in a grade of zero for this assignment.**