# Homework 2 - Conditionals
# CMSC 201 Section 60

**Due Date:** Monday, September 25th, 2023 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 - Creature Combat | creature_combat.py |
| Problem 2 - Gandalf 20 Questions | gandalf.py |
| Problem 3 - Leap Year | leap_year.py |
| Problem 4 - Knobs and Switches | knobs_and_switches.py |
| Problem 5 - Day of the Week | day_of_the_week.py |

For example you would do:

```
linux1[4]% submit cmsc201 HW2 creature_combat.py
gandalf.py leap_year.py tricky_lock.py day_of_the_week.py
Submitting creature_combat.py...OK
Submitting gandalf.py...OK
Submitting leap_year.py...OK
Submitting tricky_lock.py...OK
Submitting day_of_the_week.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 - Creature Combat

Creatures in our game will have two stats, power and toughness. Power is the amount of damage that they deal and toughness is the amount of damage that they can sustain.

Both creatures will deal damage to each other simultaneously which means that both can survive, one can survive or neither depending on their relative powers and toughnesses.

For instance a P(Power) = 3 T(Toughness) = 2 creature vs a P = 5, T = 2 creature will mutually annihilate.

Creatures, one with P = 5 T = 7 and another with P = 3 T = 6 will both survive the combat.

If one creature has P = 4 T = 2 and a second has P = 3 T = 5 then the second creature will win because it can absorb 4 damage and still have 1 point of toughness left, whereas the other creature cannot absorb three points of damage.

You should write a program which asks for the names, powers, and toughnesses of two creatures, and then fights them. It should then report the result.

Sample Output:

```
linux3[9]% python3 creature_combat.py
What is the name of the first creature? Newton
What is the power of the first creature? 10
What is the toughness of the first creature? 6
What is the name of the second creature? Liebniz
What is the power of the second creature? 8
What is the toughness of the second creature? 15
The first creature now has (10, -2)
The second creature now has (8, 5)
Newton has died, Liebniz wins

linux3[10]% python3 creature_combat.py
What is the name of the first creature? Sulla
What is the power of the first creature? 5
What is the toughness of the first creature? 4
What is the name of the second creature? Caesar
What is the power of the second creature? 10
What is the toughness of the second creature? 1
The first creature now has (5, -6)
The second creature now has (10, -4)
Both creatures die in mutual combat.

linux3[11]% python3 creature_combat.py
What is the name of the first creature? Daybreak
Chaplain
What is the power of the first creature? 1
What is the toughness of the first creature? 3
What is the name of the second creature? Impassioned
Orator
What is the power of the second creature? 2
What is the toughness of the second creature? 2
The first creature now has (1, 1)
The second creature now has (2, 1)
Both creatures live to fight another day.
```

# Problem 2 - Lord of the Rings

For this problem you'll ask the user about their favorite game.

Make sure you name the file gandalf.py (all lower case).

Let's make a guessing game to determine which character 'you are' in Lord of the Rings.

You'll get a lot of practice nesting if/elif/else statements here.

First, you should ask which race the character is. The options are human, dwarf, elf, maiar and hobbit.

1) If human, ask if they are the King of Gondor?
    a) If yes, they are Aragorn son of Arathorn
    b) If no, ask if they tried to take the ring from Frodo?
        i) If yes they are Boromir.
        ii) If no, they are Theoden, probably.
2) If they are an elf, then ask if they were in the matrix.
    a) If yes, they're Elrond.
    b) If no, they're Legolas.
3) If they're a Maiar, then ask if they are good or evil.
    a) If "good", then they are Gandalf
    b) If "evil" then ask if they forged the One Ring.
        i) If yes, they're Sauron.
        ii) If no they're Saruman.
4) If they're a hobbit ask if they carry the One Ring
    a) If yes, they are Frodo Baggins
    b) If no, ask if they are a gardener.
        i) If yes they're Samwise.
        ii) If no, then they're either Merry or Pippin.
5) If they are a dwarf then they are Gimli son of Gloin.
6) If they didn't give one of these answers then they're an Orc.

You may use .lower() on strings to make sure that whatever the user enters, it's always in lowercase. That will simplify our lives quite a bit. Whenever a question asks "yes" or "no" you only have to check against "yes." If the answer is anything else, like "affirmative", "absolutely", "negative" they all count as "no". The only thing that counts as yes is "yes".

Sample Runs:

```
linux4[116]% python3 gandalf.py
Which race are you? (human/dwarf/elf/maiar/hobbit) Blah
You are an Orc, sorry about that.

linux4[117]% python3 gandalf.py
Which race are you? (human/dwarf/elf/maiar/hobbit) human
Are you the true and rightful King of Gondor? no
Did you try to take the ring from Frodo? yes
You are Boromir, poor guy...

linux4[118]% python3 gandalf.py
Which race are you? (human/dwarf/elf/maiar/hobbit) maiar
Do you end up being good or evil? evil
Did you forge the one ring? yes
You are Sauron

linux4[119]% python3 gandalf.py
Which race are you? (human/dwarf/elf/maiar/hobbit) maiar
Do you end up being good or evil? good
You are Gandalf

linux4[120]% python3 gandalf.py
Which race are you? (human/dwarf/elf/maiar/hobbit) elf
Were you in the matrix? yes
You are Agent Smith, I mean... Elrond

linux4[121]% python3 gandalf.py
Which race are you? (human/dwarf/elf/maiar/hobbit) dwarf
You are Gimli son of Gloin
```

# 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.

You should be using modulus division for the problem. Modulus division is denoted as n % d and gives the remainder of the integer division n // d.

This means that for instance 5 % 3 is 2 because 5//3 = 1 with a remainder of 2. 17 % 4 is 1 because 17 = 16 + 1 and 16 is divisible by 4.

14 % 7 = 0 because 14 is divisible by 7.

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 - Tricky Lock

For this problem you'll ask the user about their favorite game.

Make sure you name the file tricky_lock.py (all lower case).

In order to open a lock, you need a two number combination and the correct combination on three switches.

The numbers range from 1 to 25, and the switches can be up or down.

The lock will open if the sum of the two numbers is equal to 36, and exactly two of the three switches are in the up position (meaning that the other one is down).

The rules are here:

1. If both of these conditions are true, then the lock opens.
2. If one of the two conditions are true, then the lock will clank loosely but does not open.
3. If they are both false then the lock is solidly closed.

Feel free to use the .lower() method to ensure that the 'up' and 'down' input is case insensitive. However for testing we'll ensure that all input is in lowercase.

Warning:

Be careful not to do something like

```
if x == 'up' or y or z:
```

Remember that this will only check the first condition against 'up' to see if they're equal. The rest will be converted to booleans and tested.

Sample Runs:

```
linux4[116]% python3 tricky_lock.py
What is the first number in the combination lock? 18
What is the second number in the combination lock? 18
What is the position of the first switch (up/down)? up
What is the position of the second switch (up/down)? down
What is the position of the third switch (up/down)? up
The lock opens, you gain access to the treasure.


linux4[117]% python3 tricky_lock.py
What is the first number in the combination lock? 14
What is the second number in the combination lock? 12
What is the position of the first switch (up/down)? down
What is the position of the second switch (up/down)? up
What is the position of the third switch (up/down)? up
The lock clanks but does not open.

linux4[118]% python3 tricky_lock.py
What is the first number in the combination lock? 5
What is the second number in the combination lock? 5
What is the position of the first switch (up/down)? up
What is the position of the second switch (up/down)? up
What is the position of the third switch (up/down)? up
The lock does not even budge, try again.
```

# Problem 5 - Day of the Week

Name your file day_of_the_week.py

We're going to ask for the numerical date in September 2023, for instance September 16th. We will tell the user that September 16th is a Saturday.

The first day of September is a Friday, and there are 30 days in the month total.

You should check that the user enters a number between 1 and 30 [inclusively] so that they don't go out of bounds of the month.

You should then tell the user what the day of the week is.

As an additional challenge I want you to also get the suffix right, so for instance, if the day is 1, 21 then it should say "1st" and "21st". If it is 2nd or 22nd, you should use the 'nd' suffix, and if the day is 3 or 23 you should add 'rd' as the suffix. Otherwise the suffix should be 'th'.

You should be using modulus division for this problem.

Sample Output:

```
linux3[14]% python3 day_of_the_week.py
What day of September 2023 is it? 13
September 13th is a Wednesday

linux3[15]% python3 day_of_the_week.py
What day of September 2023 is it? 21
September 21st is a Thursday

linux3[16]% python3 day_of_the_week.py
What day of September 2023 is it? 29
September 29th is a Friday

linux3[17]% python3 day_of_the_week.py
What day of September 2023 is it? 2
September 2nd is a Saturday

linux3[18]% python3 day_of_the_week.py
What day of September 2023 is it? 1
September 1st is a Friday

linux3[19]% python3 day_of_the_week.py
What day of September 2023 is it? 77
That day 77 is out of range, you must enter a number
between 1 and 30

linux3[20]% python3 day_of_the_week.py
What day of September 2023 is it? -3
That day -3 is out of range, you must enter a number
between 1 and 30
```

# 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                creature_combat.py
tricky_lock.py              gandalf.py
day_of_the_week.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 creature_combat.py
leap_year.py
Submitting creature_combat.py...OK
Submitting leap_year.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 leap_year.py
It seems you have already submitted a file named
leap_year.py.
Do you wish to overwrite? (y/n):
y

linux1[6]%
```