DATA203 Foundational Python (Prof. Maull) / Spring 2025 / HW2

Points Possible	Due Date	Time Commitment (estimated)
None	Sunday, April 13	*up to* 10 hours

- **GRADING:** Grading will be aligned with the completeness of the objectives.
- **INDEPENDENT WORK:** Copying, cheating, plagiarism and academic dishonesty *are not tolerated* by University or course policy. Please see the syllabus for the full departmental and University statement on the academic code of honor.

OBJECTIVES

- Practice writing functions.
- Finding prime numbers.
- Printing patterns with loops.

WHAT TO TURN IN

You are being encouraged to turn the assignment in using the provided Jupyter Notebook. To do so, make a directory in your Lab environment called `homework/hw1`. Put all of your files in that directory. Then zip or tar that directory, rename it with your name as the first part of the filename (e.g. `maull_hw1_files.zip`, `maull_hw1_files.tar.gz`), then download it to your local machine, then upload the `.zip` to Canvas.

If you do not know how to do this, please ask, or visit one of the many tutorials out there on the basics of using zip in Linux.

If you choose not to use the provided notebook, you will still need to turn in a `.ipynb` Jupyter Notebook and corresponding files according to the instructions in this homework.

ASSIGNMENT TASKS

(20%) Practice writing functions.

We can all use a little more practice writing functions in Python, so here we go.

§ Task: Write a function to the find the largest number from a list of arbitrary size.

This should be very straightforward, but remember:

1. you will need to *keep track* of the largest number you have seen, thus a loop will be your friend and a variable, say `max_n` to store the largest seen so far,
2. don't overthink it, you only need one loop and some conditional logic to check the current number in the loop.

(50%) Finding prime numbers.

Prime numbers are interesting for a wide array of reasons (some practical, some purely mathematical), but as a reminder, a *prime number* is a number which is only divisible but 1 and itself. Common small primes we know, love and interact with everyday are 3, 7, 13, 41, 53. It should be noted *finding* large primes is hard work, and not something we will do with this assignment.

For example, *do you know if the number 1137 is prime*?

§ Task: Write a function that takes two parameters *a* and *b* and returns the list of all primes between *a* and *b*.

You will need to use the `is_prime()` function provided in the starter notebook:

```python
def is_prime(n):
    import math
    if n <= 1:
        return False
    else:
        is_prime = True
        for i in range(2, int(math.sqrt(n)) + 1):
            if n % i == 0:
                is_prime = False
                break
        return is_prime
```

You will write finish the function that takes two numbers (*a*, *b*) and returns all the prime numbers in between them. Assume the first number is smaller than the second, and that they are both non-negative (greater than 0).

Remember:

1. you will need to loop through all the numbers between *a* and *b*,
2. `range` can be used here and might save you time,
3. if a number is even (divisible by 2: hint if you use modulus operator `8 % 2` is 0 meaning that 8 is even or that the remainder when divided by 2 is 0), it will NEVER be prime, so there is no need to run `is_prime()`,
4. don't overthink this, but remember, you will need to *store* the prime numbers you find in a list and *return* that list.

Use the `find_all_primes(a, b)` starter code below to get you started.

```python
def find_all_primes(a, b):
  primes_found = []

  ## write the solution code to add primes to primes_found

  return primes_found
```

(30%) Printing patterns with loops.

This is a fun exercise to close out our work with loops.

Here is what you will need to keep in mind:

1. there are 9 rows, the longest row is 5,
2. a loop that increments by one will get you the first half,
3. decrementing this will give you the second half.

You will need to use at least one loop, but a nested loop might be useful.

Use the cell provided and write the function named `print_text_triangle()`. The function does NOT return anything, it merely prints the triangle.

§ Task: Write a Python function to print the following pattern.

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```