# 15-110 Hw2 - Programming Portion

Each of these problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Hw2 - Programming for autograding.

All programming problems may also be checked by running the starter file, which calls the function `testAll()` to run test cases on all programs.

## #1 - `printTriangle(n)` - 10pts

Write a function `printTriangle(n)` which prints an ascii art triangle out of asterisks based on the integer `n`. For example, `printTriangle(5)` would print the following:

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

Note that the triangle is five lines long, with the top and bottom line each having only one asterisk, the second and second-from bottom lines each having two asterisks, etc. So `printTriangle(9)` would look like:

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

You'll want to create a loop where each iteration prints a single line of the triangle. To draw multiple stars on a single line, consider using a nested loop or the * operator.

**Note:** `n` is guaranteed to be positive and odd.

**Hint:** how can the program switch from increasing numbers of stars to decreasing? Consider using a conditional to check when you hit the midpoint, or two separate loops (one going up, one going down).

## #2 - `printPrimeFactors(x)` - 10pts

Write the function `printPrimeFactors(x)` which takes a positive integer `x` and prints all of its prime factors in a nice format.

A prime factor is a number that is both prime and evenly divides the original number (with no remainder). So the prime factors of 70 are 2, 5, and 7, because 2 * 5 * 7 = 70. Note that 10 is not a prime factor because it is not prime, and 3 is not a prime factor because it is not a factor of 70.

Prime factors can be repeated when the same factor divides the original number multiple times; for example, the prime factors of 12 are 2, 2, and 3, because 2 and 3 are both prime and 2 * 2 * 3 = 12. The prime factors of 16 are 2, 2, 2, and 2, because 2 * 2 * 2 * 2 = 16. We'll display repeated factors on a single line as a power expression; for example, 16 would display 2 ** 4, because 2 is repeated four times.

Here's a high-level algorithm to solve this problem. To find factors manually, iterate through all possible factors. When you find a viable factor, repeatedly **divide the number** by that factor until it no longer evenly divides the number. Our algorithm looks something like this:

```
1. Repeat the following procedure over all possible factors (2 to x)
    a. If x is evenly divisible by the possible factor
        i. Set a number count to be 0
        ii. Repeat the following procedure until x is not divisible by the
            possible factor
            1. Set count to be count plus 1
            2. Set x to x divided by the factor
        iii. If the number count is exactly 1
            1. Print the factor by itself
        iv. If the number count is greater than 1
            1. Print "f ** c", where f is the factor and c is the count
```

As an example, if you call printPrimeFactors(600), it should print

```
2 ** 3
3
5 ** 2
```

## #3 - `getSecretMessage(s, key)` - 10pts

You can hide a secret message in a piece of text by setting a specific character as a key. Place the key before every letter in the message, then fill in extra (non-key) letters between key-letter pairs to hide the message in noise.

For example, to hide the message "computer" with the key "q", you would start with "computer", turn it into "q**c**q**o**q**m**q**p**q**u**q**t**q**e**q**r**", and then add extra letters as noise, perhaps resulting in "orup**qc**rzyp**qo**m**qm**hcy**qp**whh**qu**t**qt**xt**qe**ye**qr**pa". To get the original message back out, copy every letter that occurs directly after the key, ignoring the rest.

Write a function `getSecretMessage(s, key)` that takes a piece of text holding a secret message and the key to that message and returns the secret message itself. For example, if we called the function on the long string above and "q", it would return "computer". You are guaranteed that the key does not occur in the secret message.

**Hint:** loop over every character in the string. If the character you're on is the key, add the **next** character in the string to a result string.

## #4 - `getMiddleSentence(s)` - 10pts

Write the function `getMiddleSentence(s)` that takes a string `s`, checks whether it has exactly three sentences, and if it does, returns the middle sentence. We define a sentence to be a consecutive string of one or more non-whitespace characters that ends in one of the following characters: **`. ! ?`**

For example, given the following string:

```
"You've got to ask yourself a question. Do I feel lucky? Well, do ya, punk!"
```

The function should return `"Do I feel lucky"`. Note that we remove the punctuation at the end of the returned sentence for simplicity.

If the inputted string does not have exactly three sentences, you should instead return `"Improper structure"`. Note that the test cases are guaranteed to not use ., !, or ? inside a sentence and each sentence will only end in one punctuation mark.

To solve this problem, you should use **string operations and methods**. Specifically:

- s.replace() can help turn multiple punctuation types into one
- s.count() can detect if there are exactly three sentences or not
- s.find() and slicing can help find the beginning and end of the middle sentence

## #5 - `sumAnglesAsDegrees` - 10pts

When analyzing data, you need to convert the data from one format to another before processing it. For example, you might have a dataset where angles were measured in radians, yet you want to find the sum of the angles in degrees.

Write the function `sumAnglesAsDegrees(angles)` which takes a list of angles in radians (floats) and returns the sum of those angles **in degrees** (an integer). To do this, you will need to change each angle from radians to degrees before adding it to the sum. You can do this with the library function `math.degrees()`. Make sure to round the final result to get an integer answer.

For example, `sumAnglesAsDegrees([math.pi/6, math.pi/4, math.pi])` should convert the radians to approximately `30.0`, `45.0`, and `180.0`, then return `255`.

## #6 - `onlyPositive(lst)` - 10pts

Write a function `onlyPositive(lst)` that takes as input a **2D list** and returns a new 1D list that contains only the positive elements of the original list, in the order they originally occurred. You may assume the list only has numbers in it.

Example: `onlyPositive([[1, 2, 3], [4, 5, 6]])` returns `[1, 2, 3, 4, 5, 6]`, `onlyPositive([[0, 1, 2], [-2, -1, 0], [10, 9, -9]]` returns `[1, 2, 10, 9]`, and `onlyPositive([[-4, -3], [-2, -1]])` returns `[ ]`.