HW4 CS103 Fall 2025. due: 11.02.2025

Grading & Deliverables

Deliverables: Submit `hw4.py` and `independent_completion_form` for grading.

Submission Note: Ensure the script file is named **`hw4.py`** to avoid penalties.

Rubric: **`secondLongest[15 points], arglongest[15 points], flagInitials[70 points]`**

Correct implementation: All tests pass

Naming conventions: Exact function and file naming required. Ensure to use proper variable names.

** Test your functions for edge cases. Functions must adhere to the exact names and argument structures provided to pass the automated tests.

Mandatory Functions

Mandatory Functions: Implement the following functions with the exact names and parameters specified. Call these functions as demonstrated below.

```
def myName():
    return "James Bond"
def myBlazerID():
    return "jbon12"

# Call these functions
print("My Name is =", myName(), " and my BlazerId is =",myBlazerID())
```

HW4 problems

Please use the same names for the functions and the input parameters.

`secondLongest (L)`

Write the function longest that, given a nonempty list of strings, finds the **second longest even-length** string that ends with *ing*. It takes a list of strings L as input and returns a string. If there are many second longest strings, return the first. If there are no such strings, return the empty string (which clearly indicates that no string was found, since a valid string would need to be at least length 3).

Sample Inputs	Expected Outputs
l = [“aing”, “bb”, “cccing”,”ddding”, “zzzzzing”]	“cccing”
l = [“swimming”, “flying”, “coding”, ”abracadabra UAB CS103”, “yoo”, “python programming”]	“swimming”
l = [“robinhood” , “amazon”, “ding”, “joking”, “booking”]	“joking”
l = [“fishing”, “fly”, “count”, “it”, “is”, “cooking”, “swimming”]	“fishing”

`arglongestEven (L)`

Write the function argLongestEven that, given a nonempty list of strings, finds the index of the **longest even-length** string that ends with *ing*. It takes a list of strings L as input and returns an integer, the index of the longest string in L with a suffix ing of even length. If there are many longest strings, return the index of the first one. If there are no such strings, return the index -1.

Finding the index is more valuable than finding the element: you can quickly retrieve the element using the index, but you cannot quickly retrieve the index using the element. So we want to get in the habit of computing indices of elements, not elements.

Sample Inputs	Expected Outputs
l = [“aing”, “bb”, “ccing”,”ddding”, “zzzzzzzzzzz”]	3
l = [“swimming”, “flying”, “coding”, ”abracadabra UAB CS103”, “yoo”, “python programming”]	5
l = [“robinhood” , “amazon”, “ding”, “bing”,]	2
l = [“fishing”, “doesn’t”, “count”, “it”, “is”, “not even-length”]	-1 **fishing has 7 letters*

`flagInitials ()`

Write a function **`flagInitials()`** that draws your three initials (first name, middle name, last name) using **maritime letter flags** with the **turtle graphics** module.

For example, if your initials are **JRT** (for *John Ronald Reuel Tolkien*, the author of *The Lord of the Rings*), your program should draw the maritime flags for **J**, **R**, and **T**, each as a **square** with **random size** and **random spacing** between them. (If you have multiple middle names, choose your favorite initial. If you don't have a middle name, draw letter **X** for the middle initial)

https://en.wikipedia.org/wiki/International_maritime_signal_flags

The bottom-left corner of the first flag should appear at a random position within the bounded area (for example, between **(-200, -200)** and **(0, 0)**). Each flag must have:

- A **distinct and consistent color pattern** that represents its letter (for instance, "J" could use blue and white stripes, "R" could use red and yellow, and "T" could use black and white).
- A **1-pixel black border** around the edges.
- **Randomized placement and width**, so the drawing appears slightly different each time the program runs.

You do **not** need to reproduce the exact official maritime flag designs, but each letter's design should be **recognizable, unique, and consistent** across runs.

Background:
Maritime letter flags are used on ships to send signals to other ships. Each flag corresponds to a letter of the alphabet. Note that every flag is **square-shaped** and should have a **1-pixel black outline**.

You will use **randomization** to determine the position and size of the flags. The bottom-left corner of the first flag should be placed randomly within the square bounded by **(-200, -200)** and **(0, 0)**.
Each flag has the same width w, which should be chosen randomly between **100 and 200**.
The gap between consecutive flags should be a random number between **1 and 10**.

Since these random values determine the size and position of your flags, you should see a **slightly different drawing each time** you run the program.

The purpose of using random values is to encourage the use of **variables instead of magic numbers**.

For example:

- Use forward(w / 2) instead of forward(40) — this makes your code easier to understand and reuse.

Requirements:

1. Each flag must be a **square** outlined with a **1-pixel black border**.
2. Use randomization to vary:
    - The bottom-left corner of the first flag (within **(-200, -200)** to **(0, 0)**).
    - The flag width w (random between **100 and 200**).
    - The gap between consecutive flags (random between **1 and 10**).
3. All flags should be drawn **side by side, left to right**.
4. Use **variables instead of magic numbers**.
    - Example: use forward(w / 2) instead of forward(40).
5. Each time you run your code, the drawing should appear **slightly different** due to randomization.

Hints:

- Import both the **turtle** and **random** modules.
- Define a **helper function** for each letter flag (or at least for your own initials).
- Use turtle.pendown(), turtle.pensize(1), and turtle.color("black") to draw flag borders.
- Use variables like x and y to keep track of the turtle's position when moving between flags.