Feb 13st: In class exercise
(nothing to hand in)
You are encouraged to work together in groups in class. Part of this exercise involves discussion and exploration with others.

Background:
Automated testing involves more than automating test scripts. In this exercise we look a bit at some tools which help us with test automation.

Exercise Environment:
We will be using the COM S server: pyrite.cs.iastate.edu for the first two exercises where this has been tested. You are welcome to try other environments if you can get it to run there. You will need Java, a text editor.

Download:
In Canvas under modules you can download "afl-417-exercise.s24.tar.gz". You can also download evosuite (evosuite-spring24.tar.gz) which you will find under assignment 2. Using your U drive, or sftp (you can use putty or a native shell if you have a Linux or Mac machine) transfer the file to your pyrite account.

Part I: Evosuite

(a) Use this part of the exercise to ask questions/get help with your assignment

(b) Run the evosuite tool on the stack program.
- javac11 Tutorial_Stack/tutorial/*.java
- java11 -jar evosuite-1.2.0.jar -projectCP Tutorial_Stack -class tutorial.Stack

Run again using random test generation (this will simulate how randoop might work)

- java11 -jar evosuite-1.2.0.jar -generateNumRandom 6 -projectCP Tutorial_Stack -class tutorial.Stack

Run this a couple of times and compare the statistics.csv file with the original run from evosuite (NOTE- you will want to delete the statistics.csv file prior to starting your homework so this data is no longer in the file).

Do you notice any difference? Look at the test cases and see how well Random is performing compared with the standard evosuite.

DISCUSS among your group

Part II: AFL

American Fuzzy Lop is a brute-force fuzzer coupled with an instrumentation-guided genetic algorithm. It uses a modified form of edge coverage to pick up subtle, local-scale changes to program control flow. I have provided a subset of the tool (for space reasons) compiled on pyrite. Feel free to get the full distribution from git: https://github.com/google/AFL

Copy the file to pyrite.
[[If you do this locally (on your own machine) you will need to make each of the individual programs using Make (make afl-fuzz, afl-as, afl-gcc,afl-plot,afl-clang,afl-min)- you do not need to do this on pyrite. If you want to 'make-all' you need the full distribution.]]

Unzip and untar the file on pyrite This will create a directory structure for you with the files

Inside of that directory move to the "afl-demo" directory.
In that directory there is a file called torun.txt. It has the following commands which you will use to run the program. (note – without exporting the environment you will get a security issue).

- export AFL_I_DONT_CARE_ABOUT_MISSING_CRASHES=1
- make
- ../afl-fuzz -i triangletests/ -o findings ./triangle

As a group

Let AFL run for 3 or 4 minutes or so and try to understand the output. Use Ctrl C to stop the program. Now explore a bit. You can find the input file in 'testcases/input.txt'. That is the starting test case. You can find the crash results under 'findings'. Try running the program using the crash input. First create a file called input.txt in the same directory as the triangle program. It should have a single line with 3 inputs: e.g. 3 4 5 (this will be whatever is in your crash file).
To run:

- ./triangle < input.txt

It should crash. Now try changing to a passing triangle in that file (e.g. 4 6 6). It should pass. Try to determine where the fault is that the fuzzer found. Are there more?

Let the program run for a longer time and see if you can find more crashes (note – this may not be possible during the exercise time.). Think about the implications and differences with evosuite.

Part III: Optional

Go to the Selenium site: https://www.selenium.dev/
Look at the documentation and try to learn more about how it works. If you want to try the Web IDE please go ahead. This is not required since it will ask for browser permissions that you

may not be comfortable with. In practice when you use Selenium you will most likely use your own local server and test from there – rather than testing a live website. But this is an easier set up for us to try.
You can try to use the Blockly website https://developers.google.com/blockly as a starting point.

You can find an interesting link there and put that as the base URL and record/play some test cases. First get familiar with the site. You can look for the catalog of parts and try to search for a particular part. Or just use it to find other information about igem. Then record the test and replay the it. Try adding a second test and run the test suite. Save the test **and locate the .side** file on your machine. Take a look at the test file and see if you can figure out what it is doing.