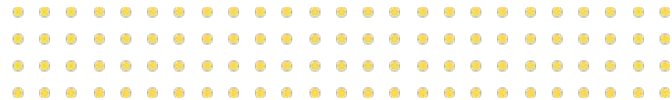




# Discussion 5:

## Unit Tests & Debugging



# Reminders



- Submit your work to **Canvas Assignment / Discussion 5**
  - Commit **at least 4 times** and push to GitHub
  - Submit the **repository link** to Canvas by the end of discussion
- **Homework 4** due this **Friday September 27 @11:59pm**
- **Spaced Practice Tool** (5 questions max/day)

# Please remember:



- Commit **at least 4 times** to get full credit on all HW assignments and projects.
- **Your file has to run for us to grade it** – please double-check that the final version you hand in runs successfully, including all tests and any necessary interactivity in `main()`.
- **Use meaningful commit messages!**
  - i.e. “Completed `__init__` method”

# What are Tests?



- Tests are a checklist of **user inputs** that your programs have to pass.
- We have to make sure the programs succeed and give expected output.

```
def calculate_average(numbers):  
    return sum(numbers) / len(numbers)  
  
#print_first_element(????)
```

Python

# How to test?



- Generate different inputs (common and edge cases)
- Calculate expected output
- Run the program with these inputs and check to see if any errors are thrown
- Compare the program output with the expected output
- Repeat!

# Unit Tests in Python



- A unit test, as the name suggests, tests **individual units of code (like functions)** in isolation from the rest of the application.
- unittest is a library (something somebody once wrote) to write tests easily in Python.

Method	Description
<code>assertEqual(expected_value, actual_value)</code>	Asserts that <code>expected_value == actual_value</code>
<code>assertTrue(result)</code>	Asserts that <code>bool(result)</code> is True
<code>assertFalse(result)</code>	Asserts that <code>bool(result)</code> is False
<code>assertRaises(exception, function, *args, **kwargs)</code>	Asserts that <code>function(*args, **kwargs)</code> raises the exception

Figure 1: **Basic assertions** that unittest offer

# Unit Tests in Python



```
import unittest

def calculate_average(numbers):
    return sum(numbers) / len(numbers)

class TestAll(unittest.TestCase): # Make a subclass of unittest
    def test_calculate_average(self): # Start each method with "test"
        # Test normal cases
        self.assertEqual(calculate_average([1]), 1)
        self.assertEqual(calculate_average([1,2,3]), 2)

        # Test edge cases
        self.assertEqual(calculate_average([]), "invalid input")
        self.assertEqual(calculate_average(["hello, world!"]), "invalid input")

unittest.main() # Run all tests
```

# Testing & Debugging Tips



1. It is often useful to **FIRST write tests**, THEN write the program. Tests will help you think of how your program will behave in edge cases.
2. **Start small**: don't wait for code to get too long to test it.
3. Common out things you don't need
4. Use **print** statements (in for loops, functions, etc.)
5. Break complicated lines into shorter ones.



# Discussion 5 Assignment



- Go to **Canvas Assignments > Discussion 5**
- Accept the GitHub Classroom assignment and clone the repo:  
<https://classroom.github.com/a/zR4-lchR>
- If you are having issues:
  - Canvas Files > Discussions > Discussion 5 > discussion\_5.py
- **Commit at least 4 times and push to GitHub**

# Typical Git Workflow



1. Clone the repository: **git clone <link>** (from GitHub Classroom)
2. Add file to staging area: **git add <file1> (<file2>)**
3. Make snapshot of current change: **git commit -m "<message>"**
4. Upload to cloud server (GitHub): **git push**

Use **git status** to check current changes. **Make sure you are in the same directory/folder of the .py file you are working on.**

# Task 1: count\_a method



- `count_a`: method that **counts the number of a's in a string**. You are going to test and see if it works.
  - Discuss with your classmates. Do the cases mentioned in the given comments make sense? How will you write tests to check for them?
  - **Write tests for these cases.**
  - **There are errors in `count_a` – fix them!** (Hint: first see if you can spot where likely errors are by looking at what test cases fail).

# Task 2: Warehouse & Item Classes



- **Item** class contains information about an item with attributes:
  - name
  - price
  - stock
- **Warehouse** class stores and makes calculations from items.
  - **Now you will implement test methods for the Warehouse class.**
  - We have created some items in setUp for you – feel free to use them if you wish!

# Warehouse Class Functions



- **add\_item:**
  - add\_item is a method for the Warehouse class that **adds an item to the warehouse**
- **get\_max\_stock:**
  - get\_max\_stock is a method for the Warehouse class that **finds and returns the item with the highest stock**
- **get\_max\_price:**
  - get\_max\_price is a method for the Warehouse class that **finds and returns the item with the highest price**
- **These methods are already written out for you. Write tests to determine if they work as expected.**