11 Prove Milestone: Student Chosen Program

Purpose

Prove that you can write a significant Python project that solves a real-world problem and is well organized with functions.

Assignment

During this lesson and the next two lessons, write a Python program that you choose. Your program must include multiple functions that each perform a single task. You must write test functions for at least two of your program functions and run your test functions with `pytest`.

Ideally your program will solve a real-world problem or become a tool in a subject area that interests you, such as chemistry, physics, geography, fashion, family history, linguistics, social work, food science, carpentry, machinery, engineering, mathematics, computing, etc. Before you begin writing your program, you must ***propose your program to your instructor*** to ensure that the program is neither too easy nor too difficult for the time remaining in this course. If you finish your program more quickly than you or your instructor expected, and there is time remaining in this course, then choose and write another program.

Proposal

To write a proposal for your program, download and save this proposal.txt file. Open the downloaded file in VS Code. Answer each of the questions and save your file.

Possible Projects

If you find it difficult to choose a program to write, you may choose to write one of the following programs:

1. A program that performs calculations that are useful in your major or hobby.

2. A program with a complete graphical user interface (GUI), including windows, frames, labels, text fields, and buttons. Use either the `tkinter` or

`kivy` module. The team assignment in the next lesson will show you how to write code for a simple GUI.

3. A program that reads a text file and counts how often each word occurs in the file.

4. A program that reads English text from a file and converts strangely spelled words to a simpler spelling that more closely matches their pronunciation. Consider just a few of the strangely spelled English words: comb, lamb, have, done, two, Wednesday, should, night ("ight" is a blight on the English language). For more information, see the English-language spelling reform article at Wikipedia.

5. A program to reorganize or rename all the files in a collection of files, such as collection of Microsoft Word documents, Excel spreadsheets, music (mp3) files, pictures (jpg) files. If you choose to write this program, we ***strongly*** suggest that you make a copy of the files that your program will reorganize or rename and run your program on the copy. This is because it is highly likely that you will make mistakes when writing this program, and such mistakes may cause the files to be deleted. Really.

6. A program that graphically simulates the spread of a disease like the simulations in this Washington Post article. Your program should include variables that can be changed, such as movement rate, probability of transmission, length of contagiousness, length of illness, and probability of death.

7. A program that retrieves data from a a server on the internet. The Python `requests` module contains functions that retrieve data from a server.

8. A program that uses the `pandas` module to process a data set and produces charts that help people visualize information about the data set.

9. A program that retrieves data from a relational database. The MySQL Python connector is a Python module that enables Python programs to read from and write to a MySQL relational database. It was written by the same developers who wrote the MySQL server. You can download it from the MySQL downloads page and install it on your Windows, Mac OS, or Linux computer.

10. A program that retrieves data from a Firebase database.

These three websites list other ideas for Python projects:

- 42 Exciting Python Projects for Beginners
- 45 Fun Python Project Ideas for Easy Learning
- Python Projects for Beginners

If you don't understand the description of one of these possible projects, ask your CSE 111 team members or your instructor to explain the project to you.

Organization

As you develop your program, be certain to divide the program into functions. Each function should perform a single task and be appropriately named. Remember that the most reusable and easily testable functions don't get user input and don't print results but instead have parameters and return a result. Functions that get user input and print results are important and do useful work but are not easily reusable and testable. It's fine if some of the functions in your program have no parameters, return nothing, get user input, and print results, but it's bad if all of your program functions are like that.

Testing Procedure

You should follow good coding practices by writing test functions for many of your program functions. Run your test functions with `pytest` to verify that your program functions work correctly.

Some students believe that the program functions they write cannot be tested with test functions and `pytest`. Such beliefs are simply false. Below are just some of the reasons that students have thought make their program functions untestable and example code that shows how to write those test functions. Notice that you can expand and collapse the example code by clicking the triangle icons (▶ or ▼).

My function can't be tested because it:

▼ Calculates numbers

```
def payment(amt, ar, y, ppy):
    """Computes and returns the payment amount
    for a loan with a fixed annual interest rate.
    """
    r = ar / ppy
    n = y * ppy
    p = amt * r / (1 - (1 + r) ** -n)
    return round(p, 2)
```

```
def test_payment():
    """Verify that the payment function works correctly."""
    assert payment(10_000, 0.07, 4, 12) == 239.46
    assert payment(80_000, 0.04, 15, 4) == 1779.56
```

▼ Processes text instead of numbers

```
def prefix(s1, s2):
    """Return the prefix, if any, that appears in both
    s1 and s2. In other words, return a string of the
    characters that appear at the beginning of both s1
    and s2. For example, if s1 is "inconceivable" and s2
    is "inconvenient", this function will return "incon".
    """
    s1 = s1.lower()
    s2 = s2.lower()
    i = 0
    limit = min(len(s1), len(s2))
    while i < limit:
        if s1[i] != s2[i]:
            break
        i += 1
    return s1[0:i]
```

```
def test_prefix():
    """Verify that the prefix function works correctly."""
    assert prefix("", "") == ""
    assert prefix("", "correct") == ""
    assert prefix("clear", "") == ""
    assert prefix("happy", "funny") == ""
    assert prefix("cat", "catalog") == "cat"
    assert prefix("dogmatic", "dog") == "dog"
    assert prefix("jump", "joyous") == "j"
    assert prefix("unwise", "ungrateful") == "un"
    assert prefix("Disable", "dIstasteful") == "dis"
```

▼ Gets user input from a terminal

```
def get_int(prompt, min, max):
    """Prompt the user for an integer, verify that it is
    between min and max inclusive, and return the integer.
    """
    done = False
    while not done:
        try:
            text = input(prompt)
            number = int(text)
            if number < min:
                print("Invalid input: "
                    f"number must be {min} or greater.")
            elif number > max:
                print("Invalid input: "
                    f"number must be {max} or less.")
            else:
                done = True
        except ValueError as val_err:
            print(f"Invalid integer: {text}. Please try again.")
    return number
```

```
from io import StringIO
import re
import sys


def test_get_int(monkeypatch):
    """Verify that the get_int function works correctly."""
```

```
    stdin = StringIO("1r\n-3\n12\n8")
    stdout = StringIO()
    monkeypatch.setattr(sys, "stdin", stdin)
    monkeypatch.setattr(sys, "stdout", stdout)

    min = 1
    max = 10
    prompt = f"Enter an integer between {min} and {max}: "
    num = get_int(prompt, min, max)

    monkeypatch.undo()
    pattern = prompt + "\\s*" \
        + "Invalid integer: 1r\\. Please try again\\.\\s*" \
        + prompt + "\\s*" \
        + f"Invalid input: number must be {min} or greater\\.\\s*" \
        + prompt + "\\s*" \
        + f"Invalid input: number must be {max} or less\\.\\s*" \
        + prompt + "\\s*"
    output = stdout.getvalue()
    assert re.compile(pattern).match(output) != None
    assert num == 8
```

▼ Gets user input from a GUI

```
"""
In order to test a program that gets user input from a
graphical user interface, separate the event functions (the
ones that are executed when a user types a value in a text
field) into two functions. Move the calculations out of the
event function and into a calculation function that takes
parameters, performs a calculation, and returns a result.
Then write a test function for the new calculation function.
"""
    # This function is called each
    # time the user releases a key.
    def calc(event):
        try:
            # Get the user input.
            w = txtWidth.get()
            a = txtRatio.get()
            d = txtDiam.get()

            # Compute the tire volume.
            v = tire_volume(w, a, d)

            # Display the volume for the user to see.
            lblResult.config(text=f"{v:.1}")

        except ValueError:
            # When the user deletes all the digits in one
            # of the text fields, clear the result labels.
                lblResult.config(text="")


def tire_volume(width, ratio, diam):
    """Compute and return the approximate volume of a tire."""
    vol = (math.pi * width * width * ratio *
           (width * ratio + 2540 * diam)) / 10_000_000
    return vol
```

```
def test_tire_volume():
    """Verify that the tire_volume function works correctly."""
    assert tire_volume(185, 60, 14) == approx(30101.58, 0.01)
    assert tire_volume(205, 60, 15) == approx(39924.49, 0.01)
```

▼ Writes to a file

```
def write_file(filename, lines):
    """Create a file and write lines of text to it.
    Parameter
        lines: a list of strings.
    Return: nothing
    """
    with open(filename, "wt") as outfile:
        for text in lines:
            print(text, file=outfile)
```

```
import os

def test_write_file():
    """Verify that the write_file function works correctly."""
    lines = [
    "Beware of false prophets, who come to you in sheep's",
    "clothing, but inwardly they are ravening wolves. Ye",
    "shall know them by their fruits. Do men gather grapes",
    "of thorns, or figs of thistles? Even so every good tree",
    "bringeth forth good fruit; but a corrupt tree bringeth",
    "forth evil fruit. A good tree cannot bring forth evil",
    "fruit, neither a corrupt tree bring forth good fruit.",
    "Every tree that bringeth not forth good fruit is hewn",
    "down, and cast into the fire. Wherefore, by their fruits",
    "ye shall know them."
    ]
    filename = "fruits.txt"

    # Call the write_file function to
    # write a file named fruits.txt.
    write_file(filename, lines)

    # Read the contents of the fruits.txt file.
    with open(filename, "rt") as infile:

        # Read all the characters in the file into a string.
        string = infile.read()

    # Split the string into a list of strings named
    # written. Each line of text from the text file
    # will be stored in its own element in the list.
    written = string.splitlines()

    # Delete the fruits.txt file.
    os.remove(filename)

    # Verify that write_file correctly
    # wrote the fruits.txt file.
    assert lines == written
```

▼ Reads from a file

```
def read_file(filename):
    """Reads and returns the contents
    of a text file as a list of strings.
    """
    lines = None
    with open(filename, "rt") as infile:

        # Read all the characters in the file into a string.
        string = infile.read()

    # Split the string into a list of strings named
    # lines. Each line of text from the text file
    # will be stored in its own element in the list.
    lines = string.splitlines()

    return lines
```

```
import os

def test_read_file():
    """Verify that the read_file function works correctly."""
    # Write a sample file with three lines
    filename = "lines.txt"
    lines = ["first", "second", "third"]
    with open(filename, "wt") as outfile:
        print(*lines, sep="\n", file=outfile)

    # Call read_file to read the sample file.
    read = read_file(filename)

    # Delete the lines.txt file.
    os.remove(filename)

    # Verify that read_file read the file correctly.
    assert read == lines
```

▼ Uses the pandas module

```
def filter_for_meter(df, meter_number):
    """Return a new DataFrame that contains only the rows
    where the meterNumber column equals the parameter
    meter_number.

    Parameters
        df: The DataFrame that this function will filter.
        meter_number: df will be filtered so that all the rows
            in the new DataFrame will have this meter number.
    Return: A new DataFrame.
    """
    filter = (df["meterNumber"] == meter_number)
    filtered_df = df[filter]
    return filtered_df
```

```
import pandas as pd

def test_filter_for_meter():
    """Verify that the filter_for_meter
    function works correctly.
    """
```

```
    # Read the water.csv file and convert the
    # readDate column from a string to a datetime64.
    df = pd.read_csv(FILENAME, parse_dates=["readDate"])

    # Call the filter_for_meter function.
    meter_num = "M4103"
    filtered_df = filter_for_meter(df, meter_num)

    # Verify that the filtered data
    # frame contains at least one row.
    assert len(filtered_df.index) > 0

    # Verify that all the meter numbers in the
    # filtered data frame are the correct number.
    for value in filtered_df["meterNumber"]:
        assert value == meter_num
```

▼ My code is so good that it doesn't need to be tested.

Good luck with that attitude, hot shot. In all seriousness, how do you know your code is good? Besides, no matter how good of a programmer you are, you still make mistakes. It is better to find those mistakes yourself by writing and running test functions than it is to make users of your software find your mistakes.

Submission

At the end of this lesson and lessons 12 and 13, you must submit a description of your work, and your teacher or teaching assistant will grade your work according to the following rubrics.

Lesson 11 Rubric

1. Proposal—40%: Does your proposal adequately answer the questions in the template?

2. Time—50%: Did you spend at least three hours on your proposal and program during the current lesson?

3. Description—10%: Is the description of your work for this lesson complete and easily understandable? At the end of each lesson, you will submit a description of your work. This description should follow normal English rules of spelling and grammar. It must include the following:
    a. The amount of time that you spent working on your program during the current lesson.
    b. A list of the documentation that you read, the videos that you watched, and the coding experiments that you tried.
    c. A description or list of the work that you finished on your program.

Lesson 12 Rubric

1. Time—50%: Did you spend at least three hours on your Python program or test functions during the current lesson?

2. Organization—20%:
    a. Is your program organized into multiple functions?
    b. Does each function in your program perform just one task?

3. Progress—20%: Did you complete some significant part of your program during the current week?

4. Description—10%: Is the description of your work for this lesson complete and easily understandable? Your description should include the following:
    a. A list of the function names in your program.
    b. A list of the test function names in your test code.
    c. A list of the documentation that you read, the videos that you watched, and the coding experiments that you tried.
    d. A description or list of the work that you finished on your program.

Lesson 13 Rubric

1. Time—50%: Did you spend at least six hours on your Python program or test functions during the current lesson?

2. Description—10%: Is the description of your work for this lesson complete and easily understandable? Your description should include the following:
    a. A list of the function names in your program.
    b. A list of the test function names in your test code.
    c. A list of the documentation that you read, the videos that you watched, and the coding experiments that you tried.
    d. A description or list of the work that you finished on your program.

3. Python program file—25%: Upload your Python program. Your teacher will evaluate it according to these criteria:
    a. Your program is divided into functions and each function performs one task only.
    b. Your program effectively uses existing Python modules such as math, random, requests, pandas, and tkinter.
    c. Your program performs a significant real world task.

4. Python test file—15%: Upload your Python test file. Your teacher will evaluate it according to these criteria:
    a. Each testable program function is covered (tested) by one test function. (Some functions, especially main and those that create GUIs are difficult to test and don’t need to have a corresponding test function.)

b. Each test function completely exercises (tests) its corresponding program function. In other words, the test function calls the program function multiple times with different arguments, including unusual or unexpected values.