CS2023: Module 1

Introduction to Software Development with Python

Dr. Fred Annexstein

CS2023 Course Organization

- There is a learning Module each week with a lab assignment due at the end of each week.
- There will be 3-4 homework programming challenge projects assigned during the term.
- There will be 2 multiple-answer exams.
- Grades will be determined by a weighted average of grades, distributed as follows:
  - Labs 40%, Homeworks 25%, Exams 35%.
- With the exception of exams, group efforts will be allowed, but must be properly documented.
- Labs and Homework assignments are auto graded using Python’s doctests.
- Labs and Homeworks are to be submitted by each student individually, and any help should be clearly stated in the submission files.
- Late Submission Policy: Late work accepted with a penalty of 10 points, and a penalty of 10 points for every week past the deadline.

Python Platform

- Recommend using the Anaconda Python distribution because it is easy to install on the most platforms such as Windows, macOS and Linux.
- Anaconda supports the latest versions of Python, the IPython interpreter, and Jupyter Notebooks. Anaconda also includes other software packages, libraries, and tools commonly used in Python programming and data science, allowing students to focus on learning the Python language and computing skills.
- Recommend using the Spyder development environment application, which is packaged with Anaconda. This environment will allow you to easily experiment and test for your Python programming exercises.
- Download the Anaconda3 installer **`https://www.anaconda.com/download/`** When the download completes, run the installer and follow the on-screen instructions. To ensure that Anaconda runs correctly, do not move any of the files after you install it.

Using Python Doctests

- Doctests are a way to embed code examples and expected outcomes within your docstrings and then automatically test them to ensure that your code is functioning as expected. This makes your documentation more interactive and serves as a form of documentation and testing at the same time.
- Writing Docstrings: Use the >>> prompt to indicate input and the expected output. For multi-line inputs or outputs, use ... after the first line. Here is a small factorial function with several doctests. Each test checks whether the code for the factorial function produces expected results.

```
def factorial(n):
    """Return the factorial of n >= 0.
    Here are three doctests:
    >>> factorial(3)
    6
    >>> factorial(30)
    265252859812191058636308480000000
    """

    f = 1
    for i in range(2,n+1):
        f *= i
    return f
```

Running Doctests

1. To run a program doctests we add the following “boilerplate” code to the end of the program.

```
if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

2.Now when we run the code all the doctests will run and the interpreter will show the results. Your output should be as follows:

```
Trying:
    factorial(3)
Expecting:
    6
ok
Trying:
    factorial(30)
Expecting:
    265252859812191058636308480000000
ok
1 items had no tests:
    __main__
1 items passed all tests:
   2 tests in __main__.factorial
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```

Assignment: Lab #1

- From Canvas download the the folder for the Lab 1 on Expressions and Control. Extract the files in the folder if needed and open the index.html file in you browser.
- You should see the Lab worksheet and it should be formatted with a Red Banner on top. Complete the worksheet.
- Copy the code for Lab 1 Required Questions and paste into your Spyder workspace.
- Your submission for Lab 1 will be a modification of the file. You should write an implementation that passes all of the doctests. Once your code passes all the doctests, you should upload this and be confident you will receive full marks.