EN.540.635 "Software Carpentry"

Syllabus

Spring 2026 (3 credits)

Class Meeting Time and Location:
Tuesdays and Thursdays, 3:00 - 4:15 pm in Shaffer 001

Office Hours:
In-person: Valentina Matos (Time **Tuesdays, 12 pm**, Loc **Croft 130**)
In-person: Luke Oluoch (Time **Fridays, 3 pm**, Loc **Maryland 322**)

Grading: Grading will be based on the following breakdown.

- Weekly Programming Challenges and Labs (30%)
- Lazor Group Project (35%)
- Final Project (25%)
- Participation (10%)

Course Instructors: Valentina Matos (vmatosr1@jh.edu), Luke Oluoch (loluoch1@jhu.edu)
External Course Administration: Prof. Paulette Clancy (pclancy3@jhu.edu)
Website: Click here

Course Description: "Software Carpentry" is a term used to describe a course that teaches basic or foundational skills needed for research computing. This course will embrace that concept by building a foundation for the internal workings of a computer, starting from the hardware level and moving to the software level. From there, students will learn how to manage file systems with confidence and begin a "crash course" or a refresher course in basic programming. The course will teach the following skills: proficiency in Linux and the command line, introductory Python skills (variables, conditionals, loops, functions, classes, plotting), data structures (lists, linked lists, stacks, queues, trees), version control, and algorithms (efficiency, big-O notation, searching/sorting methods), along with some research applications like machine learning, image processing, using LaTeX etc.

Assignments:

1. **Weekly Programming Challenges (HW) and Labs** - Required challenges sent out weekly to test your programming skills. **Solutions must be written in Python 3 and uploaded to Canvas before class on Tuesdays.** Late assignments will only be accepted through Thursdays at 11:59pm and for a max grade of 75%. These assignments will progress in difficulty as the semester progresses. This course is aimed towards programming beginners, certain assignments may be weighted more than others (this is subject to change). Generally, labs will be completed during the class period and will not be handed in for a grade (with a couple of exceptions) or they may be considered for extra credit.

2. **Lazor Project** - This will be a group project, done on Github. We will judge your individual contribution to the assignment by your own push/pull commits and peer review. Further, if you have an issue with teammates, let us know early on! If you do not, then you are stuck with whatever grade you will be given in the end, no exceptions.

3. **Final Project** - Apply what you've learned to anything you want! The only requirement is that it must be more involved than anything done during the class so far. The final project must be either independently done, or as a pair (if working as a pair, a peer-review will have to also be turned in). You must gain approval of your final project topic from the course instructors by November 17th, and there will be an opportunity for you to present your project near the end of the semester.

Week 1 - Get to know more about the machine you use everyday

Weekly Challenge 1 - Design a Computer and Setup your Programming Environment. **Due Feb. 3 before class**

Jan. 20: Lecture 1 - Introduction to the Class and What is a Computer?

Jan. 22: Lecture 2 - Linux Crash Course, SSH and SFTP, and Introduction to Command Line and File Management.

Week 2 - Intro to Python !

Jan. 27: Lab 1 - Defeat the Virus.

Jan. 29: Lecture 3 - Introduction to Python.

Week 3 - More Python !

Weekly Challenge 2 - Quadratic Equation Solver. **Due Feb. 10 before class**

Feb. 3: Lab 2 - Boot Camp

Feb. 5: Lecture 4 - More Python Basics and the Python Imaging Library.

Week 4 - Learning how data is stored in Python

Weekly Challenge 3 - Finish Lab 3. **Due Feb. 17 before class**

Feb. 10: Lab 3 - PIL Image Blurring and Luminance.

Feb. 12: Lecture 5 - Dictionaries, Modular Arithmetic, and Exceptions.

Week 5 - OOPs ! What I need to know in Python

Weekly Challenge 4 - RSA Encryption. **Due Feb. 24 before class**

Feb. 17: Lab 4 - Prime Numbers and Factorization

Feb. 19: Lecture 6 - Functions, Classes, and Graphing with Matplotlib.

Week 6 - Plotting

Weekly Challenge 5 - Object-Oriented Plotting. **Due Mar. 3 before class**

Feb. 24: Lab 5 - Python as a Graphing Calculator.

Feb. 26: Lecture 7 - Variable Scope, Recursion, and Data Structures.

Week 7 - Import this ..

Weekly Challenge 6 - Passwords and Encrypted Messaging. **Due Mar. 10 before class**

Mar. 3: Lecture 8 - Git, Version Control, and Python Modules.

Mar. 5: Lab 6 - Maze Generation.

Week 8 - Maze Generation

Weekly Challenge 7 - Maze Generation and Solving. **Due Mar. 24 before class**

Mar. 10: Continue Maze Generation and Introduce Lazor Project.

Mar. 12: Lecture 9 - Building User Interfaces with Tkinter.

Week 9 - Spring Break

Mar. 17: Spring Break - No Class

Mar. 19: Spring Break - No Class

Week 10 - Lazor Project

Start Lazor Project.

Mar. 24: Lab 7 - Unit testing, Data Parsing and Organization.

Mar. 26: Lecture 10 - Efficiency, Big-O Notation, and Compiled Languages.

Week 11 - Python Application Week

Mar. 31: Lab 8 - Pokemon Index.

Apr. 2: Lecture 11 - Searching Algorithms and Gradient-Based Minimization in Optimization Problems.

Week 12 - More Applications for Python

Complete Lazor Project, due Apr. 7 before class

Apr. 7: Lab 9 - Conjugate Gradient

Apr. 9: Lecture 12 - Data Mining, APIs, and an Introduction to Machine Learning.

Week 13 - Crawl, Walk, Code: Baby Steps into Machine Learning

Start Final Project, Submit Proposal by Apr. 21.

Apr. 14: Lab 10 - Neural Networks for Digit Classification.

Apr. 16: Lecture 13 - A Primer in Latex

Week 14 - Learning LaTex

Apr. 21: Lab 11 - Update your Resume/CV

Apr. 23: Final Project Work Session

Week 15 - The end is a new beginning?

Complete Final Project.

Apr. 28: Final Project Work Session

Apr. 30: Final Project Work Session

Resources:

The following digital resources may be of use (links provided):

1. Install Anaconda (Python 3.12). Instructions for installation can be found for Windows, macOS, and Linux. NOTE! You should use Anaconda (not miniconda), and make sure to install the proper version with the latest version of Python.
2. Overview of Secure Shell. How to SSH and SFTP in Windows and Unix/Linux.
3. The Windows Command Prompt, the macOS Terminal, and the Linux Terminal. Here is also a cheat sheet for common commands in Unix and Linux.
4. Github (website, tutorial 1, tutorial 2).
5. The Coding Train - A YouTube series by an entertaining man on programming in general (different languages than Python normally).
6. Practice coding with Project Euler and CheckIO.
7. The holy grails of finding programming help on the Internet: stack overflow and stack exchange.
8. Learn vim using a game and a cheat sheet of common commands.
9. How to Think Like a Computer Scientist: Learning with Python 3 (online Python 3 Textbook).

Common Point Deductions:

Over the years that this class has been taught, the following have been reasons that students have lost points on assignments. Please keep these in mind as you proceed in this class:

- Turning in a Python code that does not run. Please do not do this! Also make sure that if you start making edits, you can go back to a version of your code that still works if you can't solve the bugs in a later version and need to turn in the assignment.
- Failure to use PEP8 styling when preferred. PEP8 is a standard for constructing your scripts. It allows another person to easily read your code and understand the intention behind each component.
- Failure to have docstrings in the assignments (both at the start of a Python file, and for each function). The docstring at the start of the script file provides a short summary of what the script does. The subsequent docstrings for functions and classes elaborate on what each function/class's role is. It also provides supporting information on any parameters, attributes, or outputs it may have. This helps add to the readability of your code.
- Failure to put all functions prior to any code that runs.
- Failure to include dependencies outside of publicly available python libraries (maybe that you wrote) which you did not include in your submission.
- Failure to put all running code in an *if __name__ == "__main__"* block.
- Having inappropriate or confusing variable names. This includes variable names that are far too short or truncated to grasp their meaning.
- Hard-coding variables (see Addressing a Problem Statement for an example)
- Missing aspects of the assignment (Ex. if the assignment says to make a function but you instead write the code in the name block).
- Failure to incorporate appropriate exception handling.

Note, as you will learn the above in class, we will warn you when we notice them at first; however, repeated incidences will result in points taken off. Further, when it comes to the Lazor project and final project, you will need to take them into account no matter what! Our goal with this approach is for you to write code that not only runs, but is readable and easy to understand from the perspective of any user (or grader, in this case). Please ask if you have any questions!

Addressing a Problem Statement:

Coding is a creative process! When programming, you will quickly find that there are many ways of solving the same problem. In this class, we like to acknowledge well-thought code, and encourage students to pursue a general solution that addresses the entire parameter space of a problem. While writing code that doesn't address the entire scope of the parameters space of a problem won't ruin your chances for a good score on an assignment, by default, 10% of the grade will be reserved for simply addressing the entire parameter space. For example, imagine an assignment that requires a student to reverse a string. One may do the following:

```
x = "Some sentence"

y = ""
for index in [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]:
    y += x[index]
print(y)
```

This code works, but would likely have a point taken off for an easy source of error with the explicitly stated list (what if my message changes?). Furthermore, points would be removed as there are no docstrings, no name block, and poorly named variables. The ideal assignment would look something like the following:

```
"""
Assignment x — Reverse a string.
This assignment involves printing out the reverse of a string, which
has been stored as a variable.
"""

if __name__ == "__main__":
    message = "Some sentence"

    # Here we reverse the string using slicing
    inverted_message = message[::-1]

    print(inverted_message)
```

Note the use of docstrings, comments, and the name block. Finally, note that the solution is concise, and how the inverted_message variable is assigned in such a way that if the message changes to be longer or shorter than 13 characters, it will still work! This would merit full points.

General Rubric:

All code that runs will be given a 50%. From that, points are generically awarded as per the below rubric. Note that on specific assignments, these points may deviate; we will let you know how so.

Points	Category
Code Functionality (35 points total)	
15	Correct Input and Output (5 for correct variable input/output, 10 for correctly spanning parameter space)
10	Code Conciseness (be succinct but readable!) and Flexibility (no hardcoding!)
10	Code Organization (e.g. appropriately broken into classes / functions)
Code Styling (15 points total)	
5	PEP8 Styling
5	Readability (e.g. appropriate variable names)
5	Comments (docstrings for files & function; appropriate commenting)

ACADEMIC INTEGRITY: Practical problem solving is a key skill you will develop in this class. You are encouraged to discuss challenging lab work with your fellow students; however, every student is responsible for their own write-up. This means that you must personally type every keystroke in your submitted work. The only exception to this rule is if you are asked to work as a team or if you incorporate code or text from material distributed in class or on the class website. You may not copy or edit another student's work and submit it as your own, either in full or in part. You may not jointly edit or compose solutions with another student. The exchange of solutions in electronic, printed, or written form is prohibited. Both sender(s) and receiver(s) will be prosecuted according to the Graduate Academic Misconduct Policy. You may read more information on University Policies here.

PLAGIARISM STATEMENT: In this course you are not allowed to use any GPTs or ChatBots, paraphrasing tools (like QuillBots) or other source of plagiarized information to help write your code. We will check your homework and project for plagiarism using TurnItIn and other tools. If your work is assessed by these tools to contain more than 20% of plagiarized material, you will be called in for an interview to review the situation. Egregious plagiarism and any other form of cheating will be reported to the Office of Student Conduct for review. This may result in a zero grade for the assignment. Students will have access to the tools we use to check for plagiarism, so they can check their own work.

INCLUSIVITY STATEMENT: Johns Hopkins is a community committed to sharing values of diversity and inclusion in order to achieve and sustain excellence. We firmly believe that we can best promote excellence by recruiting and retaining a diverse group of students, faculty and staff and by creating a climate of respect that is supportive of their success. This climate for diversity, inclusion and excellence is critical to attaining the best research, scholarship, teaching, health care and other strategic goals of the Health System and the University. Taken together these values are recognized and supported fully by the Johns Hopkins Institutions leadership at all levels. Further, we recognize that the responsibility for excellence, diversity and inclusion lies with all of us at the Institutions: leadership, administration, faculty, staff and students. Software Carpentry (EN.540.635), in particular, is an inclusive classroom that is open to the views of every student in the classroom. Please respect your fellow classmates and appreciate the opportunity to be in an environment where we can all learn from each other.

ACCOMMODATION STATEMENT: All students with disabilities who require accommodations for this course should contact the instructor(s) at their earliest convenience to discuss their specific needs. If you have a disability and are requesting accommodations, you must be registered with Homewood office of Student Disability Services (101 Shaffer Hall; 410-516-4720) to receive accommodations.

THIS DOCUMENT IS SUBJECT TO CHANGE AT ANYTIME.