INST326: Object-Oriented Programming
Course Syllabus

Spring 2020 – Section 101 – Hornbake Library (HBK) 0302H – MWF 1-1:50

---

1 Instructor

Name:	Joshua A. Westgard, PhD
Email:	westgard@umd.edu
Phone:	301-405-9136 (office)
Office:	B0225 McKeldin Library
Office Hours:	M 10-11, W 3-4, and by appointment

2 Catalog Description

This course is an introduction to programming, emphasizing understanding and implementation of applications using object-oriented techniques. Topics to be covered include program design and testing as well as implementation of programs. Prerequisite: (must have completed or be concurrently enrolled in INST201; or INST301); and (INST126; or CMSC106; or CMSC122). Or permission of instructor. Credit only granted for: INST326 or CMSC131.

3 Extended Course Description

This course covers (1) the core features of the Python programming language, (2) using programs to collect, process, and analyze data, and (3) object-oriented programming. Object-oriented programs are built as collections of "objects", which are software representations of real-world entities and concepts. Objects combine data (attributes) with functionality (methods), and work through communicating with each other as the code is executed. By encapsulating code complexity within objects, OOP allows use and reuse of existing code in a relatively simple and easy manner. Advanced OOP concepts such as inheritance facilitate development of complex code without sacrificing robustness and possibility of code reuse. We apply computational thinking approaches such as abstraction, decomposition, algorithmic design, generalization, evaluation, and debugging.

This course also provides opportunities to develop an understanding of how programming is situated in and reflects broader social structures, constructs and issues, e.g. race, class or gender. Programming is often viewed as a value-neutral technical skill. However, the social and cultural impacts of information and technology are central concepts in our field, and the growing awareness of issues like algorithmic bias, ethical/unethical uses of algorithms and disparities in opportunities in tech jobs require that any informed professional needs to understand the larger context of programming. This is important to be ethical professionals and to be successful in the workplace. Through readings, discussion and writing, we will critically examine issues of racism, sexism and other forms of power and oppression that are pervasive

in programming and related technical activities, and discuss what companies and individuals are doing to improve programming practices and professional work environments.

4 Student Learning Outcomes

After finishing this course, students will be able to:

1. Design, program, and debug Python applications to solve non-trivial problems;
2. Write scripts to collect, process, and/or analyze data;
3. Explain OOP concepts, principles, design patterns and methods;
4. Test and assess code quality;
5. Write clear and effective documentation;
6. Explain how programming is situated in and reflects social issues (e.g. racism, classism, ableism, or sexism) and describe actions that individuals or organizations are taking to counteract disparities and inequities.

5 Teaching Notes

This course assumes a basic understanding of procedural programming, and begins with a comprehensive review of Python fundamentals—including data types, variables, loops, and conditionals—that is designed to deepen your mastery of these concepts. The first part of the course will thus be an opportunity to consolidate and extend what was covered in INST126. Alternatively, if you have worked with a language such as JavaScript, Java, C#, or Visual Basic, you should be able to apply that knowledge to learning Python. The later parts of the course will cover certain topics in program design and programming best practices (documentation, testing, etc.) that are a necessary part of producing complex, reliable, and maintainable applications.

If you have a strong foundation in Python programming already, and are interested in being challenged, I invite you to talk to me about leading a session (it's really true that you learn more by teaching), identifying more challenging exercises, or developing a more ambitious project. I want you to learn as much as you can from this course.

The course is divided into weekly modules, and will typically follow this pattern, with some exceptions:

- Before class (preparation):
  - Do assigned readings and/or watch assigned videos;
  - Complete any online worksheets, exercises, or quizzes that are due.
- In class:
  - On the first day of each module (usually Monday), we will have a lecture to introduce the topic;
  - On the second day (usually Wednesday), we will have a mix of Q&A, whiteboarding, and other hands-on activities, generally working together;
  - On the third day (usually Friday), we will have a lab activity designed to help you apply what you have learned independently;
  - Quizzes may be administered in class;

  - Each midterm exam will include both an in-class and take-home section.

- After class (programming homework):
  - There will be five homework assignments to help you apply, reflect and extend your understanding by working on a practical task;
  - All homework assignments are to be completed on your own unless otherwise stated on the assignment hand-out.

Over the course of the semester, we will also examine selected broader issues of programming and coding—-the social and organizational context, issues related to gender, race, disability, etc. This will help you prepare for situations that you are likely to encounter in your professional work. These are noted in the schedule as "Critical Reflections."

Here is my suggested general strategy for working on assignments:

1. Start early—don't wait. That will give you time to work through the problems and get help as needed.
2. Plan out your solution, but follow the incremental coding procedure as you implement it. Incremental coding means getting a version that does some small part of the whole task, and then adding to it step by step, testing each time you add something. This approach makes it much easier to locate problems, and you'll be able to see the data changing as you go.
3. Read error messages carefully and try to understand what they are telling you. Often they will point you directly to the cause of the error.
4. If the solution is not immediately obvious, spend an additional 5-10 minutes trying to solve it on your own, but then take a break. Sometimes this will allow you to come back and see something you missed.
5. If you've spent 20-30 minutes and still are stuck—ask for help! You can contact me via email or discussion board. Please provide as much information as you can. Often it helps to include a screenshot with the problem. I will respond as soon as I am able, usually within a day.
6. If you see a question on the discussion board that you can answer, or if you have an idea, please respond. Don't wait for me. You will be helping your colleagues.

6 Textbooks & Readings

There is no book required for this course, but we will make use of the following open-source texts:

- Charles R. Severance, *Python for Everybody: Exploring Data Using Python 3* ISBN-13: 978-1530051120 `http://py4e.com`
- *The Python Tutorial*, v3.7.2, Python Software Foundation `https://docs.python.org/3/tutorial/index.html`
- *Object-Oriented Programming in Python*, University of Cape Town `https://www.cs.uct.ac.za/mit_notes/python/`

Other readings (generally available online, or through Library subscriptions) may be assigned as needed.

7 Required Technology

- Laptop: We will do programming in class, so bring your laptop and be prepared to write code. Any current OS can be used. If you do not have access to a laptop, contact me immediately.
- Python: The Python interpreter (version 3), freely available from https://www.python.org/downloads.*
- Code Editor: An advanced text editor (such as Sublime Text, Notepad++, or BBEdit) and/or an integrated development environment (such as VSCode, Eclipse, or PyCharm).*

  *Please note that we will install all necessary environments together in class during the first week.

8 Grading

Your final grade for the course is computed as the sum of your scores on the individual elements below (100 possible points total), converted to a letter grade:

A+	97-100	B+	87-89.99	C+	77-79.99	D+	67-69.99		
A	93-96.99	B	83-86.99	C	73-76.99	D	63-66.99	F	0-59.99
A-	90-92.99	B-	80-82.99	C-	70-72.99	D-	60-62.99		

Final grades will be calculated based on the following components:

Module Exercises/Quizzes (22, drop two lowest)	20 points
Homework (5)	25 points
Midterms (2)	20 points
Final Project	15 points
Reflections (3)	10 points
Participation	10 points
TOTAL	100 points

9 University Course Policies

The essential purpose of the university's undergraduate course policies is to enable all of us to fully participate in an equitable, accessible and safe academic environment so that we each can be challenged to learn and contribute most effectively. They address issues such as academic integrity, codes of conduct, discrimination, accessibility, learning accommodations, etc. We are all responsible for following the policies at http://www.ugst.umd.edu/courserelatedpolicies.html. You must read them and send me any questions by the first week of classes.

10 Late Work

Submission instructions are provided with each assignment, but as a general rule the on-time submission window will close in ELMS at midnight on the date due. If you have to miss a deadline, you should inform me as soon as possible, indicating the reason and when you propose to submit your work. If you have a legitimate reason, such as a major medical or family emergency, I may agree to an extension or makeup work. Documentation of the emergency (e.g. a doctor's letter) may be required.

11 Syllabus Revision Policy

This syllabus is a guide for the course and is subject to change with advance notice. Changes will be posted in ELMS. The ELMS course site together with the course OER website, are the definitive locations for all course materials, communication, assignments, and deadlines.

12 Course Schedule

The following table shows the current projected schedule. The course content can be roughly divided into three interrelated units:

- Unit 1: Procedural Programming Review Using Python (∼weeks 1-6)
- Unit 2: Object-Oriented Programming Using Python (∼weeks 7-11)
- Unit 3: Data Analysis Using Python (∼weeks 12-16)

Week	Monday	Wednesday	Friday
1	01/27 Introduction: Syllabus & Overview	01/29 Module 1: Fundamentals	01/31 Module 1: Fundamentals
2	02/03 Module 2: Functions & Iteration	02/05 Module 2: Functions & Iteration	02/07 Module 2: Functions & Iteration
3	02/10 Module 3: Data Types	02/12 Module 3: Data Types	02/14 Module 3: Data Types
4	02/17 Module 4: Serialization & File I/O	02/19 Module 4: Serialization & File I/O	02/21 Module 4: Serialization & File I/O **HW1**
5	02/24 Module 5: Regular Expressions	02/26 Module 5: Regular Expressions	02/28 Module 5: Regular Expressions
6	03/02 **Critical Reflection #1**	03/04 Catch up & Review	03/06 **Midterm #1**
7	03/09 Module 6: OOP Fundamentals	03/11 Module 6: OOP Fundamentals	03/13 Module 6: OOP Fundamentals **HW2**
8	03/16 SPRING BREAK	03/18 SPRING BREAK	03/18 SPRING BREAK
9	03/09 Module 7: Inheritance & Composition	03/11 Module 7: Inheritance & Composition	03/13 Module 7: Inheritance & Composition
10	03/23 Module 8: Packaging & Distributing Code **HW3**	03/25 Module 8: Packaging & Distributing Code	03/27 Module 8: Packaging & Distributing Code
11	04/06 **Critical Reflection #2**	04/08 Catch up & Review	04/10 **Midterm #2**
12	04/13 Module 9: Databases and SQL	04/15 Module 9: Databases and SQL	04/17 Module 9: Databases and SQL **HW4**
13	04/20 Module 10: Data on the Web	04/22 Module 10: Data on the Web	04/24 Module 10: Data on the Web
14	04/27 Module 11: Data Analysis	04/29 Module 11: Data Analysis	05/01 Module 11: Data Analysis **HW5**
15	05/04 **Critical Reflection #3**	05/06 Project Work Day	05/08 Final Presentations
16	05/11 Final Presentations		05/15 No Class **Projects Due 11:59pm**