

***February 9, 2017***

**INST 326 - Object Oriented Programming**

MWF 1:00pm – 1:50pm

Classroom: Tydings 1118

ELMS site: <https://myelms.umd.edu/courses/1220547>

|  |  |
| --- | --- |
| **Instructor**  Bill Kules, Ph.D.  Visiting Associate Professor  [wmk@umd.edu](mailto:wmk@umd.edu) / @billkules  (301) 755-7982 (mobile)  <http://questionablepedagogy.com>  Office: HBK 4111G  Office hours: Wednesdays 2:00-3:50pm | **Teaching Assistant**  Dimitri Wolford  [dimitriwolford94@gmail.com](mailto:dimitriwolford94@gmail.com)  Office: HBK 2108  Office hours: Tuesdays, 6:00-8:00pm |

**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.*

**Extended Course Description**

This course introduces object-oriented design and programming concepts and methods using the Python programming language. 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 any informed professional needs to understand how these issues manifest in a variety of circumstances. Through readings, discussion and writing, we will critically examine issues of racism, sexism and other forms of oppression that are pervasive in programming and related technical activities.

**Student Learning Outcomes**

After finishing this course, students will be able to:

1. Explain OOP concepts, principles, design patterns and methods;
2. Design, program and debug Python applications to solve non-trivial problems;
3. Test and assess the quality of object-oriented code;
4. Write clear and effective documentation
5. Explain how programming is situated in and reflects broader social structures, constructs and issues, e.g. race, class or gender.

**Teaching Notes**

This course builds on a basic understanding of procedural programming, so you have to you have to understand data types, variables, loops, conditionals, etc. and how to use them to write and debug a program. It doesn’t matter what language you know – JavaScript, Java, C#, Visual Basic, etc. – as long as you know one. If you know a bit of Python already, you might find the first part of the course a bit of review. If you 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.

Each week will typically follow this pattern, with some exceptions:

Before class (preparation):

* Do assigned readings; watch assigned videos;
* Do pre-class activities – these help you confirm that you understand the basic material or help you identify specific aspects that you have questions about;
* Complete a low-stakes assignment or quiz the night before class. I review these before class and use them to prepare for class.

In class:

* We will use a mix of lecture, discussion and lots of hands-on activities to help you apply the materials;
* We will make extensive use of paired and group work in class;
* Class is not a time for solo learning. As members of a learning community, we are mutually responsible to each other as learners. Each of us has to be fully engaged with each other in the activities. We have to be supportive of each other as we try to explain or demonstrate something new, as we inevitably make mistakes. We aren’t successful unless everyone is learning.

After class (homework):

* There will be weekly follow-up activities as homework to help you practice, reflect and extend your understandings. You usually have the option of either working together or solo on these, as long as the final work product is your own.

During the second half of the semester, after you have had time to focus on the essential elements of Python, we will 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 perspectives.”

Our time together in class is precious. To use it effectively, you must come to class on time and prepared. Being prepared for class means that you have:

1. Completed all the readings/videos;
2. Attempted all the pre-class activities;
3. Either successfully completed them or submitted your questions the night before class, so I have time to prepare and answer them in class.
4. Arrived 5 minutes before class starts; are in your seat, with PyCharm running. You have downloaded any notes or materials for the day from ELMS. Any paper assignments are ready to hand in. You are ready to take notes.

We will always have at least one online discussion forum active, that you can use to ask/answer questions, get clarifications, point out my mistakes ☺, etc. We may use additional forums for special topics.

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. When you run into a problem, spend 5-10 minutes trying to solve it on your own.
3. Then take a break. Sometimes this will allow you to come back and see something you missed. Letting your sub-conscious work on it for a while (unsupervised, so to speak) will often lead to useful ideas.
4. If you’ve spent 20-30 minutes and still are stuck, post your question online. We are here to help each other, so don’t beat your head against a brick wall - ask for help! When you post, provide as much information as you can. Often it helps to post a screenshot with the problem.
5. I will be monitoring and will respond as soon as I am able, usually within a day (longer during weekends, travel, etc.).
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.

I encourage you to read my teaching philosophy and approach at <https://questionablepedagogy.com/teaching/>. It provides more detail on why I design and teach our course this way.

**Textbooks and readings**

Our primary text, which is also available in print, is:

*Think Python: How to Think Like a Computer Scientist, 2nd Ed.* by Allen Downey

<http://greenteapress.com/thinkpython2/>

We will also use other readings and videos, which I will make available in ELMS.

**Required technology**

* Laptop - We will do live programming exercises during most classes, so bring your laptop and be prepared to write code. Any reasonably current operating system can be used. If you don’t have access to a laptop, contact me before the first class.
* Python - Python programming language (version 3) platform that supports object-oriented programming. The programming platform is freely available from <https://www.python.org/downloads/>.
* Editor - An advanced text editor (such as Sublime Text, Notepad++, Crimson Editor) and/or an integrated development environment (such as NetBeans, Eclipse, PyCharm, Geany).

**Preliminary Schedule (will be updated in ELMS as needed)**

|  |  |  |
| --- | --- | --- |
| **Week** | **Topics** | **Assessment** |
| Pre-semester | Introductory questionnaire |  |
| 1. Jan 25 | Introductions, overview of course, structure, expectations; install Python & PyCharm; read Think Python chapter 1 (TPy 1); review programming basics | Quiz 1 |
| 2. Jan 30 | Python basics  (TPy 2); functions (TPy 3); computational thinking | HW 1 |
| 3. Feb 6 | Graphics & loops (TPy 4); conditionals (TPy 5); code tracing & reference diagrams | HW 2  Quiz 2 |
| 4. Feb 13 | More functions (TPy 6); iteration (TPy 7) | HW 3 |
| 5. Feb 20 | Strings (TPy 8, 9) | Mid-term 1 |
| 6. Feb 27 | Lists (TPy 10); dictionaries (TPy 11) | HW 4 |
| 7. Mar 6 | Tuples (TPy 12); testing & debugging | HW 5  Quiz 3 |
| 8. Mar 13 | Data structures (TPy 13)  Critical perspectives: Limitations of computational thinking | HW 6 |
| 9. Mar 20 | **SPRING BREAK** |  |
| 10. Mar 27 | Files & modules (TPy 14)  Critical perspectives: Socio-technical systems | HW 7  Quiz 4 |
| 11. Apr 3 | Classes and objects (TPy 15)  Critical perspectives: Algorithmic bias | Mid-term 2 |
| 12. Apr 10 | Classes & functions (TPy 16); methods (TPy 17)  Critical perspectives: Coding and gender | HW 8 |
| 13. Apr 17 | OO analysis and design; UML  Encapsulation & inheritance (TPy 18)  Critical perspective: TBD | HW 9  Quiz 5 |
| 14. Apr 24 | TBD  Critical perspective: TBD | HW 10 |
| 15. May 1 | TBD  Critical perspective: TBD |  |
| 16. May 8 | **Last class May 10**  **Bill on travel May 8, 10 - CHI Conference**  TBD  Course evaluations |  |
|  | **No final exam** - The final project report and critical reflection are used instead of a final exam. All materials are due by the end of the normal final exam time slot per Testudo. | Final project  Critical reflection |

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

\* Note: To receive an A+ you must have demonstrated significant contributions to the class in addition to achieving this numeric grade.

|  |  |
| --- | --- |
| **Graded item** | **Percent of**  **final grade** |
| Preparation – Low stakes assessment or quiz before class. | 10 |
| In class participation – Reflects your active and visible engagement with the in-class activities. | 10 |
| Homework – Will be assigned weekly, with some exceptions. Includes coding problems (of course), but will also include analysis questions, brief reflective writing, and other activities. | 30 |
| Quizes (5) - In class (10-15 min) paper-based. | 10 |
| Midterms – There will be two in-class, paper-based mid-term exams. These are diagnostics for you to assess your understanding of the programming basics that are necessary as you proceed through the course and prepare for term project. You will want to address any weaknesses these diagnostics identify to ensure you are well prepared for the project. | 20 |
| Project – The term project will give you an opportunity to apply and extend what you learn in class. | 15 |
| Critical reflection - At the end of the semester you will submit a critical reflection that builds on our critical readings. It will also address the course, your learning, and your plans for professional growth in this area. | 5 |

**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.

**Late Work**

I do not accept late work unless I have approved it by prior arrangement. This is because we usually review solutions in class the day after the assignment is 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. The general policy is that late work will be deducted 20% of its total grade per calendar day, starting on the same day it is due. If you have a legitimate reason, such as a major medical or family emergency, I may agree to an extension or makeup work, which I will grade at the end of the semester. Documentation of the emergency (e.g. a doctor's letter) may be required.

**Syllabus Change 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 is the definitive location for all course work, and communication, including class schedules, assignments and deadlines.