Course Information

Instructor: Chanathip Namprempre. Email: `chanathipn at reed edu`.

Lecture periods: MW 13:10 - 14:30 ETC 208

Lab periods: F 13:10 - 14:30 ETC 208

Access: The course website is at `https://nchanath.github.io/121-S23/` Please check the syllabus before every class period.

Course description from the course catalog: One-unit semester course. An introduction to computer science, covering topics including elementary algorithms and data structures, functional and procedural abstraction, data abstraction, object orientation, logic, and the digital representations of numbers. Emphasis is on mathematical problems and calculations and on recursive algorithms and data structures. The course includes a significant programming laboratory component where students will solve computational problems using a high-level language. The mechanisms for processing and executing programs will be surveyed. Prerequisite: three years of high school mathematics. Lecture-laboratory.

Course overview: Our goal is to introduce you to the principles of computation. That includes teaching you how to write a program abd how a computer interprets and runs that program. Writing a program well can be a difficult skill to master. You must understand the tools available to you, and you must understand the principles of good design that allow those tools to be used in manageable, understandable ways.

Our vehicle for building these skills will be programming in Python. Python is a good introductory language for a variety of reasons. First, it uses a reasonably simple syntax, reducing the need to get bogged down in messy details. Second, it is very flexible, allowing us to show you several different ways of writing programs. Finally, it's a widely used language, so knowing it could be useful to you in projects outside this course and your computer science training.

Given how much time we will spend programming in Python, you could be excused for thinking this was simply a course in how to write Python programs. It is not. There are some areas of programming (like graphical interfaces) that are very useful for practical applications, but which do not require any new conceptual understanding. We will largely be ignoring those areas, though the course should make it easy for you to learn those things on your own with some quick internet searches. We will also be covering topics which do little to aid in the writing of programs, but are important to understanding computation as a whole.

This course is also meant to be an introduction to computer science as a field. To that end, we will be surveying some advanced computer science topics in lecture. These topics won't be discussed in full detail as many of them are covered in semester-long courses offered by our department, but we think it's important to show you how deep and interesting a field computer science can be.

Resources: The following textbooks are required for this course:

- "Think Python: How to Think Like a Computer Scientist" by Allen Downey. ISBN-13: 978-1491939369. Second Edition. Available at `https://www.greenteapress.com/thinkpython/thinkpython.pdf`
- "Composing Programs" by John DeNero. Available at `http://composingprograms.com/`

Coursework and grading:

- Reading: There will frequently be assigned reading, especially in the first half of the course. The material from readings is usually covered in class as well, but the presentation in class and in the book can be quite different. Neither presentation is necessarily better. We deliberately give you multiple presentations because what works best will be different for each student. Sometimes we assign reading from “Think Python”, sometimes instead from “Composing Programs”, and sometimes both. You can choose whichever you find better. They each cover other topics, and your curiosity might lead you to read those extra chapters too.
- Lab homework: Lab programming assignments are your weekly work in this class, used to practice the new material you learn. The first several problems assigned each week will be completed in lab, either as a class or with a partner. The rest must be done outside of class. Don’t be afraid to come to office and tutoring hours if you’re having trouble (but make sure you at least try things for yourself some first). Homework will account for 30% of your grade. This portion will be penalized if you fail to attend labs. They will be due on Fridays at 1pm except for Homework 1, which is due on Sunday Jan 29th at 11pm.
- Projects: There will be several larger programming projects. These involve bigger blocks of code than the homework and give you some experience with programming tasks that are richer, and more involved. They’ll often give you a little room to be creative and to decide for yourself exactly how you want your program to behave. These projects can take considerable time, and you will be working on them at the same time you are completing regular homework. It is important that you manage your time and not put them off too long. Projects contribute to another 30% of your grade.
- Quizzes and mid-term exams: Every so often at the beginning of Wednesday lectures there will be a 10-20 minute quiz. We need to work hard to switch from quiz to lecture, so you need to make sure you are in class on time. Quizzes can be on any material that has been covered in class. They will often be on the most recent material, but they will sometimes cover older material from earlier in the course. I normally announce a Wednesday quiz on the Monday before I give it. There will also be one or two in-class exams, 80 minutes in length, that will cover several weeks of material. I will usually announce the exam the week before it is given. Quizzes and exams will be worth a total of 30% of your grade.
- Class participation: There will be practice sessions during lecture periods to make concepts more concrete. You are required to submit code or handwritten work during some of the periods. This portion will be worth a total of 10% of your grade.

Other policies:

- Submitting work: Lab homework and projects will be submitted online using Gradescope. For nearly all the lab homework we “autograde” your code, using test procedures we’ve configured on Gradescope for each assignment, and this will give you feedback immediately as to whether your code appears to be working and will immediately give you a score. There can sometimes be technical hiccups with this system. If you see any problems with the system, do not hesitate to contact us or one of the course TAs. In addition to our automated testing, for some homework and especially for the projects, we will give you feedback on the quality of your coding, not just on whether or not it worked and worked well.
- Clarity in programming: When you write programs for homework or projects, you are writing code that needs to be understood by others. That means writing code in understandable ways and documenting that code well. We will talk in class about how to do this, and your grade will depend on how well it has been done. It is not enough for the program to do the correct thing.
- Relatedly, its ofen possible to write code using Python tricks and features that we do not teach in class. While learning Python beyond what’s taught in the course is useful and fun, be careful when you do so. Most of the time our exercises are asking you to solve puzzles using only a few tools, and we will purposely restrict what you can use to solve a problem. This kind of puzzling and problem solving strengthens your skills as a programmer and makes a lot of the programming a fun challenge.

  - Attendance: You need to attend class regularly. Missing quizzes and in-class activities will greatly impact your performance in the course. Attendance will also be taken in lab and absences will be penalized. We assume everyone is in class, and make announcements about class under that assumption. If you need to be excused from class or lab, please talk to us as soon as possible. Some excuses (such as illness) may require documentation (such as a doctor's note). If you will be missing class for an excusable but predictable reason (say, a religious holiday) you should inform us before the absence. We will not excuse absences after the fact for reasons that were known ahead of time.
  - Academic integrity: Unless otherwise stated, all work you turn in for this class should be yours and yours alone. We take this very seriously and will not hesitate to report violations of this principle. Working on homework together can be great, and we encourage you to do it. You are allowed to discuss homework with other students and even help a classmate find a bug in their code. You are not allowed to give code to each other. You are allowed to work together to write code collaboratively, especially during in-class exercises, but if so you must list your collaborators in the submitted file. If you have any question about whether some level of cooperation is acceptable, ask us.
  - You are also allowed to look up general background on the internet. This can include looking up rules of Python syntax, for example. You cannot use the internet to find code that is intended to solve the problem you are trying to solve, or to copy such code.

Advice: Don't procrastinate! It is very hard to predict how much time a given program will take to write, even a very simple one. Sometimes what seems easy will turn out to be hard, and what seems hard will turn out to be easy. Often a program feels 99% done, with only a couple little things remaining, and that final 1% ends up taking as long as the first 99% took. You might find that you don't understand something you thought you did and need to ask questions in lab or office hours. Leave extra time. Most of the time you won't need it, but sometimes you will.

Don't get frustrated! Programming and computer science in general require thinking in ways that will be new to many of you. That's part of why this is such a valuable class to take and why it can be so much fun, but it can also make it very difficult, especially at first. Sometimes things will click easily and sometimes they won't. Don't expect everything to work out perfectly the first time. It is entirely normal to run into obstacles along the way.

You should make it a goal to learn to write and assess code without having the computer sitting in front of you. This is a critical skill for communicating your work and ideas to other people. It also gets you into the habit of anticipating what will happen when your code runs, especially any bugs that might be lurking in that code. The quizzes and exams are designed to strengthen this practice, but you might consider, at times, sketching out your homework and programs in this way—on paper, say, away from the computer—as you go through the course.

Have fun! Learning to work with computers can be a fantastic experience. It changes mysterious objects you just take for granted into something you can understand. Whatever you want to do outside computer science, whether it's biology or art or economics, programming can help make it happen. And most importantly, it's just extremely interesting and intellectually rewarding. Don't get so lost in the details of the work that you don't enjoy the experience.

Outcomes: Having completed this course, students will be able to:

- Write moderately complex programs using a high-level programming language. This programming practice includes:
  - file and console input and output
  - conditional execution, iteration, and exception handling
  - program decomposition into procedures, functions, methods, and classes
  - recursive functions
  - higher-order functions
  - object-oriented programming

  - standard data structures and link-based data structures
  - use of existing libraries

- Write common procedures for sorting and searching.
- Analyze the running time of simple programs.
- Write code quickly without the help of computer tools or reference materials.
- Use a computer to test, debug and improve larger, more complex code.
- Reason about how code is executed in the program interpreter.
- Organize and document their code.
- Explain, for several sub-fields of computer science, some main goals and important problems in that field.
- Think creatively and logically about challenging or puzzling problems and apply appropriate strategies to that effort, including persisting through difficulty, working with others, and asking for help when appropriate.