Computing is the art and science of solving problems. If mathematics provides a language to declarative knowledge then computing provides a language to imperative knowledge.

It is about the *how* as opposed to the *what*.

This course will focus on solving problems using paper, pencils, and the programming language Python. Our playground will be applications in scientific computing, text and image processing, social network data, databases, and the World Wide Web. We will begin by exploring basic **control structures** like conditional statements, iteration and recursion, and methods of **abstraction** like functions and types—the building blocks of procedural language. We then shift our focus to **data structures**—lists, arrays, dictionaries, streams—and **algorithms**—searching, sorting—to efficiently represent and manipulate data (or, if you will, information).

This course has several core tenets:

- Computing and computers are not magical.
- Computing, its tools, and its ideas are accessible to everyone provided they work hard and have an open mind.
- Programming requires practice; programming well requires significant practice.
- Computer science provides a language in which to think about and describe problems and their solutions; that language has practical import almost everywhere.

Our goal in this course, above all, is to learn to think like a computer scientist. We *will not be* learning the interfaces of the newest tools; instead, we will learn the principles these tools use to perform their tasks. This distinction is important. Our focus on concepts rather than implementations will future-proof our skill sets.

There are exceptions to this rule, and sometimes we will choose and use a specific tool when doing so helps us to explore important ideas in computer science. We will do our best to point out these exceptions.

Course Structure

Lectures: a mixture of cool topics in computing and specific programming constructs in Python, delivered with interaction in mind, combined with a healthy dose of small group problem-solving in the classroom.

Labs: labs will focus on data-driven applications with a goal of reinforcing the core concepts in computing discussed in class; techniques and material that extend the lecture topics in (usually) novel ways.

Tools and Workflow: one goal of the course is to introduce and enforce modern development workflows—best practices that are followed outside of college and mimic those performed by seasoned programmers, data analysts, and researchers. To this end, we will use `git`, a **version control system**, for lab distribution and lab collection; we will also use `virtualenv` and `pip` for **isolation** and **package management**. Our default editor will be Atom, but this is not a requirement. The most important thing to is learn about your editor—its power and limitations—and to experiment with it actively!

Practice: Thinking like a Computer Scientist

This activity is about writing and following directions. In one role, the students are programmers; in another, they are the computer. This exercise is meant to make clear the necessary preciseness with which computing operates, both in description and execution.

1. Break up into groups of 3–4 people.
2. Choose some spot in the science building as a secret destination. Choose a place that is not universally known—some place for which the directions "Go to X" don't necessarily work.
3. Write directions from our classroom to this place such that someone can find it reasonably efficiently.

4. Trade directions with another team; follow their directions; on a piece of paper, write down where you ended up or where you felt sufficiently ambiguous about where to go that you were stuck; fold your paper in half and don't show it to anyone.

5. Repeat this a few times.