SPACIOUS SYLLABUS*

Why would you want to program in Python?

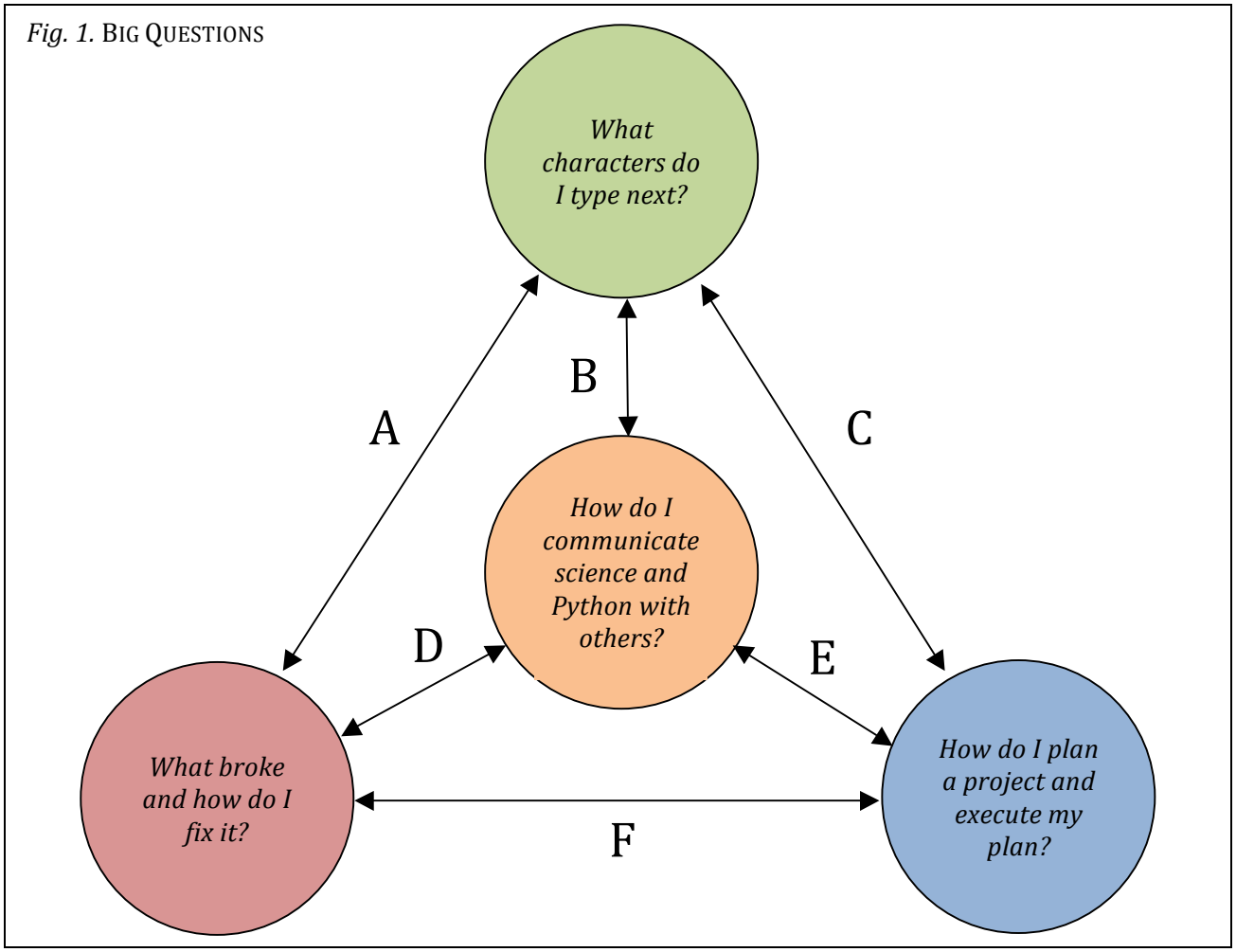
If you're considering going into a scientific field in the modern era, writing code will likely be an essential job skill. For these applications, Python is the most legible and most versatile language around. It's easy to start learning because it's transparent, and it's easy to keep learning because bountiful libraries and documentation are available online. This means that in Python you can get your code working very quickly and get to the science right away. Furthermore, you can use available libraries to optimize your code so that it can efficiently process large simulations and data sets. If this is what you're looking for, then Python is the language for you.

What will you get out of this course?

You will use Python to complete a project that is interesting and relevant to you.

Everything we do is to prepare you to succeed on that project, and to help you apply the lessons from that project to future work. Towards that goal, we will focus on four questions (see Fig. 1): We came up with these questions by examining our own process for approaching Python projects.

Fig. 1. BIG QUESTIONS



**Syllabus originally prepared by Gabriel Ehrlich (Physics 91SI Teacher '13, Advisor '14 – '16).*

These are questions that we ask and answer constantly while working on a project, to the extent working on the project *constitutes* answering these questions. Consequently, our approach to helping you develop proficiency with Python is to help you answer these questions for yourself.

What are the big ideas of this course?
The questions above have no simple answers. The answers are large and complicated, which is why we'll spend 10 weeks helping you start to unpack them. So that we can constantly let you know where we are in the process of answering each question, we've associated each question with a concept that scientific programmers use to think about the process of programming.

These are *our* big ideas:

What characters do I type next?

I. Syntax & Functionality

Syntax is the set of rules that the computer uses to read your code. Functionality is what the computer can do with it. You'll learn all of the syntax and functionality Python has to offer. You'll also learn how to use a few common libraries used for scientific computing, which provide their own functionality. This big idea is about the easiest and fastest ways to get Python to do what you want it to. This is the big idea we'll talk about most.

How do I plan a project and execute my plan?

II. Data Flow & Workflow

A program takes input data and produces output. That's data flow. A programmer takes a desired task and produces a program that accomplishes it. That's workflow. This big idea is about building the perspective and infrastructure to support a programming project.

What broke and how do I fix it?

III. Debugging

Python is pretty good at telling you what went wrong. But you need to know how to interpret what it's telling you, and figuring out how to fix it can be a lot more complicated. This big idea is about how to know when your code doesn't work, and what to do about it.

How do I communicate science and Python with others?

IV. Plotting & Documentation

For scientists, the ultimate goal in programming is to learn and communicate what they've learned. That's what plotting is for. For software engineers, the final step in writing a program is often to make it accessible to others who wish to use it. That's what documentation is for. We'll learn how to make plots and how to read and write documentation. This big idea is about profiting from and giving back to the existing scientific computing community.

What enduring understandings will I gain from this course?

Some of the ideas we'll discuss provide powerful ways of combining skills from different big ideas. They may seem obvious at first, but they have counterintuitive repercussions that deserve special emphasis. While Big Ideas are about ideas and declarative knowledge, these enduring understandings are about habits and procedural knowledge. We've separated them because you'll need a different kind of practice to make them effective and automatic. For ease of recall, we've integrated them into Fig. 1.

SYNTAX & FUNCTIONALITY

A: PYTHON IS A PHYSICAL SYSTEM. EXPERIMENT!

DEBUGGING

Scientific programmers can theorize and experiment about their code. When your code misbehaves, change something and see if it works! If not, figure out why. You'll learn to investigate your code by experimenting with it.

SYNTAX & FUNCTIONALITY

B: LET ME GOOGLE THAT FOR YOU.

PLOTTING & DOCUMENTATION

Google is renowned for its ability to produce results relevant to programmers. When you need to learn or re-learn how to use a piece of functionality, Google it! You'll learn when to turn to Google to find the answer to your question.

SYNTAX & FUNCTIONALITY

C: COMPUTING TIME IS CHEAP—USE IT.

DATA FLOW & WORKFLOW

For everyday tasks, the slowest step is almost never the computer—it's the programmer. Python gives you the resources to write things quickly and get them done, so use them! You'll learn how to prioritize coding time over runtime.

PLOTTING & DOCUMENTATION

D: READ THE ERROR OUTPUT. READ IT.

DEBUGGING

When a remote-control car stops working, it doesn't tell you why. Did the remote batteries die, or the car batteries? Python, on the other hand, is downright loquacious about errors. You'll learn to read and interpret the error output to fix your code.

PLOTTING & DOCUMENTATION

E: DON'T REINVENT THE WHEEL.

DATA FLOW & WORKFLOW

In many languages, you have to write everything from scratch. In Python, that's almost never true: if it's general enough, someone else has probably done it. You'll learn to recognize the point when you can use the wheels others have already built.

DEBUGGING

F: WRITE AND TEST, WRITE AND TEST,...

DATA FLOW & WORKFLOW

You know that coding project at the end of the quarter? You're not going to write it correctly all at once, and if you try to it's going to be a pain to debug. Split it up! You'll learn to write code in chunks and test them along the way.

What are the learning outcomes of this course?
We've picked one learning outcome for each big idea, and one for each enduring understanding. By the end of the course, you should show evidence that you can do these things:

Table 1. LEARNING OUTCOMES

Label	Learning Outcome
I.	Apply **syntax and functionality** of Python and common Python scientific computing modules, including control statements, loops, functions, data structures, programming methodologies, and regular expressions, towards solving real-world programming tasks.
II.	**Organize project files** and decompose functionality in a manner that transparently shows **workflow** and **data flow**. Use **Git** for version control. Navigate **Unix** and a text editor.
III.	Apply the Python **debugging** toolkit, including print debugging and UnitTest, to identify, explain, and solve problems with self-written code.
IV.	Learn/relearn features of Python and its modules by **reading documentation**. Produce transparent documentation for code and plots for data.
A.	Use **experimentation** to explain bugs in code and learn/relearn functionality and syntax.
B.	Find answers to syntax and functionality questions using **online search engines.**
C.	Identify when processing power is not an issue. In those situations, **write code quickly** instead of writing fast code.
D.	Respond to Python-raised exceptions by **reading the error report,** including the description and the traceback. Interpret it to inform debugging.
E.	Predict when desired functionality is likely already implemented and available online.
F.	Write code modularly and test functionality incrementally.

How will you be assessed?

The summative assessment of this course will be the final project you will complete throughout the quarter on a topic of interest and relevance to you. It will assess your progress towards all 10 learning objectives of the course. We will track your progress towards the learning objectives by evaluating your in-class your coding assignments (see below).

This class is graded on a Satisfactory/No Pass basis. In order to guarantee a passing grade, you need to complete the final project satisfactorily (we'll provide a rubric), on time, in its entirety; be present for every lesson; and submit substantial working code at the end of every lab. If you do not complete one or more of these requirements, let's talk, but we can't guarantee you'll pass.

The Final Project

Your task will be to plan, implement, debug, and document a piece of software of a topic of interest and relevance to you. This project will be the exclusive homework for the class, and will require a time commitment of between 10 and 19 hours over the course of the quarter. (Along with the 38 hours of class time, this completes the 57-hour commitment expected of a 2-unit class over 9.5 weeks.)

Most of the time you'll spend on the project will be spent coding. But since this is your first experience using Python for a project, we'll ask you to spend some portion of your time *thinking about coding* using the tools from this class. This will help you apply those tools to the project and to future projects, and it also provides an avenue through which we can assess your learning. Here's a preview of all the parts of the project:

I. SYNTAX AND FUNCTIONALITY: We'll be looking through the code you submit for evidence that you can select elements from the categories of syntax and functionality we'll discuss and apply them appropriately towards your programming task.

II. DATA FLOW & WORKFLOW and F. WRITE AND TEST: We'll inspect the way you have decomposed your code to look for transparency of data flow. We'll also ask for the repository in which you store your final project so that we can inspect the way you have stored your data, tracked your progress on the project, and tested functionality incrementally. There will be a short presentation on the last day of class during which we will assess your ability to navigate Unix and operate a text editor.

III. DEBUGGING: We can guarantee that you'll encounter bugs during the project. We'll ask you to approach them conscientiously, using the tools we'll discuss in class, and then to write briefly about how you used those tools to identify and solve one bug.

IV. PLOTTING & DOCUMENTATION: We will ask you to research and implement functionality from one module we do not discuss in class that helps you with your project, and then briefly explain how that piece of functionality works. This will require reading and interpreting other programmers' documentation. We will further ask you to document sparsely your own code, as well as one portion in depth. Finally, we will ask you to produce a plot showing the results that your project produced, and to explain it during the presentation.

A–E. ENDURING HABITS: Each of these learning objectives describes a habitual response to a certain situation: (A) unexplained program behavior, (A and B) missing knowledge, (C) implementation options, (D) error reports, and (E) wanting complicated, general functionality. We will ask you to approach these coding situations conscientiously, with an eye for the habits that we've presented as solutions, and then to describe briefly one instance in which you used the habit to solve your coding problem.

How will you help us prepare to complete the final project?

One way that we'll help you complete your final project is by setting due dates for parts of the project. This encourages you to work on the project steadily throughout the quarter, rather than completing the whole thing the weekend before it's due. Here's a preview of what will be due when:

- **Week 2 (Apr 6):** Read this syllabus. Set up the workflow for your project.
- **Week 3 (Apr 12):** Take a crack at a tentative project idea.
- **Week 4 (Apr 19):** Take another crack. Write up the topic, scope, and data flow of your project.
- **Week 5 (Apr 26):** Get a piece of code working and document it. Habit summary 1.
- **Week 6 (May 3):** Write a piece of object-oriented or functional code. Habit summary 2.
- **Week 7 (May 10):** Research and implement an online library. Habit summary 3.
- **Week 8 (May 17):** Submit a plot of some preliminary results. Habit summary 4.
- **Week 9 (May 23):** Submit working rough draft. Habit summary 5.
- **Week 10 (May 31):** Submit final draft and present. Habit summary 6.

The other way we'll help you is by using our class sessions explicitly to help you learn and practice the skills you'll need to succeed on the final project. Each class session is divided roughly in half:

- **Interactive lecture:** Andrew and Kaitlyn will spend the first 50 minutes bringing you from where you are now to where you'll need to be to complete the lab. Lectures will begin with feedback to the class based on recent assignments. Much of the rest of the time Andrew or

Kaitlyn will be at the board demonstrating how new syntax and functionality work, modeling its use on the projector. They'll intersperse the lecture with brief practice questions and discussions to gauge your understanding and give you targeted practice. They'll also model techniques for workflow, debugging, and documentation, and help you organize and prioritize the concepts we'll be discussing.

- **Individual and group lab work:** You'll spend the last hour working on programming tasks we've prepared that give you targeted practice with the skills you'll be learning. We've set learning goals for each lesson (see Table 2), your progress towards which we'll assess by evaluating the lab assignments you submit. While the mid-lecture exercises will provide targeted practice of isolated skills, the labs will give students opportunities to practice integrating these skills with existing skills and applying them to realistic problems. We'll provide if-you-have-time problems for the lab so that learners at different levels of proficiency can challenge themselves appropriately.

  Labs will be submitted individually, but the lab format will take advantage of the students as a team. You'll be able to refer to your classmates for help, and in addition we'll use individual students' epiphanies as learning opportunities for everyone.

Table 2. CLASS OUTLINE

WEEK 1 Mar 29, Mar 31	Unix, workflow if/else, I/O, loops, funcs, strings	PROJECT MILESTONE DUE DATES
WEEK 2 Apr 5, 7	Text editors, Jupyter, Git lists, advanced data structs, types print debugging	Read syllabus; Set up workflow
WEEK 3 Apr 12, 14	NumPy arrays data flow file I/O NumPy and SciPy funcs how to read errors	First crack at project; pick topic, scope, data flow
WEEK 4 Apr 19, 21	basic plots functional programming how to read docs	Second crack at project
WEEK 5 May 26, 28	basic object-oriented programming how to write docs exceptions and testing	Piece of working, documented code; Habit summary 1
WEEK 6 May 3, 5	how to find libraries guest lecture project work time	Piece of OOP or functional code; Habit summary 2

WEEK 7 May 10, 22	regular expressions; how to make beautiful plots; project work time	Research, implement library; Habit summary 3
WEEK 8 May 17, 19	advanced topics; guest lectures; project work time	Plot of preliminary results; Habit summary 4
WEEK 9 May 24, 26	advanced topics; guest lectures; project work time	Working rough draft; Habit summary 5
LEARNING OBJECTIVES	I. SYNTAX & FUNCTIONALITY II. DATA FLOW & WORKFLOW III. DEBUGGING IV. PLOTTING & DOCUMENTATION	FINAL PROJECTS DUE Jun 1; PRESENTATIONS