Introduction to programming
Professor Benjamin Schmidt
benschmidt.org/code20
Email: bs145@nyu.edu
Office Hours: TBD: or please see my e-mail signature for booking slots.

Overview

In this class, you will learn to code. As a result, you will be finally be able to implement that app idea, convince your uncle that grad school was a good idea, and retire to a private island with a robot butler you build yourself that *doesn't* cache all your drink orders into a cloud registry.

… Not really. (That's a three-semester project, at least). But you will learn to program. The reason to learn to code in a humanities class is twofold.

One is that you have some project you want to do, and coding will help you do it. Computers are dumb, but fast. If you want to download thousands of books, count millions of words, categorize hundreds of pictures, you'd be foolish not to use code to do it.

Another is that you *don't* have a project yet, but would like to work one out.

This course is part of a triptych offered in the NYU digital humanities program. The other two introduce working with data and web development.

We use the python language, but self consciously make forays into understand general principles of programming.

Goals

1. Learn the fundamentals of programming in the Python language so that you can automate a variety of common tasks, such as searching through documents, downloading files, or altering/editing texts.

2. Understand the basic higher-order concepts of programming that Python enables, including the difference between different styles of programming–including imperative, functional, and object-oriented paradigms in python; and understand differnet data types and some fundamental data structures.

3. Be able to understand the social constraints and goals of computer languages–why does that tool not do what I want it to? How are larger programs designed and fit together?

4. Know how to use the fundamental infrastructure behind most modern programming projects, including iPython, git/Github, and the command line; know how to find and re-use existing code for your purposes.

Coding Languages

In this class, you'll learn Python, with a few glimpses at some other languages just for the sake of comparison.

We do this because Python has two features that make it ideal as the first (or only) language that you learn. First, it's widespread as an instructional language in undergrad CS courses because it's kind of a happy medium of all the different programming languages out there. It's amenable to a wide variety of styles of programming–what you'll learn to call imperative, object oriented, or functional paradigms. Second, it's the lingua franca of a particular sort of research computing particularly around artificial intelligence and the humanities themselves.

But what you learn in python transfers easily elsewhere. Don't be confused by the name–computer languages aren't really languages at all, and one is not nearly as difference from another as Spanish is from French.

For the record, you should know about the following other languages:

1. **Javascript**. This is the language the Web runs on, both in your browser and, increasingly, on the servers behind them. It's the only other general- purpose language I would think of using in an intro to programming course; we don't because while Python is well designed and consistent, Javascript has rightfully earned the reputation of being, let's say, kind of a mess. That said, if you want to write for the Web, you'll have to do it in Javascript. And I personally find the new flavors of the language more fun to write in than Python, although this is an extreme minority opinion.
2. **R**. The second- or third-most widely used language in the humanities (after Python and maybe Javascript) can do everything Python can, but is especially designed for working with data. That's why we use in the "Working with Data" class here in DHSS. Especially with the so-called `tidyverse` tools, it offers a clearer path to describing and modeling data directly than Python; but it is often more opaque when you want to do something like, say, shuffle the letters in a string.
3. `C` and `C++`; these are older languages that allow code to get closer to the performance limits of your machine because they require you to be more explicit about what you want than does Python. (Specifically, they make you)

Requirements

Writing computer programs is a practice that you must do to work at.

This class will require some concerted work on your part week-in, week-out. Since much of the benefit of working with other humanities students is that we all have a rather different way of thinking about what code is, and how it works, I've also endeavored to build in some assignments and readings that veer towards the disciplines of Critical Code Studies and public humanities.

Coding hours

A once-a-week seminar is often not especially suited to this type of thing. We'll discuss in the first day of classes whether to find an hour (or two hours) for more concrete digging. This might be one in-person and one remote, or two remote.

Textbook

This class will use Nick Montfort's Exploratory Programming for the Arts and Humanities as a text, **second edition**. This was published in May 2021; you're unlikely to find a used copy, but it's not very expensive.

Assignments

Assignments are noted weekly in the syllabus. Please submit them as Colab notebooks **shared with my NYU Google Account**. Be sure to restart and re-run your notebook before sending it to me–the most common source of errors in notebooks is mis-ordered cells.

Grading

30% Coding assignments. Each one is Pass/Fail–did you make a good effort to complete it? Is it clear you know where you're having trouble? Late assignments get 2/3 credit if they're in by the next session, 50% credit if they take longer.

30% Non-coding assignments.

25% Engagement. Do you help others? Do you ask productive questions? Do you attend the working hours?

15% Free assignments from Monfort. Do you find interesting and creative ways to apply the principles. Or find other ways to engage in unstructured application of programming, and call it a "freer exercise."

Policies

Inclusion Programming is a field that has catastrophic problems with inclusivity. It is extremely important that we not replicate them here. Please do your best to be sensitive, kind, and avoid anything that might make your classmates uncomfortable. On NYU classes, there is a web form to submit anonymous comments directed at either me or that can be passed to others in the class.

Academic Integrity Work you submit should be your own. Please consult the CAS academic integrity policy for more information: https://cas.nyu.edu/content/nyu-as/cas/academic-integrity.html – penalties for violations of academic integrity may include failure of the course, suspension from the University, or even expulsion. **That said, be aware that it is OK to copy and paste pieces of code–in fact, it is bad programming not to! I will not be upset if I find out that you copied and pasted anything less than 5-10 lines without attribution, though you really should.**

Religious Observance As a nonsectarian, inclusive institution, NYU policy permits members of any religious group to absent themselves from classes without penalty when required for compliance with their religious obligations. The policy and principles to be followed by students and faculty may be found here: The University Calendar Policy on Religious Holidays (http://www.nyu.edu/about/policies-guidelines-compliance/policies-and-guidelines/university-calendar-policy-on-religious-holidays.html)

Disability Disclosure Statement Academic accommodations are available for students with disabilities. The Moses Center website is www.nyu.edu/csd. Please contact the Moses Center for Students with Disabilities (212-998-4980 or mosescsd@nyu.edu) for further information. Students who are requesting academic accommodations are advised to reach out to the Moses Center as early as possible in the semester for assistance.

Schedule

The required text for this course is Nick Montfort's *Exploratory Programming for the Arts and Humanities.* We're working with the second edition, published in 2021. **You may not use the first edition**, because it teaches an older version of the Python language. I suggest buying the text online. If because of shipping issues you cannot get access to the text in the first week or two, please let me know. Additional IPython notebooks and explanations are at https://github.com/HumanitiesDataAnalysis/code20

Mon, Sep 06 No class: Labor Day

Introductions and thinking like a computer.

Mon, Sep 13 What is code?

Classroom activity

- Functions, variables, and letters: Google colab notebook.

Readings

- Paul Ford, "What is Code?" *Bloomberg*, 2015. (Yeah, seriously, Bloomberg. I sure didn't think my Ph.D. in History would have me assigning Bloomberg articles, either. But I think you'll see the relevance. Be aware this piece is **long**.)

Due Thu, Sep 16: Explore colab by making some letter point information and submit online. Worksheet online

Mon, Sep 20 Code criticism

Readings

- Mark Marino, *Critical Code Studies*, MIT Press, 2020, Introduction
- (Read Montfort, chapters 1, 3, and 4.)

agenda: Class agenda

Due Thu, Sep 23: Montfort Exercises 1 (Free project 3-1)

Mon, Sep 27 Fundamentals

Readings

- First: catch up on Montfort that we didn't do last week. Come prepared to talk about his attitude and philosophy.
- Then read more closely, typing in the examples, Montfort 5 (Double, Double) and 6 (Programming Fundamentals).

task: Ongoing: do something in python that raises an error, and post the code and the error to the dedicated Slack channel. As we figure out Slack intricacies, too, if you get the same error type with a different message as someone else, post that as a **reply** to their comment, not as a whole new thread; and try to get your own type of error. Before class, you should submit at least one error.

agenda: Class agenda

Due Mon, Sep 27: Montfort Free Project 5-1.

Due Thu, Sep 30: Montfort Exercises 5-1 to 6-6, free project 6-1.

Mon, Oct 04 Reboot

Readings

- Montfort 7 (Standard Starting Points)

Due Mon, Oct 04: Adopt an open-source project related to humanities research, cultural heritage, or another field of interest of yours. This will very likely be from the site github.com, but any online site that includes code files is acceptable. Before class on Slack, post a couple paragraphs on Slack about what you chose, including:

1. What does the project try to do?
2. What kind of files does it have?
3. How are they organized?

Note that you might not need to actually open any files yet–we'll get to it later.

Due Thu, Oct 07: Exercises 7-1 to 7-6; free project 7-1 to 7-3. Note that 7.1 tries to get you running python on your *own* computer. This is a worthwhile endeavor, but if you find it's not working, just post to the "Help Me" Slack channel and raise your issues in the next breakout section. Don't lose sleep if it's not straightforward–this is increasingly less important.

Mon, Oct 11 Back to strings

Readings

- Montfort 8 (Strings and Their Slices)

Due Mon, Oct 11: Describe a function from the project that you adopted–what does it do? Why did they write it? Does it call other functions? Link from Slack to the online portion of the code.

Due Thu, Oct 14: Montfort 8 (Exercises and Free project.)

Mon, Oct 18 Getting technical with strings

Readings

- Montfort 9 (Text 2: Regular Expressions)

Resources

- Montfort refers you to the Python documentation for more on regular expressions. I feel like a terrible person for saying this, but I've generally found the online Python documentation so comprehensive as to be almost unusable, so I don't really recommend reading it. You could also
- Do some cumulative exercises in the browser here: Regex One
- Use this Cheat sheet

Due Thu, Oct 21: Montfort 9, exercises and free projects.

Programming paradigms

Mon, Oct 25 Some things Montfort doesn't discuss, Part II.

description: Montfort's text describes, fundamentally, a *functional* approach to programming–that's why we've spent so much time talking about things like *recursion*, that you are unlikely to use. But an equally powerful model is *object-oriented* programming, in which the programmer defines reusable **classes**–things that bundle together information into themselves.

As you've seen looking over code online, the object-oriented method is very common in Python projects, especially large ones.

This week we're taking a digression from the Montfort text to expose you to the basics of this method.

Readings

- To follow up on some of the work manipulating strings, let's try some some heavy duty stuff: Can Computers Create Meanings? A Cyber:Bio:Semiotic Perspective. N Katherine Hayles. Critical Inquiry, 2019.
- "Python for Everyone," Chapter 14.

This isn't our normal textbook, so don't feel like you need to understand everything. In particular, all the stuff involving methods starting with two underscores ("**getitem**", etc.) are not things you need to understand. But you should understand the general idea of a class, including the key terms:

- class.
- instance.
- attributes and methods. (Analogous to variables and functions.)
- constructor (i.e., **init**) functions.
- The meaning of `self` in a method definition.

There's some slightly more advanced stuff it is safe to ignore for now. * destructor * inheritance * child class

Due Mon, Oct 25: TBD: Working with strings.

Mon, Nov 01 Tutorial following

readings: Montfort 10-11 on images (Don't invest **too** much time if you don't care about images.)

agenda: Class agenda

Due Mon, Nov 01: Creating the DHSS Tweet parser/National Archives Parser/etc." Fill in some methods so that we move towards a bot that can actually do something.

Due Thu, Nov 04: Montfort Exercises for Chapter 10.

Mon, Nov 08 Polyglotland

Readings

- Montfort 12 (Statistics, Probability, and Visualization) Don't do anything in Processing. Instead, explore some notebooks on ObservableHQ.
- Mar Hicks, "Built to Last," Logic Issue 11, August 31, 2020. link.

Due Mon, Nov 08: After reading Hicks, choose a language from the following list, arranged roughly in order of how likely you are to encounter them in the digital humanities. Find some code online that uses it. You are not going to learn this language, but write a few paragraphs on Slack describing the language with some examples, either code or images. (Don't worry about editing this too heavily, but do try to make it worth everyone's time to read). How it is like or unlike Python as we've experienced it? As far as you can glean, **why** does this language exist? Can you find anything online that indicates what sort of people use it? Your choices are: [Javascript, R, Ruby, Java, C, Perl, Go, C++, Haskell, Lisp, Fortran.] Claim it on Slack so we all get a different one.

Due Thu, Nov 11: Custom exercises, Observable Notebooks.

Applying your skills

Mon, Nov 15 Advanced Text

Readings

- Montfort 15 (Read regardless of whether you care about text.)
- OpenAI, Codex discussion. We'll see if we can get you all Copilot betas beforehand.

Mon, Nov 22 Working with APIs

readings: In class review of Organisciak and Capitanu, Text Mining in Python through the HTRC Feature Reader

Due Mon, Nov 22: Montfort 15, exercises. If you wish to work more heavily with images, let me know.

Mon, Nov 29 Web Browsing

Due Mon, Nov 29: Choose an API (or two) and get at least one item from it.

Due Mon, Nov 29: Find a website that looks like it might be scrapable for the assignment below.

Some important criteria:

1. It's a URL that everyone knows. I.e., not the New York Times, not CNN, not Instagram, etc. We're looking for something that a smaller person or organization put online.

Due Thu, Dec 02: Write a notebook to explore something in an API from online OR using Hathi Trust book count data OR the National Archives API.

It's OK to include private user credentials in the notebook you submit to me, but you can also save and export it.

What you do may differ. You could try:

1. Getting a bunch of data and merging it into some summary statistics.
2. Grabbing a bunch of text and playing with it using TextBlob
3. Grabbing some images and looking at–or mashing together–them using the PIL.

Mon, Dec 06 Images and beyond.

Due Wed, Dec 08: Write a python program that creates a list with at least 100 of something from the Internet. It can use *either* a scraper or an API. A few notes, here:

1. You must check the site's '/robots.txt' file to ensure that it's permitted to download from this source.
2. Include in your loop the following piece of code that forces the loop to wait three seconds between requests; otherwise you may bomb the server.

   ```
   import time
   time.sleep(3)
   ```

3. Don't do anything that requires sign-in/authentication.

Mon, Dec 13 TBD

Due Mon, Dec 13: Start building out a project that you could follow through on in the winter break. It doesn't need to be done, but it should be clear– delete unused code.

1. Explore the vast world of Colab notebooks and try adapting a neural network application *creatively*. Find a style transfer program online, a word embedding-based text replacement, whatever.
2. Continue building on your work with web scraping and/or APIs to produce a narrative notebook.

Agenda Notes

Notes for Mon, Sep 20

- Mozart Wuerfenspiel: http://www.playonlinedicegames.com/mozart
- On sorting: https://bost.ocks.org/mike/algorithms/
- House of Dust (online notebook): https://observablehq.com/@bmschmidt/house-of-dust

Notes for Mon, Sep 27

- Three things Montfort doesn't describe. Online notebook
  - **methods** and their distinction from **strings**.
  - dicts ({}).
  - error handling.

Notes for Mon, Nov 01

Object Orientation wrapup

Concepts:

1. Returning new copies of the thing itself.
2. Inheritance.

Major thanks to Nick Montfort for his exploratory programming syllabus at MIT.

I've also learned about these things over the years from Deena Engel, Ryan Cordell, Lauren Klein, and many others.

The language on university policies is taken from Sam Bowman's 2020 Machine Learning syllabus; I was just trying to get the standard Moses Center guidelines, etc., but it's possible he edited it.