Introduction to Programming

Matthew X. Curinga

Table of Contents

**Computer Science 0145-602, Fall 2020**

**Keywords:** computer programming, CS1, python, computational thinking, critical computational literacy, Mycroft, virtual assistant

**Description:** This course introduces students to programming and core concepts of computer science, using a modern, object oriented programming language (currently Python). Students learn concepts of variables, functions, repetition/loops, basic data structures (arrays, lists, dictionaries), and basic object oriented programming.

**Class meetings:** Online, asynchronous (coordinated through Moodle)

**Instructor**

* [Matthew X. Curinga](http://matt.curinga.com), [mcuringa@adelphi.edu](mailto:mcuringa@adelphi.edu)

**Dr. Curinga’s Office Hours by appointment**

* Monday, 3-4PM
* Thursday, 3-5PM

# Learning Goals

* understand the types of problems that can be solved using computational techniques
* understand the basic concepts of computation (CPU, RAM, permanent storage, GUIs, file systems, network connections)
* learn core computer programming concepts (abstraction, variables, conditions, functions, repetition, recursion)
* think algorithmically to design and test computer programs
* master the basic syntax and idioms of the Python programming language
* use technical documentation, APIs, and the internet to learn new technical concepts
* develop step-by-step problem solving and debugging practices

# Required Text

Downey, A. B. (2016). [*Think Python: How to Think Like a Computer Scientist, Version 2.4.0*](http://greenteapress.com/thinkpython2/html/index.html). Green Tea Press.

# Required Software/Online Accounts

* Software
  + [Slack Client](http://slack.com) (*must install* desktop and mobile clients, using the web client is not sufficient)
  + Firefox or Chrome web browser
* Accounts
  + [repl.it](https://repl.it)
  + [AU Ed Tech Slack channel #code](https://auedtech.slack.com/signup)

# Required Hardware

For the final project (see below) we will be designing and programming “skills” for the [Mycroft AI voice assistant.](https://mycroft.ai) Mycroft is like an open source Alexa or Siri. In order to work on your project, you will need a Raspberry Pi, microphone, and speakers, as well as display, keyboard, and mouse to work with the Pi. You can use any suitable hardware that you already have from the list below, but we also recommend certain items if you are purchasing new hardware.

I am posting links to the items on SparkFun and Amazon. You should be able to get most or all of the items locally, at [MicroCenter](https://www.microcenter.com/) (which may be the easiest option). You *must* have a working Raspberry Pi setup by week 4 (Sep 22). If you have any trouble acquiring equipment, please let me know as soon as possible.

* Raspberry Pi model 3 or 4, recommended Raspberry Pi 4 (2GB) [SparkFun](https://www.sparkfun.com/products/15446) [Amazon](https://www.amazon.com/Raspberry-Model-2019-Quad-Bluetooth/dp/B07TD42S27)
* a microphone that works with Raspberry Pi, recommended ReSpeaker 4-Mic Array [SparkFun](https://www.sparkfun.com/products/14645) [Amazon](https://www.amazon.com/seeed-Studio-ReSpeaker-4-Mic-Raspberry/dp/B076SSR1W1/)
* micro hdmi cable [SparkFun](https://www.sparkfun.com/products/15796) [Amazon](https://www.amazon.com/dp/B07VRCK5W1/)
* two (2) 16GB micro SD cards [Amazon](https://www.amazon.com/Micro-Center-Class-Memory-Adapter/dp/B07K81Z6DF/) (these are 32GB, but they’re a good deal)

You probably have these items already:

* USB-C power cord (like a USB-C phone charger) [SparkFun](https://www.sparkfun.com/products/15448) Amazon [[RPI official](https://www.amazon.com/Raspberry-Model-Official-SC0218-Accessory/dp/B07W8XHMJZ/)] [[AMZ Basics](https://www.amazon.com/dp/B07BDKK44S/)]
* TV or computer monitor with HDMI input
* USB or Bluetooth Mouse / Keyboard

# Class meetings

This is a fully online class, which will run on a Tuesday-Tuesday schedule, meaning new topics will begin each Tuesday, and assignments will be due by end of day on Monday. There will be a live video session lab held every-other Tuesday. No new topics will be covered in these labs that aren’t also in course materials, but they will give students the chance to ask questions and receive quick feedback on practice problems.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Week | Date | Topic | Read | Due |
| 1 | 1-Sep | Critical Computational Thinking | TIP 1 | - |
| 2 | 8-Sep | Variables & Statements | TIP 2 | - |
| 3 | 15-Sep | Functions | TIP 3 | Quiz 1 |
| 4 | 22-Sep | Case Study 1: Hello World | - | Mycroft Concept |
| 5 | 29-Sep | Conditionals & Recursion | TIP 5 | - |
| 6 | 6-Oct | Fruitful Functions | TIP 6 | - |
| 7 | 13-Oct | Iteration | TIP 7 | Quiz 2 |
| 8 | 20-Oct | Strings | TIP 8 | - |
| 9 | 27-Oct | Case Study 2: Attendance | - | Mycroft Prototype |
| 10 | 3-Nov | Lists | TIP 10 | - |
| 11 | 10-Nov | Dictionaries | TIP 11 | - |
| 12 | 17-Nov | Tuples | TIP 11 | Quiz 3 |
| 13 | 24-Nov | Files | TIP 12 | - |
| 14 | 1-Dec | Case Study 3: Random Groups | - | Mycroft UX Testing |
| 15 | 8-Dec | Working Session | - | - |
| 16 | 15-Dec | Final Projects Due | - | **Final Project** |

TIP: *Thinking in Python*

# Assignments and Grading

|  |  |
| --- | --- |
| Assignment | Pct |
| Quizzes | 60% |
| Final Project | 40% |

## Lab Exercises

Most weeks there will be ungraded lab exercises where students can practice the new materials covered. In general, you should spend about one hour working on these exercises. If you understand the exercises, you will be on track with the course. There may be bonus problems that are a little bit more challenging, which are optional. You are encouraged to work on the exercises with other students and friends.

## Quizzes

Quizzes will consist of 5 questions, similar to the lab exercises. Each question is worth 4 points, with possible credit available.

* *0-1 points*: for not turning in any work, or code that does not address the problem
* *1 point*: for a basic attempt, but code isn’t working or has fundamental flaws
* *3 points* (mostly) solution demonstrates mastery of relevant concepts, but doesn’t work in all cases or fails due to minor errors
* *4 points*: solution works, demonstrates mastery of concepts, and is well formatted and clearly written

Quizzes will be timed, taken through Moodle. You can use the textbook, course examples, and any documentation or internet resources you find. **You may not ask other people for help.** While programming is a highly collaborative practice, these quizzes are meant to assess *your* work and understanding.

## Final Project: Mycroft Skill

During the course of the semester we will be learning about the software design process, and have the opportunity to write a larger program. This is a group project, and you should work in a group of 2-4 students.

[Mycroft](https://mycroft-ai.gitbook.io/docs/about-mycroft-ai/why-use-mycroft) is a free open source software voice assistant. It is not as advanced as better known products from big tech companies: Siri, Alexa, Cortana, Google Assistant. However, it works very well and is much easier to get started programming with. Also, since the project is designed to protect users and their data, it is a good fit for classrooms and students.

Working with your team, you will design a new Mycroft “skill.” Skills are the “apps” of the Mycroft system. You activate them with a spoken phrase and Mycroft can ask follow up questions or speak a response. Mycroft can also affect other changes, if it’s connected properly, such as changing lights, playing audio or video, or controlling motors or sensors connected to your Raspberry Pi. See [What can a Skill do?](https://mycroft-ai.gitbook.io/docs/skill-development/voice-user-interface-design-guidelines/what-can-a-skill-do) from the Mycroft developers guide in order to get a better sense of what you might make.

For this project, you will conceive of a novel Mycroft skill, refine the skill by talking to people who are in the target audience, and code the skill through an iterative process of developing, testing, and improving.

### Final Project Grading:

There are a total of 40 possible points for this assignment, which will be evaluated on the following criteria:

1. **Design** (*5 points*)  
   How well is the skill designed? Does it solve a real problem or need in the world? Has the team spoken to enough real users to understand the problem? How usable is the skill? Is it easy to learn and understand? Has it been validated and revised through user testing?
2. **Coding fundamentals** (*15 points*)  
   The program demonstrates a grasp of the programming concepts covered in this class, including:
   * *variables*
     + data is not “hard coded” and can be easily changed by using variables, soliciting user input, and/or reading from files
     + data is separate from functionality
   * *functions*
     + abstraction through function parameters
     + encapsulation through function parameters and return statements; use “pure functions” with zero-side effects when possible
     + composition and re-use of code
   * *design*
     + the program is organized through the use of functions
     + functions’ “scale” is appropriate to the task and discrete: concerns are separated logically, such as one function for gathering results and another for outputting results
     + functions are written in a way that they are used several times in the program
     + code is not repeated
   * *data structures*
     + use python built-in data structures appropriately: list, dict, tuple, set, etc
     + use index/slice notation if needed
     + sort data structures
     + map, filter data as needed
3. **Risk Taking** (*5 points*)  
   How “adventurous” is this code? Does the team show that they move beyond the template given to them? Do they come up with a really novel and desirable project? “Riskier” project push beyond the material strictly covered in class and demonstrate the teams’ ability to learn new things and push their horizons.
4. **Code Style** (*2 points*)  
   Is the program consistently formatted according to Python conventions?
   * 2 points: consistently follows the spirit of the *Think Python* and PEP 8 style guides.
   * 1 point: follows guide most of the time, shows internal consistency for style
   * 0 points: lack of consistency in style makes the program harder to read and (potentially) harder to debug and maintain
5. **Usability & Usefulness** (*3 points*)  
   The usability is different from the design, in that it incorporates things such as the usefulness of the results provided, speed and accuracy of the software, and the total experience of working with the skill.

### Deliverables:

1. **Code Files and Resources**  
   You should turn in all files related to your project in a single archive (e.g., .zip, .tar, .7z). Project files *must* contain your **Python program** and any data and media files that are required.

* To create an archive file, Mac users can simply right-click the project folder from the finder and choose “Compress”. This will create a .zip archive of the project directory. Windows does not come with a compression utility by default. If you do not have one installed or are not sure, Adelphi IT recommends [7-zip, which you can download and use for free.](http://7-zip.org/)

1. **Video Demo**  
   You should provide a narrated screencast or video of 4-5 minutes that demonstrates your skill, running on your Raspberry Pi. Your group can decide who to divide the work. Upload the video to YouTube and provide the link when you submit your assignment to Moodle.