Introduction to Programming

Matthew X. Curinga

Table of Contents

**Computer Science 0145-602 Section 002 (online), Fall 2019**

**Keywords:** computer programming, CS1, javascript, computational thinking, critical computational literacy, problem solving

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

We are looking at a society increasingly dependent on machines, yet decreasingly capable of making or even using them effectively. ― Douglas Rushkoff, *Program or Be Programmed: Ten Commands for a Digital Age*

**Class meetings:** *fully online*

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

**Office hours:**

* Monday 1-2pm, Alumnae Hall Room 226A (Garden City campus)
* Wednesday 4:30-5:30pm, online
* Thursday 2:30-4:30pm, Alumnae Hall Room 226A (Garden City campus)
* *office hours by appointment*

# 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 Javascript programming language
* use technical documentation, APIs, and the internet to learn new technical concepts
* develop step-by-step problem solving and debugging practices

# Required Books

* [Think Javascript](https://mcuringa.github.io/think-js/) (free online)

# Required Software/Online Accounts

* Software
  + [Slack Client](http://slack.com) (recommend desktop and mobile clients)
  + Firefox or Chrome web browser
* Accounts
  + [repl.it](https://repl.it)
  + [AU Ed Tech Slack channel #code](https://auedtech.slack.com/signup)

# Class Sessions

|  |  |  |  |
| --- | --- | --- | --- |
| Week | Date | Topic | Due |
| 1 | Aug 29 | The way of the program | - |
| 2 | Sep 05 | Problem solving in computer programming | Lab 1 |
| 3 | Sep 12 | Variables and data | Lab 2 |
| 4 | Sep 19 | Functions and testing | Lab 3 |
| 5 | Sep 26 | Conditions | Lab 4 |
| 6 | Oct 03 | Repetition with while | Lab 5 |
| 7 | Oct 10 | Strings and repetition with for | *study!* |
| 8 | Oct 17 | Midterm | - |
| 9 | Oct 24 | Arrays | - |
| 10 | Oct 31 | Objects & Sets | Lab 6 |
| 11 | Nov 07 | JSON data | Lab 7 |
| 12 | Nov 14 | Networks and files | Lab 8 |
| 13 | Nov 21 | Graphs & Charts | Lab 9 |
| 14 | Nov 28 | Final project design (Happy Thanksgiving) | Lab 10 |
| 15 | Dec 05 | Final project workshops | - |
| 16 | Dec 12 | Finals | Final project |

# Assignments and Grading

|  |  |  |
| --- | --- | --- |
| Assignment | Pct | Due date |
| Labs (10 total) | 50% | Sep 13 |
| Midterm exam | 25% | Oct 18 |
| Final project | 25% | Dec 14 |

## Labs (50%)

Most weeks there will be a programming lab due. Labs will consist of a single exercise that focuses on using the computer programming concepts we’re learning to solve a problem. Each lab will be worth 0-5 points. Roughly:

* *0 points*: for not turning in any work
* *1-2 points*: for a basic attempt, but code isn’t working or has fundamental flaws
* *4-5 points* (mostly) solution demonstrates mastery of relevant concepts
* *5 points*: solution works, demonstrates mastery of concepts, and is well formatted and clearly written

Everyone’s lowest grade will be thrown out (so your lab score will be the average of your 9 highest lab grades).

## Midterm (25%)

The midterm will consist of 5 “short answer” styled problems where you write a function for each prompt to solve a specific problem. The short answer prompts will be very similar to the textbook exercises from *Think JS*. This section is worth 10 points.

Part 2 of the exam will require writing a longer program that consists of several functions. Students will choose to answer 1 of 3 possible problems. The program should demonstrate the student’s ability to break down a problem and write a working computer program that provides a solution.

## Final project (25%)

For the final project you will work in a team of 2-3 people to create your own data analysis and visualization using a data set that is available for download or as a live open data set such as the [NYC Open Data](https://opendata.cityofnewyork.us/).

Of course, you may incorporate other data sets as you see fit.

\*\*The final solution will be scored using the following guide:

**Solution (7 points)**

How well does the program written solve the problem? Does the project seriously engage with the data? Does it use compute analysis to provide interesting insights? Is the data presented in a meaningful and usable way?

Your team’s ability to design a solution to the problem is evaluated by this measure.

**Elegance & Robustness (8 points)**

This area evaluates the quality of the computer code produced by the team. An *elegant* program provides a parsimonious solution that is both efficient and clear. A *robust* program is flexible and able to change. In the case of data analysis, it would be easy to modify if the input data changed or the output requirements are updated. It wouldn’t “break” if it encountered unexpected data, and would continue to operate even if the amount of data were increased dramatically. Often, both elegance and robustness are achieved through *refactoring*: the process of reflecting on code and revising it after an initial working solution is achieved.

**Risk Taking (5 points)**

Learning should be an adventure. One of the most exciting things about writing software is the sense of new possibilities and discovery. The “risk taking” aspect of your team’s grade will reflect the chances that you take with your project. Even if your solution isn’t quite what you hoped for, or your code isn’t as elegant as you’d like, it’s important that you take chances and try new things. To do well in this section, you might want to shoot for an ambitious, or complex analysis; or integrate some Javascript libraries or techniques that haven’t been explicitly covered in the course.

**Code Style (5 points)**

Your code should be well formatted and easy to read. Your functions and variables (aka “identifiers”) should have clear, meaningful names. Comments should be used sparingly, but appropriately to guide the human reader through your code.

*All project members will receive the same grade.*

## Javascript Documentation and References

* [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/javascript)
* [W3 Schools](https://www.w3schools.com/js/default.asp)
* [OverAPI](http://overapi.com/javascript)
* [ES6 Lang Spec](https://www.ecma-international.org/ecma-262/6.0/index.html)

## Books & Tutorials

* [Javascript the Right Way](http://jstherightway.org/)
* [You Don’t know JS (book series online)](https://github.com/getify/You-Dont-Know-JS)
* [Eloquent JavaScript](https://eloquentjavascript.net/)
* [Understanding Programming through JavaScript](https://cs.stanford.edu/people/eroberts/CS106AJ-Reader.pdf)
* [Mastering Regular Expressions](http://shop.oreilly.com/product/9780596528126.do)
* [JavaScript & jQuery: Interactive Front-End Web Development Hardcover](http://www.wiley.com/WileyCDA/WileyTitle/productCd-1118871650.html), also J. Duckett, same series