

Mid-Quarter Review

CSE 120 Spring 2017

Instructor:

Justin Hsia

Teaching Assistants:

Anupam Gupta, Braydon Hall, Eugene Oh, Savanna Yee

Administrivia

- ❖ Assignments:
 - Color Filters due tonight (5/8)
 - Word Guessing due Thursday (5/11)
 - Living Computers Museum Report due Sunday (5/14)
- ❖ Midterm & Creativity Assignment grades released
 - + 0.75 points to everyone's midterm scores
- ❖ Guest lecture on Friday: Security

Word Guessing

- ❖ Learn to use text input & output
 - Player 1 enters a secret phrase
 - Player 2 tries to guess the secret phrase
 - Game tells you how many letters correct & # of attempts
- ❖ Strings (e.g. `String str = "hello";`)
 - `'c'` is a character, `"c"` is a String, `"cool"` is a String
 - Strings are *sort of* like arrays of characters
 - Can get individual character using `str.charAt(i)`, starting at 0
 - Can get length using `str.length()`
 - Can concatenate using `'+'` operator
 - e.g. `"hello, " + "world!"` gives you `"hello, world!"`

Outline

- ❖ **Mid-Quarter Survey Feedback**
- ❖ Coding Style
- ❖ Programming Tips
- ❖ Assignments Review

Lecture

- ❖ Polls are too fast
- ❖ Lectures feel rushed, especially at the end
- ❖ Upload lectures ahead of time
- ❖ More coding examples
 - Go over issues with assignments?

Section

- ❖ More examples, preferably related to recent lectures
- ❖ Handouts for presentation information
- ❖ Time management

Reading and Discussions

- ❖ Readings are interesting, but discussions are lacking
 - Awkward silences if students don't answer
 - Readings seem long-ish
- ❖ Why is the timing set the way it is?
- ❖ TAs will try to expand beyond what is asked in the Reading Checks
- ❖ **Knowledge is not useful if you can't *apply it***
 - Defend your position, explain to someone else, etc.

Innovation Exploration

- ❖ Mini-research project to let you explore a computing topic *that is interesting to you*
 - Pick a recent and relevant topic
 - Think of this as your “project” for the reading & writing portion of this course

- ❖ **Part 1: Innovation Post (5/21)**
 - 4+ paragraphs, 550-750 words – posted to Canvas discussion board
 - Well-researched, insightful post, including 3+ citations
 - *Purpose, Effects and Impacts, Technical Aspects*

- ❖ **Part 2: Respond to Posts (5/26)**
 - Comment on 3+ other students’ posts

Outline

- ❖ Mid-Quarter Survey Feedback
- ❖ **Coding Style**
- ❖ Programming Tips
- ❖ Assignments Review

Why is Style Important?

- ❖ Makes code understandable!
 - Functional abstraction relies on well-documented *interfaces*
 - You or others may need to look at your code in the future
 - Fixing things, change in computation problem, forgotten details
- ❖ Much of software engineering today is built on top of the work of others
 - Large projects are worked on in groups
 - Most programming languages provide *libraries* of functions for you to use

CSE120: Indentation

- ❖ Amount of white space in Processing generally doesn't matter, but proper indentation improves readability
 - Bonus: helps readability when you are debugging
- ❖ Anything within a block of code should be indented one level farther in
 - *e.g.* function definition, conditional statement, loops
 - As blocks get nested within each other, indentation continues to increase

CSE120: Types of Commenting

- ❖ Individual lines of code
 - Off to the right side

- ❖ Blocks of code (stuff between { })
 - *e.g.* function definition, conditional statement, loops
 - Because these describe collections of instructions, should be more descriptive of *purpose* and *usage*
 - On line above, at same indentation level

- ❖ Program block comment
 - At top of program – first thing anyone reads

Outline

- ❖ Mid-Quarter Survey Feedback
- ❖ Coding Style
- ❖ **Programming Tips**
- ❖ Assignments Review

Programming Reminders

- ❖ Programming is commanding an *agent* to achieve a *goal* by giving it *instructions*
 - The agent follows the instructions flawlessly and mindlessly
 - The trick is to find the right instructions to match your *intent*
- ❖ Programming requires you to take the agent's point of view
 - Because it is a sequence of instructions, you must account for everything that happened before (*i.e.* **trace** the program)

Building Blocks of Algorithms

❖ Sequencing

- The application/execution of each step of an algorithm in the order given

```
fill(255);  
rectMode(CORNERS);  
rect(-r, -r, 0, r);  
ellipse(0, -r/2, r, r);
```

❖ Iteration

- Repeat part of algorithm a specified number of times

```
for(int i=20; i<400; i=i+60) {  
    line(i, 40, i+60, 80);  
}
```

❖ Selection

- Use of conditional to select which instruction to execute next

```
if(mousePressed) {  
    fill(0, 0, 255);  
}
```

❖ Recursion

- Algorithm calls itself to help solve the problem on smaller parts

Testing

- ❖ Manually tracing your code
 - Come up with a set of inputs to test, then follow your program's execution line-by-line to see if the outcome matches what you want
- ❖ Trial and Error
 - **Unit Test**: Test an individual function on a representative set of inputs
 - **Integration Test**: Run the entire program and see if it behaves as it should
- ❖ Other methods exist (*e.g.* formal code verification)

Debugging Tips

- ❖ “Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime.”
- ❖ Always start with *simple* examples
 - Easier to trace example through your code
- ❖ If doing calculations (*e.g.* arithmetic, loop updates), double-check that you are getting the values that you want
 - Can print values to console or drawing canvas
 - `println()`, `text()`, colors or other drawing clues if you're clever

Debugging Tips

- ❖ Don't just randomly tweak things until it works – understanding your errors is always beneficial
 - Correct your own misunderstandings
 - Random tweaks may lead you further away or make your code harder to understand
- ❖ Learn to interpret the Processing error messages
 - Some can be Googled, or just ask on Piazza

General Tips

- ❖ If you're unsure of *what* something in your code accomplishes or *how*, it's critical that you figure it out
 - Try following the program code instruction-by-instruction
 - Talk to your partner
 - Ask in office hours or on Piazza
 - Comment your code as you go!
- ❖ Functional abstraction is your friend!
 - Use functions to hide away details (combat *monolithic code*)
 - If you find yourself using repeated or similar instructions over and over again, consider rewriting in a function or loop

Refactoring

- ❖ Despite your best planning, you can't always map out the cleanest solution from the start
 - Often times you'll just start coding and adjust as you go
- ❖ **Refactoring** is the process of restructuring existing code without changing its external behavior
- ❖ After refactoring, it's a good idea to go back and test everything again to make sure you didn't break something
 - Good to save a working copy of your code separately before you make major changes

Outline

- ❖ Mid-Quarter Survey Feedback
- ❖ Coding Style
- ❖ Programming Tips
- ❖ **Assignments Review**

Jumping Monster

- ❖ Main Topics:
 - Functions, program “state”, motion

Controlling Elli

- ❖ Main Topics:
 - **Arrays**, Variables, keyPressed

Recursive Tree

- ❖ Main Topics:
 - Recursion, **Control Flow**