Intensive Introduction to Computer Science

Course Overview
Programming in Scratch

Computer Science S-111
Harvard University
David G. Sullivan, Ph.D.

Welcome to CS S-111!

*Computer science is not so much the science of computers as it is the science of solving problems using computers.*

Eric Roberts

• This course covers:
  • the process of developing *algorithms* to solve problems
  • the process of developing computer programs to express those algorithms
  • fundamental *data structures* for imposing order on a collection of information
  • the process of *comparing* data structures & algorithms for a given problem
Computer Science and Programming

- There are many different fields within CS, including:
  - software systems
  - computer architecture
  - networking
  - programming languages, compilers, etc.
  - theory
  - AI

- Experts in many of these fields don’t do much programming!

- However, learning to program will help you to develop ways of thinking and solving problems used in all fields of CS.

A Rigorous Introduction

- Intended for:
  - those who plan to work extensively with computers
  - future concentrators who plan to take more advanced courses
  - others who want a rigorous introduction
  - no programming background required, but can also benefit people with prior background

- Allow for **20-30 hours** of work per week
  - start work early!
  - come for help!
  - don’t fall behind!

- Computer Science S-1 is a less rigorous alternative
CS 111 Requirements

- Lectures and sections

- Ten problem sets (50%)
  - part I = "written" problems
  - part II = "programming" problems
  - grad-credit students will have extra work on most assts.

- Nine unit tests (25%)
  - given at the end of lecture (see the schedule)
  - 25 possible pts. for each
  - if score lower than 18, can take a retest for a max. of 18

- Final exam (25%): Friday, August 4, 8:30-11:30 a.m.
  - comprehensive exam

Textbooks

- **Required**: The CSCI S-111 Coursepack
  - contains all of the lecture notes
  - will be available at Gnomon Copy on Mass Ave.

- **Optional** resource for the first half:
  *Building Java Programs* by Stuart Reges and Marty Stepp
  (Addison Wesley, 2016).

- **Optional** resource for the second half:
  *Data Structures & Algorithms in Java, 2nd edition* by
Other Course Staff

- Teaching Assistants (TAs):
  - Cody Doucette (head TA)
  - Caitlin Fournier
  - Justin Ingwersen
  - Eli Saracino

- See the course website for contact info.

- **Piazza is your best bet for questions.**

- For purely administrative questions: libs111@fas.harvard.edu
  - will forward your email to the full course staff

Other Details of the Syllabus

- **Schedule:**
  - note the due dates and test dates
  - no lectures or sections on most Wednesdays
    - **exceptions:** July 5 (July 4 is off), July 12 (July 14 is off), August 2 (August 3 is off)

- **Policies:**
  - 10% penalty for submissions that are one day late
  - please don't request an extension unless it's an emergency!
  - grading

- Please read the syllabus carefully and make sure that you understand the policies and follow them carefully.

- Let us know if you have any questions.
Algorithms

• In order to solve a problem using a computer, you need to come up with one or more algorithms.

• An algorithm is a step-by-step description of how to accomplish a task.

• An algorithm must be:
  • precise: specified in a clear and unambiguous way
  • effective: capable of being carried out

Example of Defining an Algorithm
Programming

• Programming involves expressing an algorithm in a form that a computer can interpret.

• We will primarily be using the Java programming language.
  • one of many possible languages

• The key concepts of the course transcend this language.

What Does a Program Look Like?

• Here's a Java program that displays a simple message:

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

• Like all programming languages, Java has a precise set of rules that you must follow.
  • the syntax of the language

• To quickly introduce you to a number of key concepts, we will begin with a simpler language.
Scratch

• A simple graphical programming language
  • developed at the MIT Media Lab
  • easy enough for middle school kids
  • makes it easy to create animations, games, etc.

• Download version 1.4 for free here:
  http://scratch.mit.edu/scratch_1.4/
  • this is not the latest version, but it's the one we will use

Scratch Basics

• Scratch programs (scripts) control characters called sprites.

• Sprites perform actions and interact with each other on the stage.
Sprites

- At the start, there is a single cat sprite on the stage.
- You can add and remove sprites, and change their costumes.

- Clicking on a sprite from the list below the stage shows you the scripts for that sprite (if any).

Program Building Blocks

- Grouped into eight color-coded categories

- The shape of a building block indicates where it can go.
- Right-click a building block to get help on that type of block.
**Program Building Blocks: Statements**

- Statement = a command or action
  ```
  say Hello!
  move 10 steps
  turn left 15 degrees
  if on edge, bounce
  hide
  play sound meow
  pen down
  wait 1 secs
  set pen size to 1
  ```

- Statements have bumps and/or notches that allow you to stack them.
  - each stack is a single script

- A statement may have:
  - an input area that takes a value (Hello!, 10, 15, etc.)
  - a pull-down menu with choices (meow)

**Program Building Blocks: Statements (cont.)**

- Clicking on any statement in a script executes the script.

- When rearranging blocks, dragging a statement drags it and any other statements below it in the stack.
  - example: dragging the wait command below
Flow of Control

• Flow of control = the order in which statements are executed

• By default, statements in a script are executed sequentially from top to bottom when the script is clicked.

• Control blocks (gold in color) allow you to affect the flow of control.
  • simple example: the wait statement above pauses the flow of control

Flow of Control: Repetition

• Many control statements are C-shaped, which allows them to control other statements.

• Example: statements that repeat other statements.

• Drag statements inside the opening to create a repeating stack.

• In programming, a group of statements that repeats is known as a loop.
Flow of Control: Responding to an Event

• *Hat blocks* (ones with rounded tops) can be put on top of a script.

  ![Hat blocks diagram](image)

• They wait for an event to happen.
  • when it does, the script is executed

What Does a Program Look Like?

• Recall our earlier Java program:

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

• Here’s the Scratch version ... and here’s the result:

  ![Scratch diagram](image)
Stage Coordinates

- Dimensions: 480 units wide by 360 units tall
- Center has coordinates of 0, 0

What Does This Program Draw?
What Changes Are Needed to Draw Each of These?

Flow of Control: Repeating a Repetition!

- One loop inside another loop!
  - known as a *nested loop*

- How many times is the *move* statement executed above?
Making Our Program Easier to Change

- It would be nice to avoid having to manually change all of the numbers.
- Take advantage of relationships between the numbers.
  - what are they?

Program Building Blocks: Variables

- A variable is a named location in the computer's memory that is used to store a value.
- Can picture it as a named box: numSides
- To create a variable:
Using Variables in Your Program

Program Building Blocks: Operators

- Operators create a new value from existing values/variables.

note: you must drag a variable into place, not type its name
Our Program with Variables and Operators

Getting User Input

- Use the *ask* command from the *sensing* category.

- The value entered by the user is stored in the special variable *answer*, which is also located in the sensing category.

- Allowing the user to enter *numSides* and *numCopies*:
Program Building Blocks: Boolean Expressions

- Blocks with pointed edges produce boolean values:
  - true or false

- Boolean operators:
  - `<` Reports true if first value is less than second.
  - `>` Reports true if first value is greater than second.
  - `=` Reports true if two values are equal.
  - `and` Reports true if both conditions are true.
  - `or` Reports true if either condition is true.
  - `not` Reports true if condition is false; reports false if condition is true.

- There are also boolean expressions in the sensing palette:

Flow of Control: Conditional Execution

- Conditional execution = deciding whether to execute one or more statements on the basis of some condition

- There are C-shaped control blocks for this:

- They have an input area with pointed edges for the condition.
Flow of Control: Conditional Execution (cont.)

- If the condition is true:
  - the statements under the \textit{if} are executed
  - the statements under the \textit{else} are \textbf{not} executed

- If the condition is false:
  - the statements under the \textit{if} are \textbf{not} executed
  - the statements under the \textit{else} are executed

Dealing With Invalid User Inputs
More Info on Scratch

- Documentation is available online:
  - getting started guide for version 1.4:
    
  - reference guide for version 1.4:
    

- Creating a Scratch account is not required for this course.