Unit 1, Part I

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:
  - 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!

CS 111 Requirements

- Lectures and sections
  - attendance at both is required
- Ten problem sets (40%)
  - part I = "written" problems
  - part II = "programming" problems
  - grad-credit students will have extra work on most assts.
- Four unit tests (25%)
  - given at the end of lecture (see the schedule)
- Final exam (35%)
  - Friday, August 7, 8:30-11:30 a.m.

Textbooks

- **Required**: *The CSCI S-111 Coursepack*
  - contains all of the lecture notes
  - print it and mark it up during lecture
- **Optional** resource for the first half:
  Building Java Programs by Stuart Reges and Marty Stepp (Addison Wesley).
- **Optional** resource for the second half:
  Data Structures & Algorithms in Java, 2nd edition by Robert Lafore (SAMS Publishing).

Other Course Staff

- Teaching Assistants (TAs):
  Ashby Hobart
  Libby James
- See the course website for contact info. and office hours
- **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 1 (July 3 is off), July 8 (July 10 is off), August 5 (August 6 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

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?

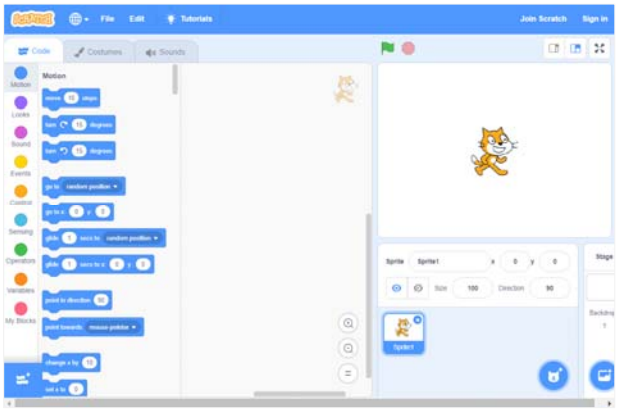
- Here's a Java program that displays a simple message:

```
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.

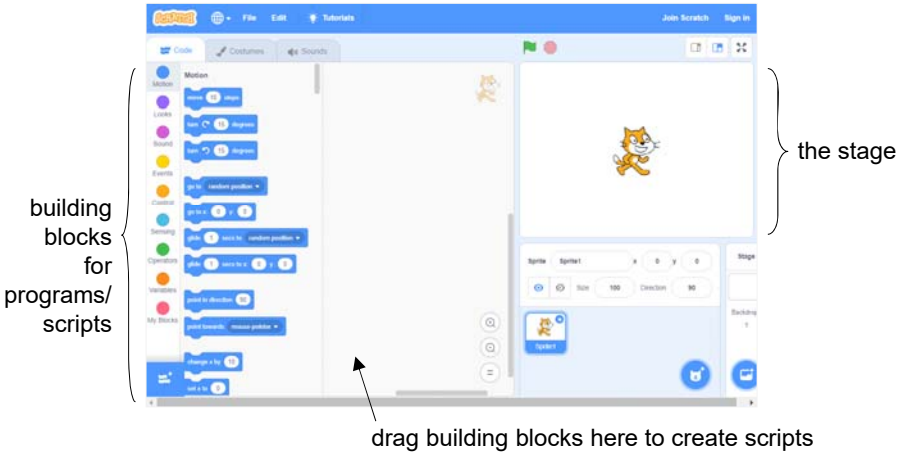
Scratch

- A simple but powerful graphical programming language
  - developed at the MIT Media Lab
  - makes it easy to create animations, games, etc.



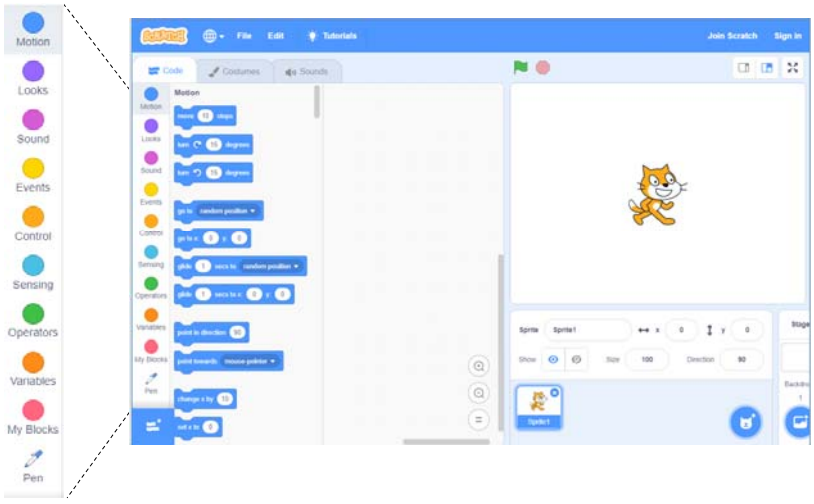
Scratch Basics

- Scratch programs (*scripts*) control characters called *sprites.*
- Sprites perform actions and interact with each other on the *stage.*



Program Building Blocks

- Grouped into color-coded categories:



- The shape of a building block indicates where it can go.

Program Building Blocks: Statements

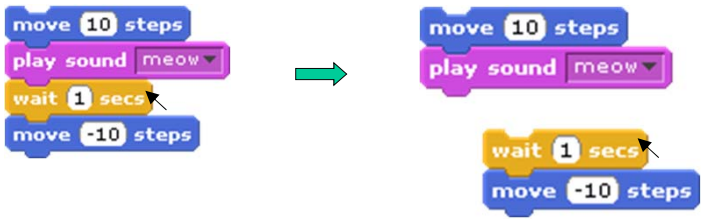
- Statement = a command or action

- 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 (10, 1, etc.)
  - a pull-down menu with choices (meow)

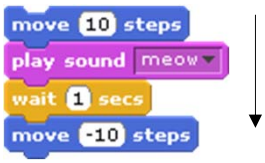
Program Building Blocks: Statements (cont.)

- Clicking on any statement in a script executes the script.
- When rearranging blocks, dragging a statement drags it *and* any other statements <u>below it</u> 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

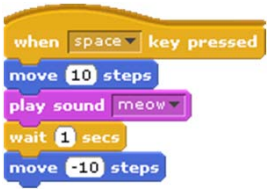
- Many control statements are C-shaped, which allows them to control other statements.
- Example: statements that repeat other statements.

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

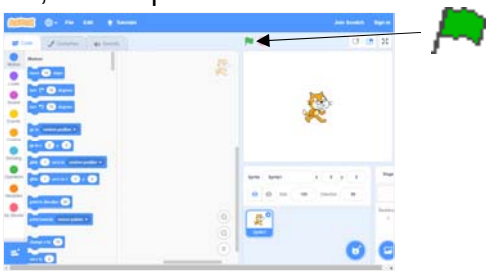
- In programming, a group of statements that repeats is known as a *loop*.

Flow of Control: Responding to an Event

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



- 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:

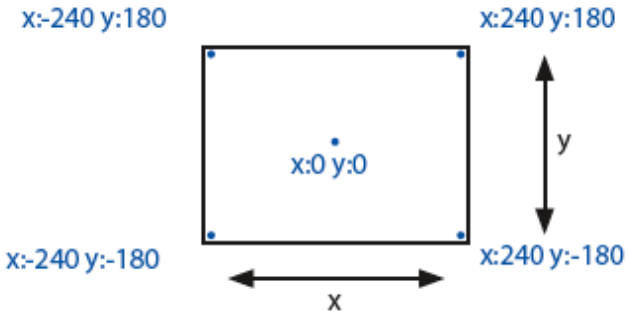
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello, world");
    }
}
```

- Here's the Scratch version

… and here's the result:

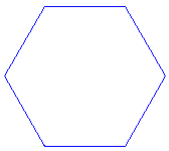
Stage Coordinates

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

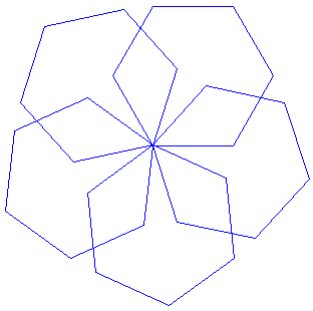


What does this program draw?

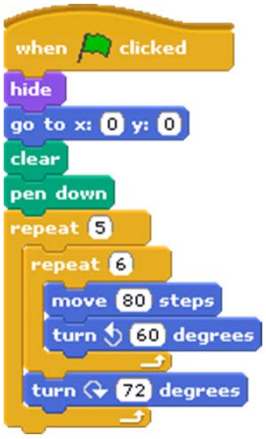
How many changes would be needed
to draw this figure instead? (What are they?)



How could we draw this figure?

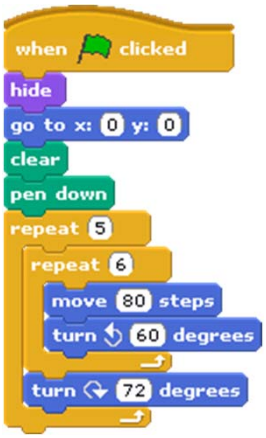


Flow of Control: Repeating a Repetition!



- 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

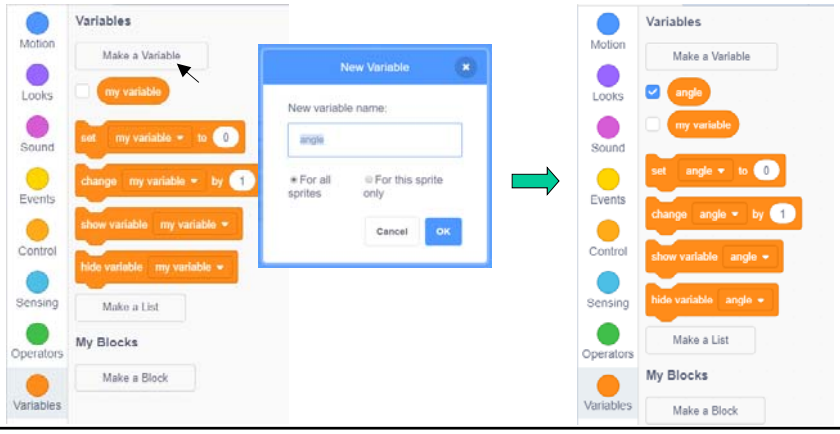


- It would be nice to avoid having to manually change *all* of the numbers.
- Take advantage of relationships between the numbers.
  - what are they?

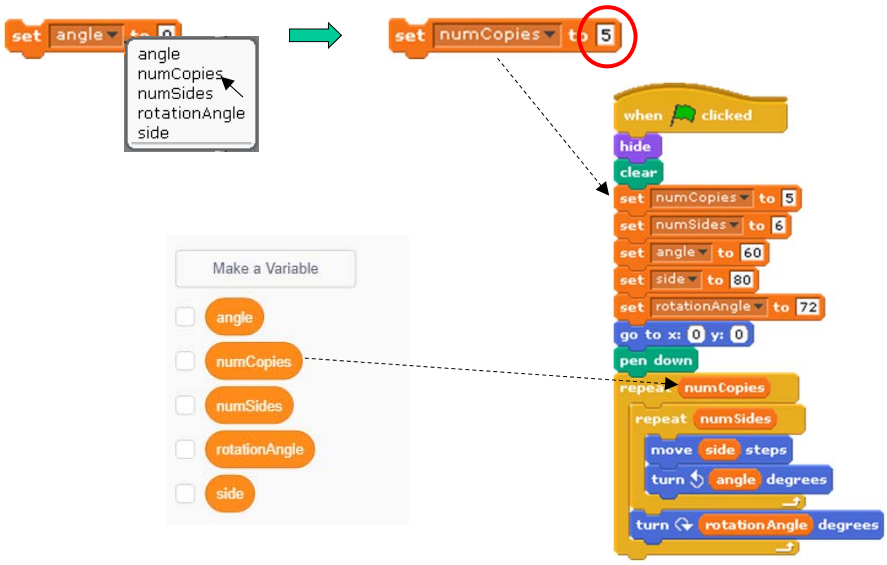
Program Building Blocks: Variables

- 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: 

- To create a variable:



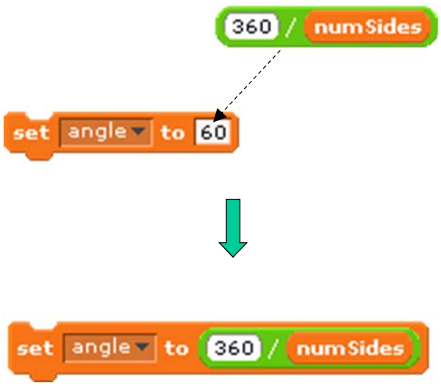
Using Variables in Your Program



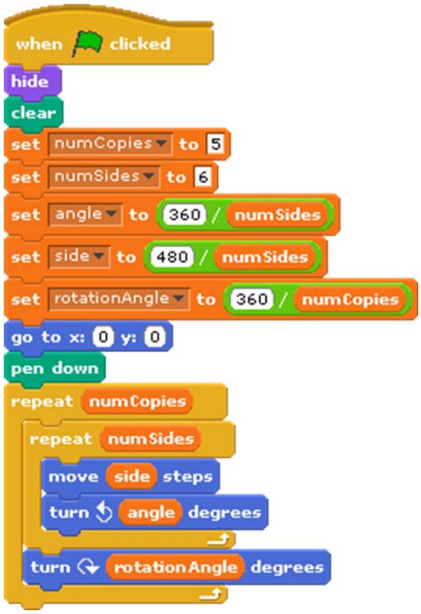
note: you must drag a variable into place, <u>not</u> type its name

Program Building Blocks: Operators

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



Our Program with Variables and Operators

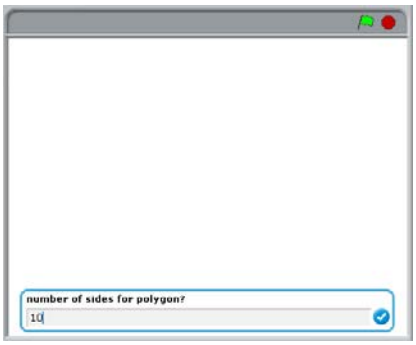


Getting User Input

- 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 two values are equal.

Reports true if first value is greater than second.

Reports true if both conditions are true.

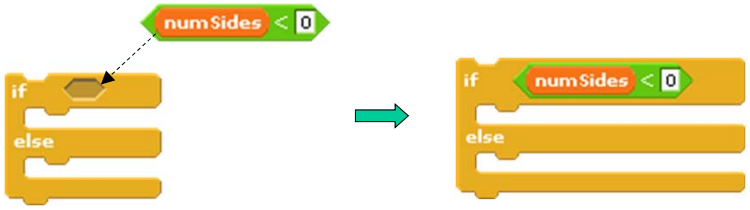
Reports true if either condition is true.

Reports true if condition is false; reports false if condition is true.

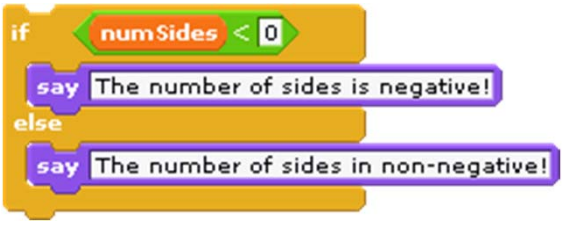
Flow of Control: Conditional Execution

- 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.

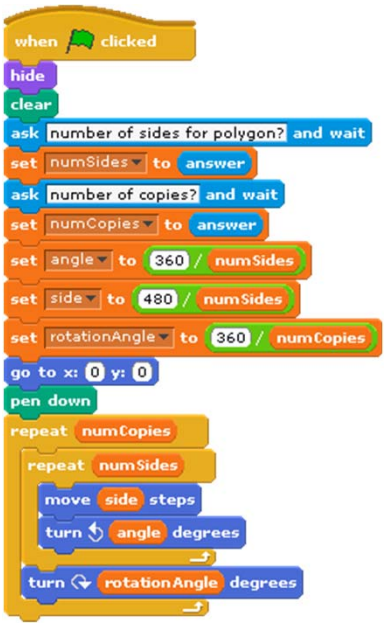


Flow of Control: Conditional Execution (cont.)



- If the condition is true:
  - the statements under the *if* are executed
  - the statements under the *else* are not executed
- If the condition is false:
  - the statements under the *if* are not executed
  - the statements under the *else* are executed

How can we deal with invalid user inputs?

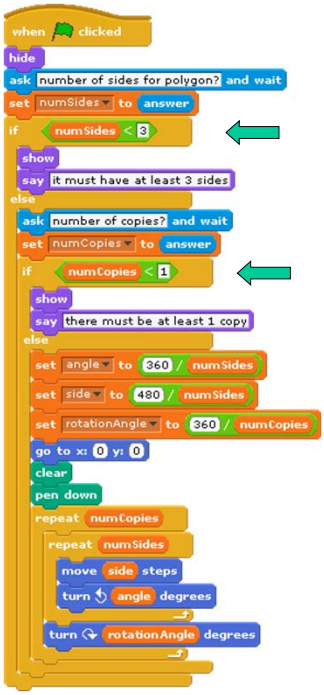


More Info on Scratch

- We're using the latest version:

  https://scratch.mit.edu/projects/editor

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



Final version

- We use two `if-else` statements to check for invalid inputs:
  - one checks for `numSides < 3`
  - one checks for `numCopies < 1`
- If an invalid input is found, we:
  - show the sprite
  - have the sprite say an error message
  - end the program
- Otherwise, we continue with the rest of the program.