UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 202
Introduction to Computation

Professor Andrea Arpaci-Dusseau
Fall 2010

Lecture 3: Telling the computer what to do

Exercise: How do you tell a computer what to do?

Groups of two:

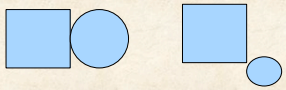
- Programmer
- Computer (Drawer)

Role of Programmer:

- Give instructions so "Computer" draws specified picture

Role of Computer (Drawer):

- Must follow instructions, but can do so in annoying way



What primitives are known?

Basic geometric shapes

- Line, circles, rectangles, octagons, hearts
- Not houses, not smiley faces, not trees

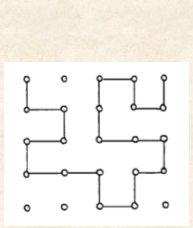
Numbers, sizes, and distances

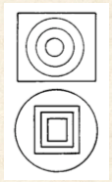
- Quantitative measurements (inches, cm)
- Qualitative measurements (bigger, smaller)

Coordinates and layout

- Up (above), down (below), top, bottom, left, right, vertical, horizontal, middle, half, divide, center…

Step 1: Create Secret Picture





Draw a picture

- You will tell others how to copy

Make sure no one else in room sees!

- Will switch partners

Pick something interesting, but relatively simple

Step 2: Follow Instructions with Partner

Version 1: No feedback
- Programmer cannot watch drawer
- Drawer/computer cannot communicate or ask questions back
- Drawer does not need to be cooperative but must follow directions (subject to interpretation)

Version 2: Visual feedback
- Programmer watches drawer and corrects mistakes
- Drawer cannot communicate or ask questions back

Discussion Questions

Why is English not good for "programming"?
- Other domains where English is not a good match?

How do different versions impact difficulty?

Which version corresponds to traditional computer programming?

Take-Away Lessons

Programs need set of basic primitives

Multiple programs (drawings, outputs) can be made from those same instructions

Must be precise: English is not always

Versions: Easier with more feedback

Traditional programming languages give no feedback until end
- Scratch (very visual) continuously gives feedback, should be easier!

Language for Exploring Algorithms

Need a programming language for
- Specifying algorithms
  - What exactly does it do?
- Comparing algorithms
  - Which one is faster?
- Executing algorithms
  - Have fun running it!

Options:
- English: Not precise enough and can't execute it!
- Traditional languages: Assembly, C, Java, …

Traditional Programming: C

```
void requestError(int fd, char *cause, char *errnum, char
*shortmsg, char *longmsg)
{
 char buf[MAXLINE], body[MAXBUF];

 printf("Request ERROR\n");
 /* Create the body of the error message */
 sprintf(body, "<html><title>CS537 Error</title>");
 sprintf(body, "%s<body bgcolor=""ffffff""\r\n", body);
 sprintf(body, "%s%s: %s\r\n", body, errnum, shortmsg);
 sprintf(body, "%s<p>%s: %s\r\n", body, longmsg, cause);
 sprintf(body, "%s<hr>CS537 Web Server\r\n", body);

 /* Write out the header information for this response */
 sprintf(buf, "HTTP/1.0 %s %s\r\n", errnum, shortmsg);
 Rio_writen(fd, buf, strlen(buf));
 printf("%s", buf);

 sprintf(buf, "Content-Type: text/html\r\n");
 Rio_writen(fd, buf, strlen(buf));
 printf("%s", buf);

 sprintf(buf, "Content-Length: %d\r\n\r\n", strlen(body));
 Rio_writen(fd, buf, strlen(buf));
 printf("%s", buf);

 /* Write out the content */
 Rio_writen(fd, body, strlen(body));
 printf("%s", body);
}
```

```
int requestParseURI(char *uri, char *filename, char *cgiargs)
{
 char *ptr;

 if (!strstr(uri, "cgi")) {
  /* Static content */
  strcpy(cgiargs, "");
  sprintf(filename, ".%s", uri);
  if (uri[strlen(uri)-1] == '/') {
   strcat(filename, "home.html");
  }
  return 1;
 } else {
  /* Dynamic content */
  ptr = index(uri, '?');
  if (ptr) {
   strcpy(cgiargs, ptr+1);
   *ptr = '\0';
  } else {
   strcpy(cgiargs, "");
  }
  sprintf(filename, ".%s", uri);
  return 0;
 }
}
```

Problems with Traditional Languages

High overhead to learning language

- Must get "syntax" just right
  - Keywords, semi-colon placement

Debugging can be frustrating

- Get wrong answer, must figure out why
- Program crashes, must figure out why

Sometimes hard to find motivating problems

- Results don't always look sophisticated

New Introductory Language: Scratch

Low overhead for learning

- Specifically designed for beginners
- No syntax errors (drag and drop building blocks)

Bugs in program not (usually) frustrating

- Bugs are visual, so entertaining
- See bugs right away when problem occurs (Exercise)

Lots of creative projects

- Games, interactive art, music

Simplifies transition to other languages

- Same basic control structures, concepts

Scratch Demo

Overview parts of environment

- Stage, Sprites, Blocks, Scripts, Costumes, Sounds

Different categories of blocks

- Motion, Looks, Sound, Pen, Control, Sensing, Operators, Variables

Example Project: Make walking cat

- Each sprite has own code and costumes
- Code within a script runs sequentially (multiple scripts can run concurrently)
- Activate script with "hat" block
- Different backgrounds, different Sprites

What essential features?

Computation: Perform calculations, work of algorithm
- Arithmetic and logical operations

Input/Output: Get data from user; Show result to user
- Input: Keyboard and mouse; Output: Display
- Scratch Limitations: Can't access disk or network

Control Structures: Repeat loops, if statements
- Run code only in some circumstances

Expressions: Query values and environment
- Ask questions: mouse clicked? Object touching edge?

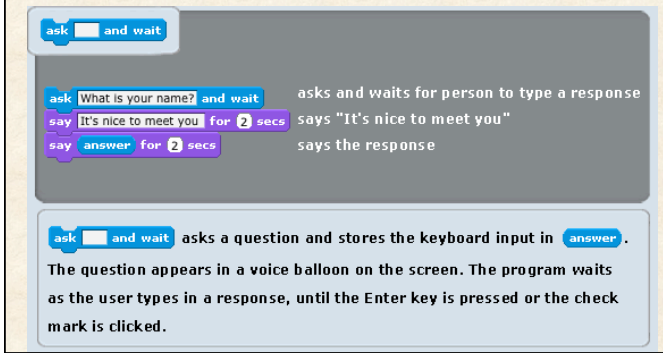
Variables: Remember data while computing over it
- Store numbers, strings, lists

1) Computation

Perform calculations, work of algorithm
- Arithmetic and logical operations
- Scratch: Operator blocks

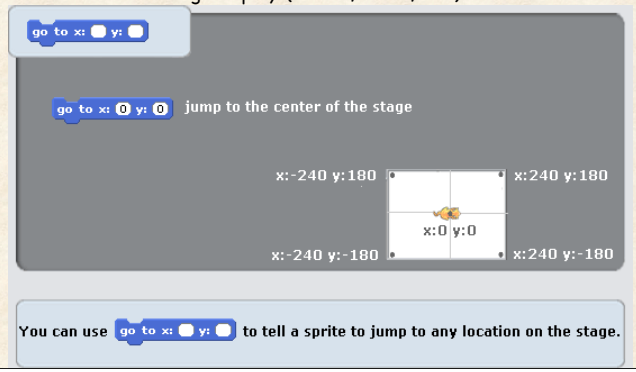
2) Input/Output

Input: Get data into computer
- Scratch: Sensing blocks: keyboard and mouse



2) Input/Output

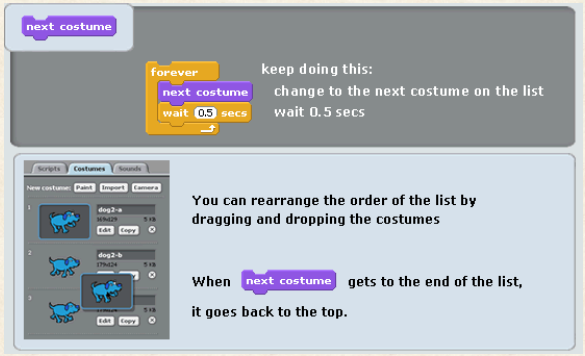
Output: Get data out of computer
- Scratch: Change display (Motion, Looks, Pen) and Sounds



2) Input/Output

Output: Get data out of computer

- Scratch: Change display (Motion, Looks, Pen) and Sounds



2) Input/Output

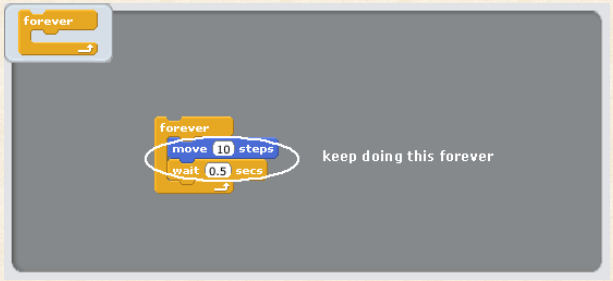
Output: Get data out of computer

- Scratch: Change display (Motion, Looks, Pen) and Sounds

3) Control Structures

Control Structures: Run code in non-sequential order

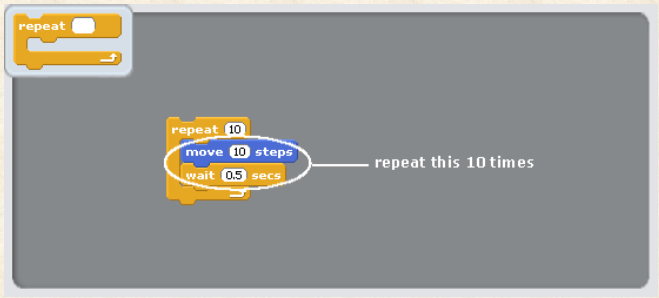
- Scratch: Control



3) Control Structures

Control Structures: Run code in non-sequential order

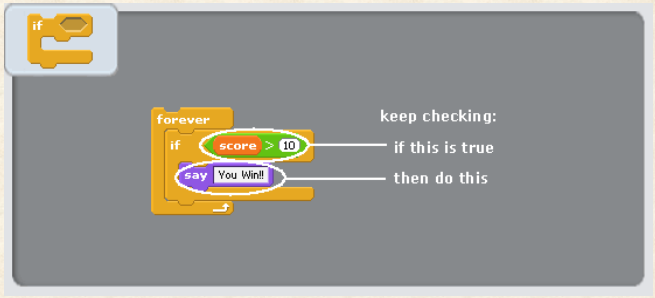
- Scratch: Control



3) Control Structures

Control Structures: Run code in non-sequential order

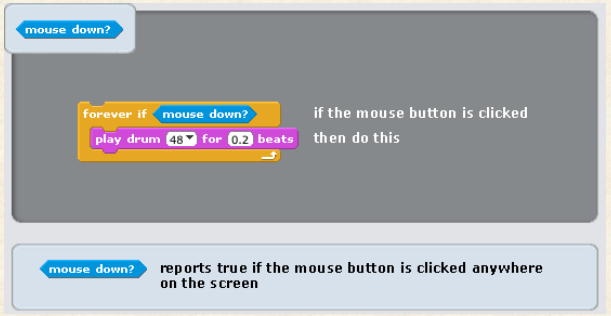
- Scratch: Control



4) Expressions

Expressions: Ask questions; Query values and environment

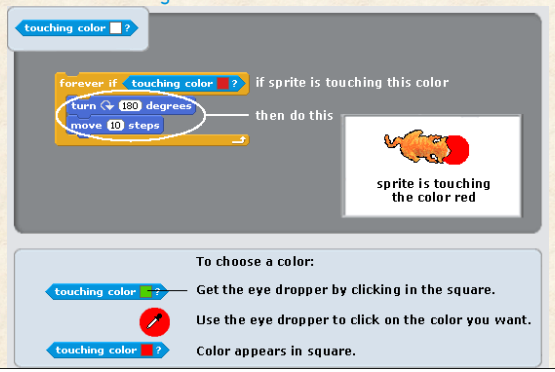
- Scratch: Sensing



4) Expressions

Expressions: Ask questions; Query values and environment

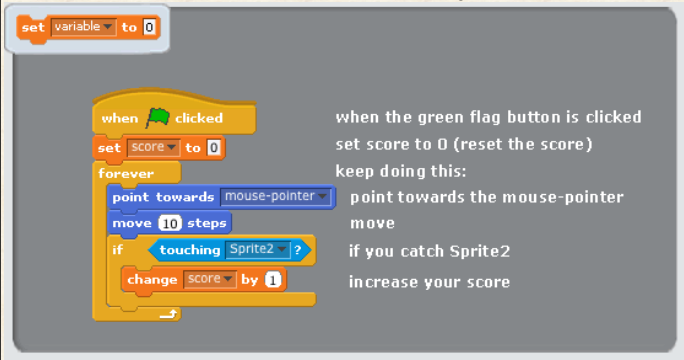
- Scratch: Sensing



5) Variables

Variables: Remember data while computing over it

- Scratch: Variables – Store numbers, strings, lists



Today's Overview

Today's Topics

- Motivation: English not precise enough for specifying algorithms
- Introduction to Scratch

Reading:

- Sections 2.1 and 2.2

Announcements

- Assignment 1 Due Today
  - Grades for weekly homeworks: 10 point scale
  - Use Learn@UW to check grades and comments (we'll announce)
- Download Scratch 1.4 as needed from http://scratch.mit.edu
  - Assignment 2 available, due next Friday; Easier after Monday's lecture