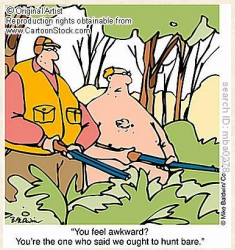
UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 202: Introduction to Computation — Professor Andrea Arpaci-Dusseau

How to tell computers what to do? (or, express algorithms?)



5 Minute Exercise: How to tell computer what to do?

Groups of two; take turns being:

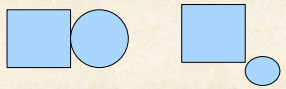
- Programmer
- Computer (Drawer)

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

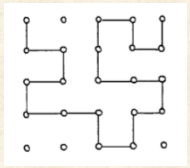
Numbers, sizes, and distances

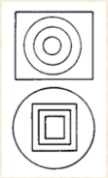
- Quantitative measurements (inches, cm)
- Qualitative measurements (bigger, smaller)

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 someone how to copy

Make sure partner does not see

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)

SWITCH ROLES:

- Person who was programmer is now drawer (and vice versa)

Version 2: Visual feedback

- Programmer watches drawer and corrects mistakes
- Drawer cannot communicate or ask questions back

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

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, stories, animation

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

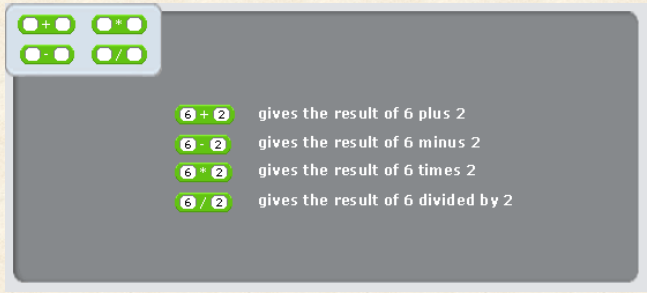
1) Computation

How would you use Scratch to calculate:
512 * 3019?

1) Computation

Perform calculations, work of algorithm

- Arithmetic and logical operations
- Scratch: Operator blocks



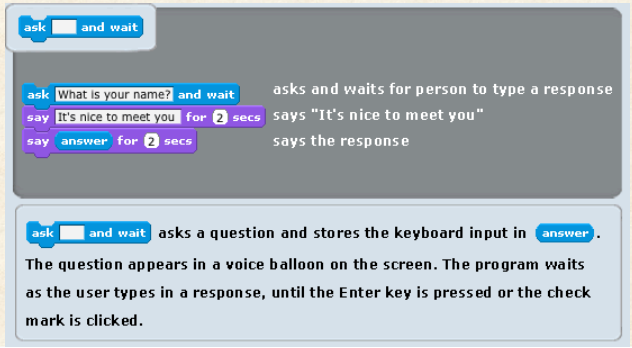
2) Input and Output

How would you ask the user their name and then say hello back to that person?

2) Input/Output

Input: Get data into computer

- Scratch: Sensing blocks: keyboard and mouse



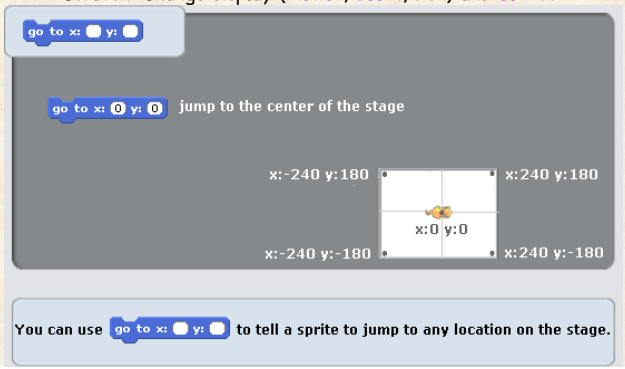
3) Input/Output

How would you tell a Sprite to move to the center of the Stage?

3) Input/Output

Output: Get data out of computer

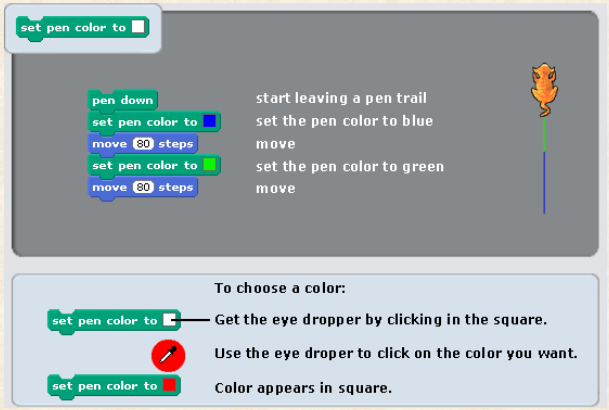
- Scratch: Change display (Motion, Looks, Pen) and Sounds



4) Input/Output

How to draw blue line for 80 steps, then green line for 80 steps?

4) Input/Output



5) Input/Output

How would you make a Sprite laugh when "space" is pressed?

5) Input/Output

Output: Get data out of computer

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

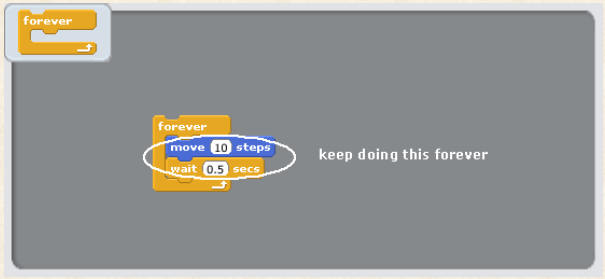
6) Control Structures

How would you get a Sprite to forever move around the Stage?

6) Control Structures

Control Structures: Run code in non-sequential order

- Scratch: Control



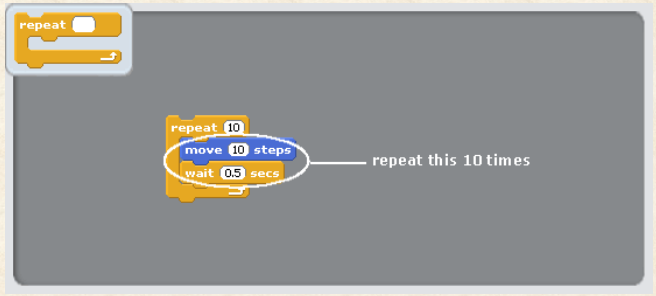
7) Control Structures

How would you get a Sprite to move 10 steps and pause – a total of ten times?

7) Control Structures

Control Structures: Run code in non-sequential order

- Scratch: Control



8) Expressions

How can you make a Sprite play a drum sound whenever the user presses the mouse button down?

8) Expressions

Expressions: Ask questions; Query values and environment

- Scratch: Sensing

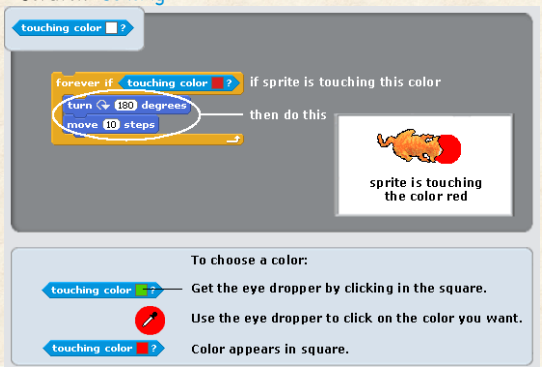
9) Expressions

Can you make a Sprite move and turn whenever it hits a red spot on the Stage?

9) Expressions

Expressions: Ask questions; Query values and environment

- Scratch: Sensing



Learning Scratch is Useful

Many of the lessons from Scratch are directly applicable to other more traditional programming languages

What features are essential to most programming languages?

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 and sound
- Scratch Limitations: Can't access file system or network

Control Structures: Repeat loops, if statements

- Default: Run instructions in sequential order
- Control allows code to run 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
- Later lecture!

Today's Checkup

In Scratch, what is the difference between a block and a script?

What is the difference between a Sprite and a costume?

Under what category would you find the following blocks: "move x steps"? "next costume"? "repeat"?

How many total steps will a Sprite running this script move?

Announcements

Assignment 2 Due Friday at 5pm

- Download Scratch 1.4 from http://scratch.mit.edu

When HW 1 is graded, will post to Learn@UW

- Will send mail to classlist

Lab hours change

Day	Time	Instructor or TA	Room
Mon	11:00 - 12:00	Prof. A Arpaci-Dusseau	CS 7375 (office)
Tue	12:30 - 2:25	Thea Hinkle	CS 1370 (lab)
Wed	11:00 - 12:00	Prof. A Arpaci-Dusseau	CS 7375 (office)
Wed	12:00 - 2:00	Victor Bittorf	CS 1370 (lab)
Thurs	4:15 - 6:15	Thea Hinkle	CS 1370 (lab)
Friday	11:00 - 1:00	Victor Bittorf	CS 1370 (lab)