



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:

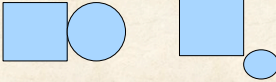
- Programmer
- Computer (Drawer)

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

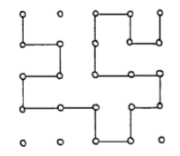
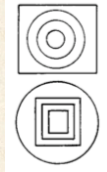
Numbers, sizes, and distances

- 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];

    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%s", uri);
            return 0;
        }
    }

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

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

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

    sprintf(buf, "Content-Length: %d\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);
}

```

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**

$6 + 2$ gives the result of 6 plus 2
 $6 - 2$ gives the result of 6 minus 2
 $6 * 2$ gives the result of 6 times 2
 $6 / 2$ gives the result of 6 divided by 2

2) Input/Output

Input: Get data into computer

- Scratch: **Sensing blocks:** keyboard and mouse

ask [] and wait asks and waits for person to type a response
say [It's nice to meet you] for [2] secs says "It's nice to meet you"
say [answer] for [2] secs says the response

ask [] and wait asks a question and stores the keyboard input in **answer**.
 The question appears in a voice balloon on the screen. The program waits as the user types in a response, until the Enter key is pressed or the check mark is clicked.

2) Input/Output

Output: Get data out of computer

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

go to x: 0 y: 0 jump to the center of the stage

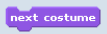
x:-240 y:180 x:240 y:180
 x:0 y:0
 x:-240 y:-180 x:240 y:-180

You can use **go to x: [] y: []** to tell a sprite to jump to any location on the stage.

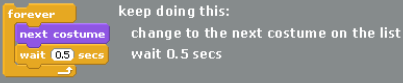
2) Input/Output

Output: Get data out of computer


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



keep doing this:



change to the next costume on the list
wait 0.5 secs



You can rearrange the order of the list by dragging and dropping the costumes

When **next costume** gets to the end of the list, it goes back to the top.

2) Input/Output

Output: Get data out of computer

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

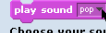



whenever space key is pressed
play this sound



If your computer has a microphone, you can record your own sounds.

Go to Sounds and click Record.




Use this to record. Then click OK.
Choose your sound from the menu.

3) Control Structures

Control Structures: Run code in non-sequential order

- Scratch: Control


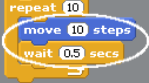



keep doing this forever

3) Control Structures

Control Structures: Run code in non-sequential order

- Scratch: Control

repeat this 10 times

3) Control Structures

Control Structures: Run code in non-sequential order

- Scratch: **Control**

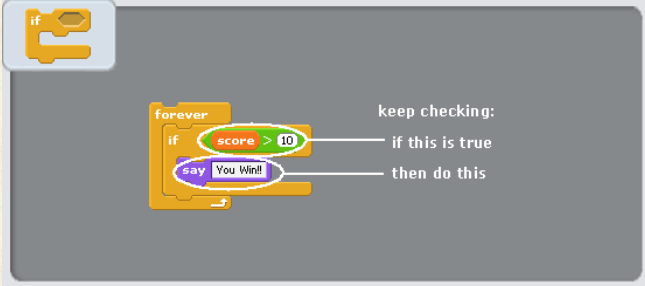


Diagram illustrating a control structure:

- if** block
- forever** loop containing:
 - if** block: *score > 10* (keep checking: if this is true)
 - say You Win!** block (then do this)

4) Expressions

Expressions: Ask questions; Query values and environment

- Scratch: **Sensing**

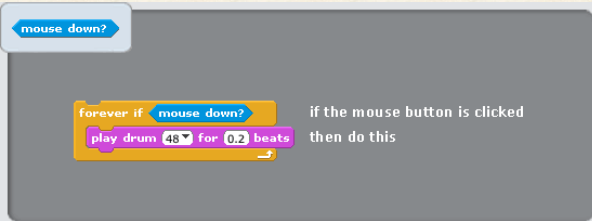


Diagram illustrating an expression:

- mouse down?** block
- forever if** block: *mouse down?* (if the mouse button is clicked) then do this: **play drum 48 for 0.2 beats**

mouse down? reports true if the mouse button is clicked anywhere on the screen

4) Expressions

Expressions: Ask questions; Query values and environment

- Scratch: **Sensing**

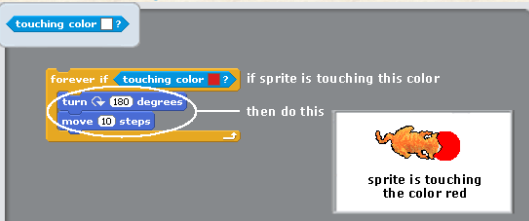


Diagram illustrating an expression:

- touching color?** block
- forever if** block: *touching color?* (if sprite is touching this color) then do this: **turn 180 degrees**, **move 10 steps**

sprite is touching the color red

To choose a color:

- touching color** (with eye dropper icon): Get the eye dropper by clicking in the square.
- touching color** (with red dot icon): Use the eye dropper to click on the color you want.
- touching color** (with red dot icon): Color appears in square.

5) Variables

Variables: Remember data while computing over it

- Scratch: **Variables** - Store numbers, strings, lists




Diagram illustrating variables:

- set** block: *variable* to 0
- when green flag clicked** block
- forever** loop:
 - point towards** block: *mouse-pointer* (point towards the mouse-pointer)
 - move** block: 10 steps (move)
 - if** block: *touching Sprite2?* (if you catch Sprite2)
 - change** block: *score* by 1 (increase your score)

when the green flag button is clicked set score to 0 (reset the score)

keep doing this:

- point towards the mouse-pointer
- move
- if you catch Sprite2
- increase your score

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