Computer Science 50: Intro to Computer Science I — Scribe Notes
Harvard College — Week 0: Wednesday
Fall 2007 — Anjuli Kannan

What Is Programming? (6:00-8:00)

- Ultimately this course is about solving problems and thinking methodically.
- But we will learn programming as a tool to do this.
- What is programming? Telling a computer what to do.
- No matter what language we write in, we are simply communicating instructions to the computer
- We give instructions in the form of an *algorithm*, or procedure.

Example 1: Looking in a Phone Book (8:00-15:00)

- Suppose we want to find the phone number of a paint store in a 1500-page phone book
- Naïve way: begin at front of phone book and begin flipping pages until we get to the paint store section in the yellow pages.
- It turns out the paint store section is on page 1011. Therefore, this method would require 1011 steps to complete.
- Can we do better?
- Well, we know that the P section is going to be about three-quarters of the way through the yellow pages, so maybe we can estimate where the P section is, and then begin flipping.
- But we can't simply tell computers to "estimate", so we need a more precise way of defining this algorithm.
- Better way: Open phone book to center and rip in half. We know that the desired page is in the second half, so throw the first half out. Open the remaining chunk to the middle and rip in half. Now, we know that the desired page is in the first half, so throw the second half out. Continue to repeat this process, each time retaining only the necessary half.
- As long we check correctly to make sure we keep the correct half, we must eventually arrive at the desired page
- In each round of this algorithm, we cut the size of the phone book in half. So if we start with 1024 pages, we will be down to a single page in 10 rounds. This algorithm takes just 10 steps. 10 steps!!
- Compare the two algorithms: 1000 steps vs. 10 steps. The second algorithm is much more *efficient.*
- Now suppose we had a million pages. Then the first algorithm would take a million steps, while the other would just take 22!
- Now suppose we had a billion pages. Then the first algorithm would take a billion steps, while the other would take just 32!
- Clearly, our second algorithm has huge payoffs in efficiency. In this class, we will not just solve problems, but strive to solve them *well*, i.e., efficiently.

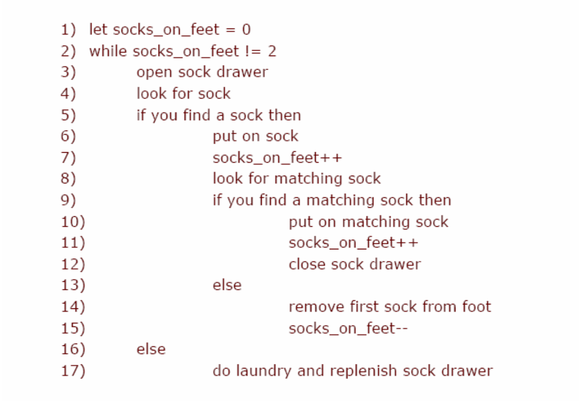
Example 2: Counting Students (15:00-22:00)

- Suppose we wish to determine the number of students in the lecture hall.
- Naïve way: Have one person stand in front of the lecture hall and point to each student once, saying "1, 2, 3, …" until each student has been counted.

- This will take as many steps as there are students, say, 250.
- Better algorithm (performed by all students):
    1. Stand up
    2. Assign yourself the number 1
    3. Find someone that is standing up.
    4. Add your number to that person's number. The total is your new number.
    5. One of you sit down.
    6. If you are still standing, go back to step 3.
    7. Algorithm finishes when there is one person standing. His number is the number of students in the class.
- How many steps does this take? Each "round" cuts the number of students still standing in half. So if there are about 250 students, it will require 8 rounds.
- Again, we have a much more efficient and elegant algorithm.

Example 3: Putting Your Socks on in the Morning (22:00-26:30)

- Here is an algorithm for putting your socks on in the morning written in "pseudocode."



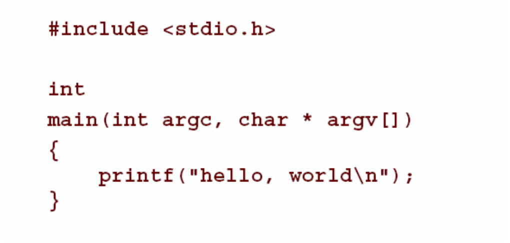
```
1)  let socks_on_feet = 0
2)  while socks_on_feet != 2
3)          open sock drawer
4)          look for sock
5)          if you find a sock then
6)                      put on sock
7)                      socks_on_feet++
8)                      look for matching sock
9)                      if you find a matching sock then
10)                                 put on matching sock
11)                                 socks_on_feet++
12)                                 close sock drawer
13)                     else
14)                                 remove first sock from foot
15)                                 socks_on_feet--
16)         else
17)                     do laundry and replenish sock drawer
```

- Pseudocode is in language that humans can understand, but written methodically so that it can easily be translated into real code.
- In step 1, we set a variable socks_on_feet to the number 0.
- In step 2, we enter a loop that will be repeated as long as the condition (socks_on_feet != 2) is true.
- Inside the loop, we open the sock drawer, look for a sock, and then enter a condition:
    - If a sock is found, put it on.
    - Otherwise, do laundry and replenish sock drawer.
- This code has a bug. If only one sock is in the sock drawer, it will remain in the while loop forever.
- This is called an infinite loop. As you will see later, sometimes programs we write will contain infinite loops. We will recognize these cases because when we run the code, the computer will "hang"—it will keep running and produce no output because it is stuck in a loop.

- It might seem obvious to us what to do in the case where there is only one sock in the drawer, but a computer cannot make assumptions the way we can.
- Therefore, our algorithms must be extremely *specific* and *precise* and account for all possible cases. Nothing can be assumed.

Programming and Languages (26:30-30:00)

- As you know, the computer understands only ones and zeros.
- Back in the day, people had to communicate with the computer using switches or punch cards to indicate ones and zeros.
- Today, we can use languages like C to abstract away some of the really low-level control.
- We write code in C, and then use a compiler to translate it into ones and zeros that the computer can understand.
- However, even C can still be pretty unwieldy.
- For instance, what does this code sample do?



```
#include <stdio.h>

int
main(int argc, char * argv[])
{
    printf("hello, world\n");
}
```

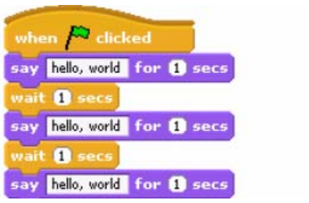
- It prints the phrase "hello, world!" on the screen.
- This code excerpt is pretty straightforward, but you can see there is a lot of syntax involved in accomplishing a fairly simple task.
- This syntax is necessary for the computer to understand our instructions, but can make it difficult for us to produce interesting, useful programs.
- In order to focus on programming concepts without getting bogged down in syntax, we will go even higher level than C and start with a programming environment called Scratch.
- Scratch abstracts away much of the detailed and confusing syntax and makes it very easy to make large, interactive programs very early on.

Introduction to Scratch (30:00-34:30)

- You can download Scratch from the website.
- It is a program written by a group MIT in order to teach children.
- On left, notice the puzzle pieces, which represent statements. Programs will be composed by putting puzzle pieces together in a particular order.
- At bottom right are sprites, or characters that will carry out your instructions.
- At top right is the stage, where the program will be carried out.
- To the left of the stage is the scripts area, where puzzle pieces must be dragged and strung together.

Basic Elements of a Program (34:30-44:00)

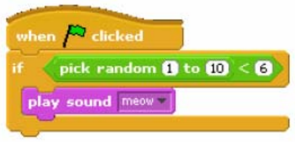
- Statements: single line instructions that tell sprite to do something. Example: say "Hello!"
- The following program (Hello2.sb) will say "Hello, world!" for 1 second, wait 1 second, say "Hello, world!" for 1 second, wait 1 second, say "Hello, world!" for 1 second:



```
when [flag] clicked
say hello, world for 1 secs
wait 1 secs
say hello, world for 1 secs
wait 1 secs
say hello, world for 1 secs
```

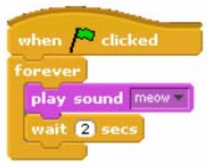
- But what if we wanted it to say "Hello, world!" ten times? We could add more statements: wait 1 sec, say "Hello, world!" for 1 sec, wait 1 sec, etc.
- We see this is a scalability issue:
    - Resource inefficient.
    - Tedious. Takes time to type out.
    - Makes it difficult to change what the cat is saying.
- So we don't just want to do things, we want to do them *elegantly* and *efficiently*.
- Remember, your time is a resource. You want to program in a way that will maximize use of your own time in writing and maintaining the program.

- Boolean expressions: things that are either true or false. Examples: sprite is touching mouse-pointer; a < b.
- We can use AND operator to compose two Boolean expressions into another Boolean expression. The resulting expression is true iff. the original two expressions are true.
- Conditions: instructions to execute a piece of code only if a particular Boolean expression is true.
- The following program (Hello5.sb) picks a number randomly between 1 and 10 and meows only if the number is less than 6 (i.e., half the time):



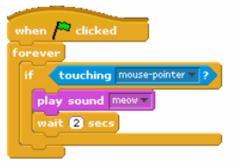
```
when [flag] clicked
if < pick random 1 to 10 < 6 >
    play sound meow
```

- Loops: repeatedly execute some piece of code, either forever or a specified number of times.
- The following program (Hello6.sb) makes use of a loop to make the cat meow every 2 seconds. By using a loop, we don't have to write out instructions repeatedly:



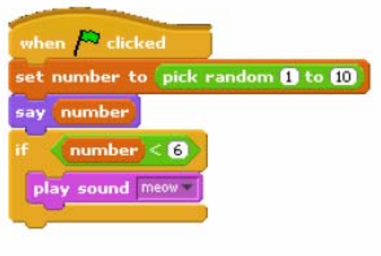
```
when [flag] clicked
forever
    play sound meow
    wait 2 secs
```

- In Hello7.sb, we combine a loop and a condition so that the cat only meows if we are "petting it":



```
when [flag] clicked
forever
    if touching mouse-pointer ?
        play sound meow
        wait 2 secs
```

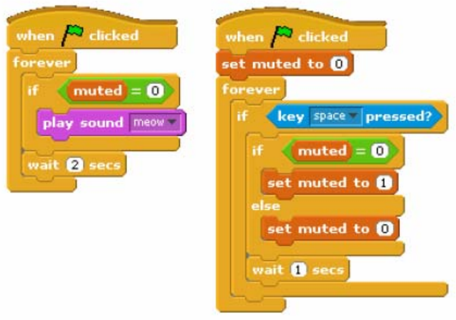
- Variables allow you to store a value and retrieve or change it later
- Hello9.sb is like Hello5.sb, but will store the random number in a variable, and then tell us its value:



```
when [flag] clicked
set number to pick random 1 to 10
say number
if number < 6
    play sound meow
```

More Complex Programs: Threads (44:00-50:00)

- In move1.sb, we can change the number of steps the cat moves to make it speed up while the program is running.
- This shows us that the programming environment is dynamic.
- In Move2.sb, we have two sprites: a cat and a bird. It appears their scripts are being executed simultaneously.
- In the same way, our computers can may seem to be doing multiple things (running several programs, for instance) at once. Actually, it is just an illusion; computers really just run each program for a few milliseconds at a time before switching to the next one. But they appear to be doing multiple things at once.
- In Scratch, we can have two scripts executing simultaneously.
- In Hello10.sb, one script handles repeated meowing, while a second handles our ability to mute the meowing:



```
when [flag] clicked
forever
    if muted = 0
        play sound meow
    wait 2 secs
```

```
when [flag] clicked
set muted to 0
forever
    if key space pressed?
        if muted = 0
            set muted to 1
        else
            set muted to 0
        wait 1 secs
```