Programming

This is CS50. Harvard University. Fall 2015.

Cheng Gong

Table of Contents

Questions

- Last time we talked about binary, and one question on last night's assignment asked:

```
01000001 01101110 01110011 01110111 01100101 01110010
00100000 01101001 01110011 00100000 00110100 00110010
00101110
```

- Hm, we can translate these to numbers:

```
65 110 115 119 101 114 105 115 52 50 46
```

- But we can also decode this message further with a standard mapping of numbers to letters, called **ASCII**[1].
- This website[2] shows you the mappings of all 256 values that an 8-bit piece of data can be. For example, the letter `A` is the decimal number 65, `B` 66, `C` 67, and so on. And lowercase letters are another set of numbers. Numbers, too, that you type, would be represented as different numbers according to the table.
- Since ASCII only specified characters for most of the ones we need in English but not other languages,another standard called Unicode includes mappings for more characters.
- In many programming languages, we actually need to also specific if a set of bits should be interpreted as raw numbers, or as characters under some standard.

[1] http://en.wikipedia.org/wiki/ASCII
[2] http://www.asciitable.com/

- Colors, too, are represented by numbers in computers. RGB is the standard most computers use, which indicates how much red, green, and blue are mixed together. For example, `72` `73` `33` for red, green, and blue will be a brownish color:

- "24-bit color" on a computer, then, means that 24 bits total are being used to represent colors, with 8 bits per each color.
- In any case, the message of the question translated to `Answeris42.`
- And to recap the issue with Ariane 5's launch, integer overflow led to a chain of events that ended in a tragic crash.
- In the case of the Patriot defense system, imprecision in calculating missile speed also led to significant failure.
- Now David will make a peanut butter and jelly sandwich based on a random submission from last night:
  # Remove two pieces of bread from the loaf.
  # Lay each piece flat on the plate.
  # Take the knife and put it in the peanut butter jar.
    # There's no mention of removing the lid from the jar, so David sticks the knife into the jar with the lid on.
  # Scoop out a tablespoon of peanut butter.
    # David uses his hand to remove some peanut butter.
  # Spread the peanut butter fully on one piece of bread.
  # Take the knife and put it in the jelly jar.
  # Scoop out a tablespoon of jelly with the knife.
  # Spread the jelly fully on the other piece of bread (without peanut butter).
  # Pick up the piece of bread with jelly and place it face down on top of the bread with peanut butter.

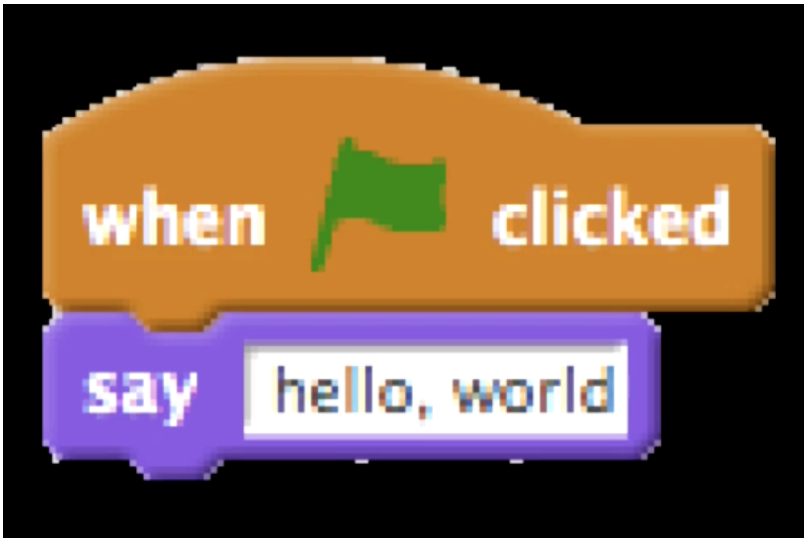
- We'll try this one more time:

```
# 1. Grab loaf of bread
# 2. Take two slices of bread (assuming pre-sliced)
# 3. Name these two slices Slice 1 and Slice 2
# 4. Put slices of bread onto plate
# 5. Put away rest of bread loaf
# 6. Grab peanut butter jar
# 7. Open peanut butter jar
# 8. Grab knife with right hand
# 9. Take dollup from jar using right hand
    # Uh oh, here it asks us to use our hand!
# 10. Pick up slice 1 from bottom with left hand
# 11. Apply substance from knife in right hand to top of slice 1 in left hand
# 12. Spread evenly
# 13. Put down knife
# 14. Put slice 1 back onto plate with substance face up
# 15. Close peanut butter jar
# 16. Put away peanut butter
# 17. Grab jelly jar
# 18. Open jelly jar
# 19. Repeat steps 8-14
# 20. Close jelly jar
# 21. put away jelly jar
# 22. Pick up slice 2
# 23. Place slice 2 on slice 1
# 24. Align sides of slice 1 and slice 2.
```

- Even though this "program" is many steps longer and seemingly more precise, there are still many opportunities for misinterpretation.
- Legal contracts especially are long in order to try to avoid ambiguity.
- We'll have seminars[3] that cover specific technical topics and office hours[4] for one-on-one discussions about questions, with our team of teaching fellows.

Scratch

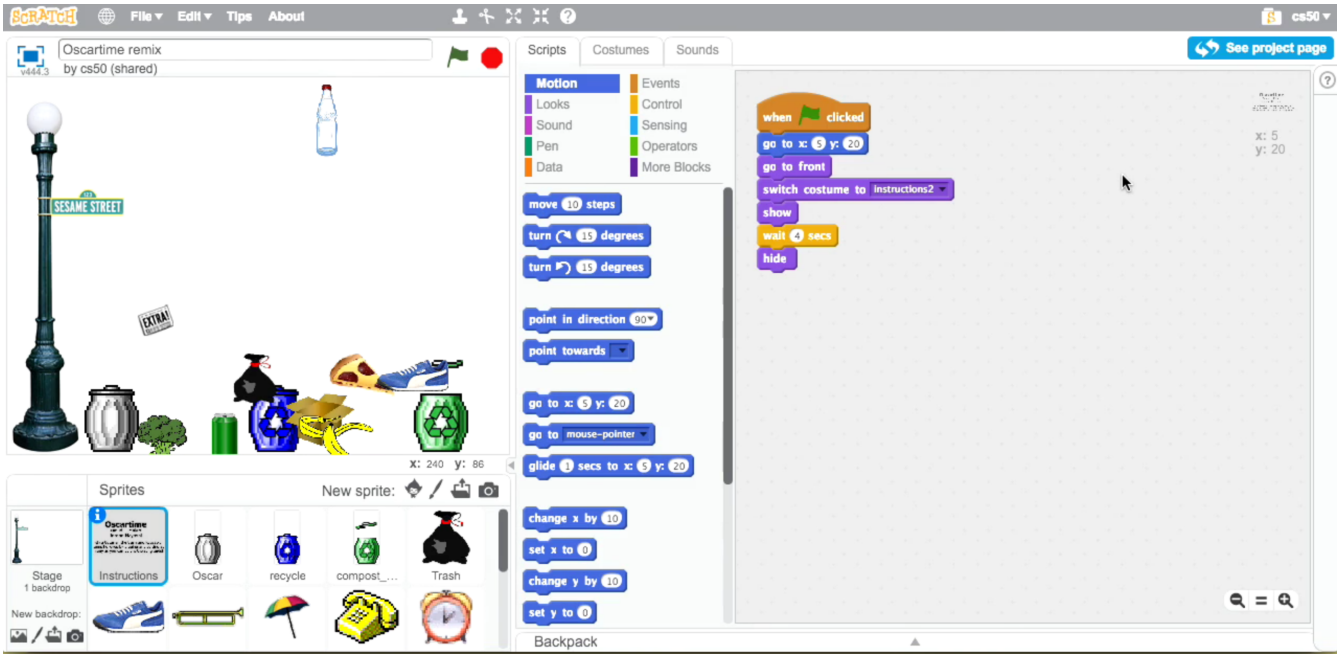
- Scratch[5] is a simple graphical programming language, and perhaps the simplest program for it looks like this:




```
# We can read this quite literally, as in "when green flag clicked, say hello, world".
```

- Learning Scratch in itself might not be too useful, but we will be able to talk about some programming ideas without having to worry too much about the smaller details.
- We take a look at Oscartime remix[6] again:

[3] https://cs50.harvard.edu/mba/seminars/
[4] https://cs50.harvard.edu/mba/hours/
[5] http://scratch.mit.edu
[6] https://scratch.mit.edu/projects/76196420/

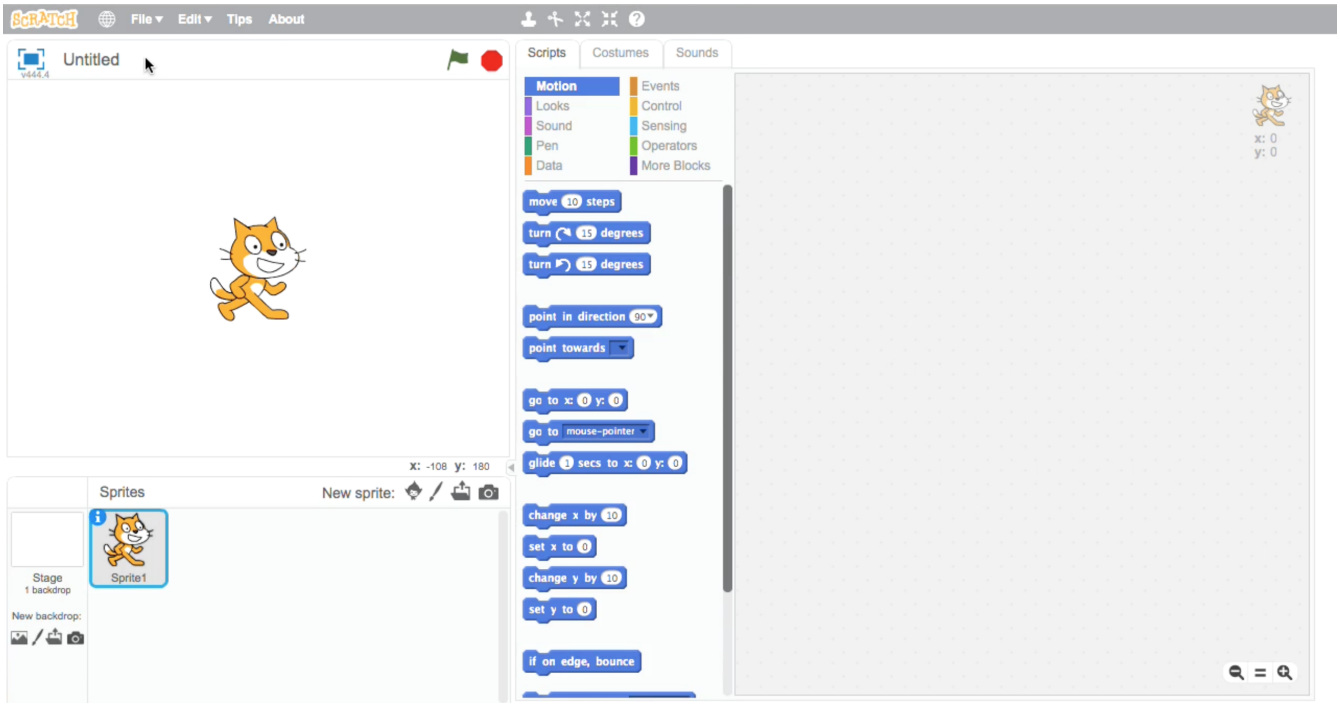


The top left area is the stage, where everything happens.

The bottom left area shows the sprites, or characters that can each be doing different things.

We try to play this again, and we notice that some items disappear when they are at the right trash can, and remain visible if they are at the wrong one, hinting that there's a condition somewhere that checks this for each item.

- David will start with a new project:

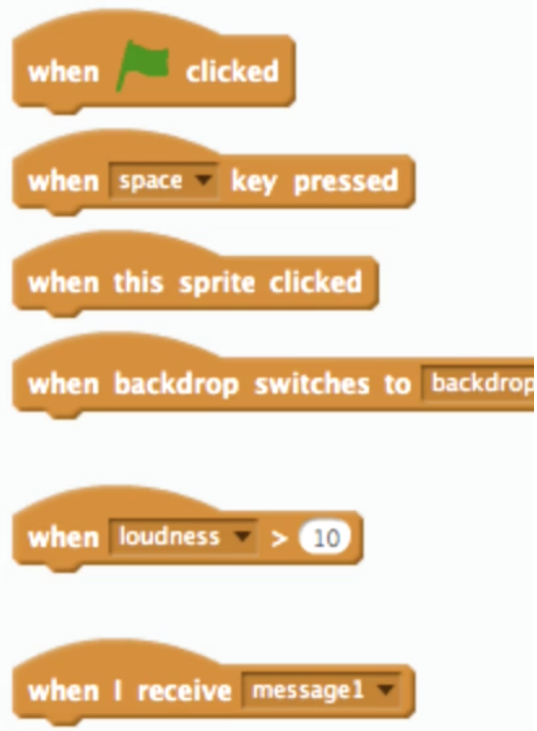


Notice that the middle has different categories for puzzle pieces, or blocks, that we can drag to the right to control the behavior of our cat.

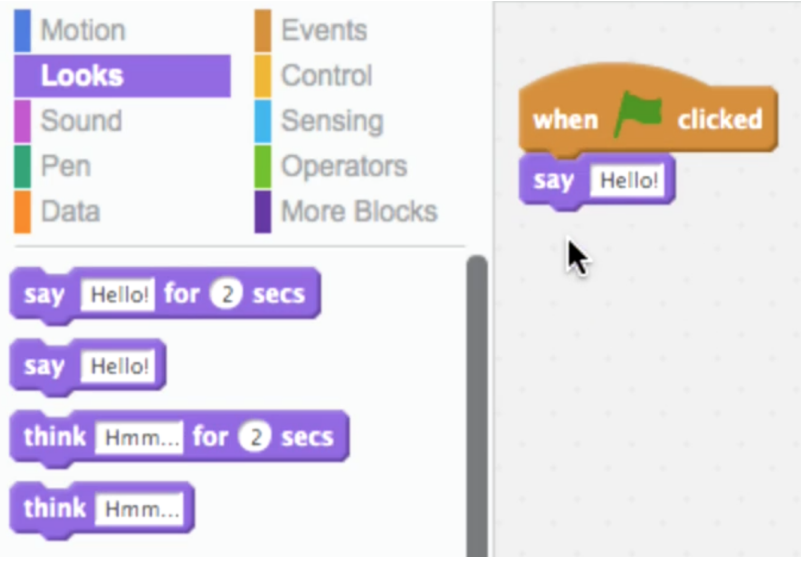
- Many of the pieces are visually self-explanatory. For example, the shape of conditions imply that we can place other pieces inside of them:

It seems like we can only fit two branches in our code, but actually adding more `[ if < > ] else]` blocks inside will cause the outside ones to grow, so we can have many branches.

- There are also events, where something happens and the program should respond. For example, these blocks start statements that are under them when their events occur:



- For example, we can drag these two pieces together:




```
# And what we expect to happen does, when we click the green flag, the cat says "Hello!"
```

- Let’s talk about a few topics to help us make our programs:
  - # functions
    - # groups of statements that do things
  - # conditions
  - # Boolean expressions
    - # a question that has a true or false answer that we use in conditions
  - # loops
  - # variables
    - # like letters in Algebra that are placeholders for other values
  - # events
  - # threads
    - # allows programs to do multiple things at once
- Let’s try this:

  - # Hmm, it sounds like a constant chirp. That’s because we’re playing the "meow" sound constantly, without waiting for the sound to complete before starting it again.
- We can remove the purple block from inside the loop and put it under the green flag block:

But now we have to click the green flag over and over again if we want to hear the "meow" more than once.

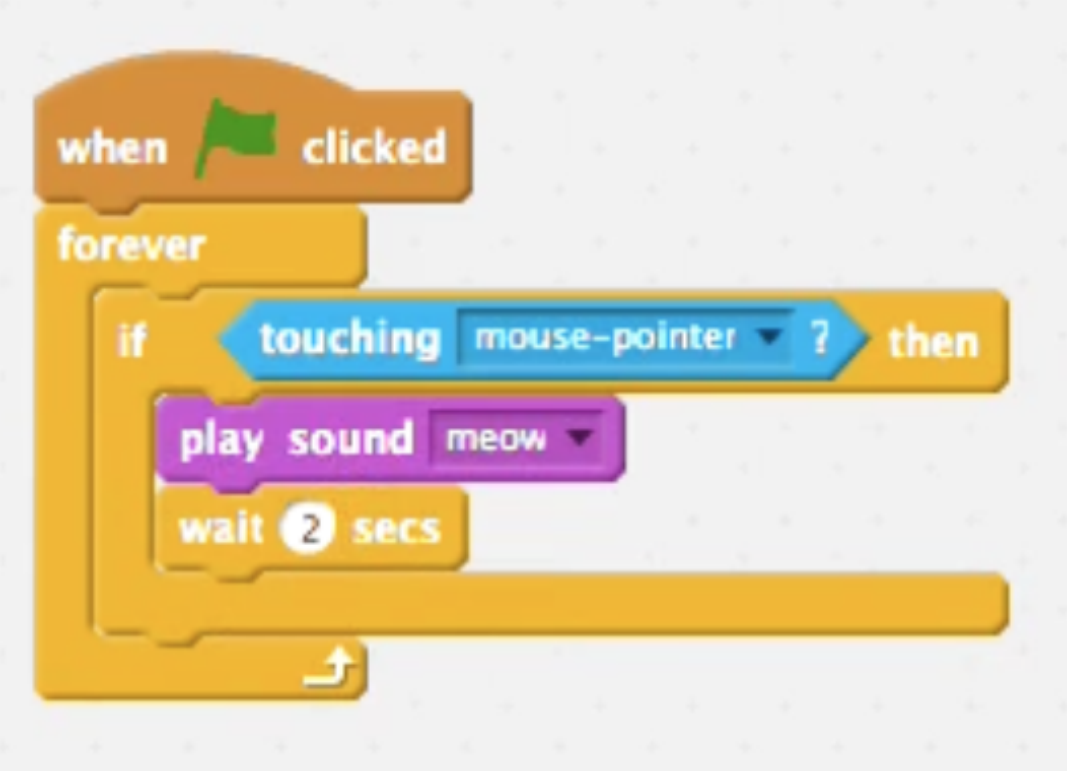
- We can solve this problem by having the program wait between playing the sound:

- We'll build this program with other pieces:
image::scratch_meow4.png[alt="Scratch meow", width=400]

Logically, it seems like we'll only hear the "meow" sound now if a random number chose is less than 6.

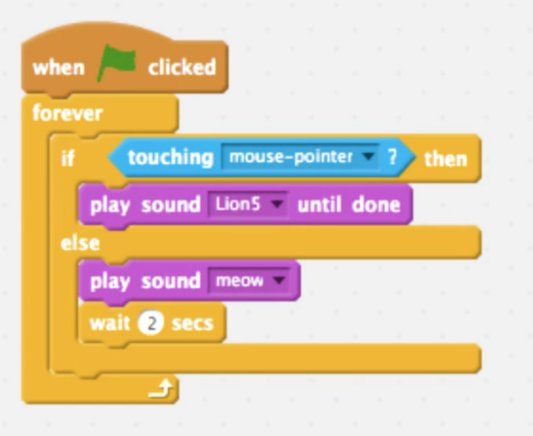
- In theory, if we did this over and over again, we should hear the sound 50% of the time. But computers aren't truly random, since they can only do what they are instructed to. There's probably some code that tells it how to "think" of a random number, which is difficult to do, since we want randomness in our security and cryptography programs. Patterns in encrypting a message, for example, might make it weaker and easier to exploit.

- A physical device like a computer or phone can use other inputs, like the current time or the speed its accelerometer detects, to use as input for some algorithm that then produces some pseudorandom numbers.
- Now we'll build this program:

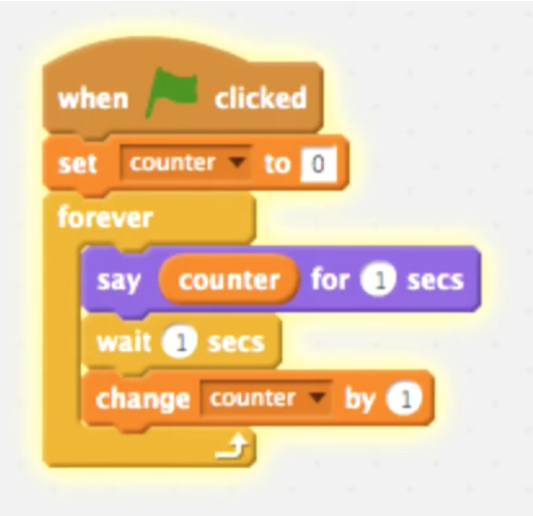



```
# And it does what we might expect logically, where our cat only says "meow" if our mouse pointer is touching it.
```

- By the way, many of these examples will be at http://cdn.cs50.net/2016/mba/classes/1/src1/, and this one is called "pet the cat"!
- Another example, "don't pet the cat":

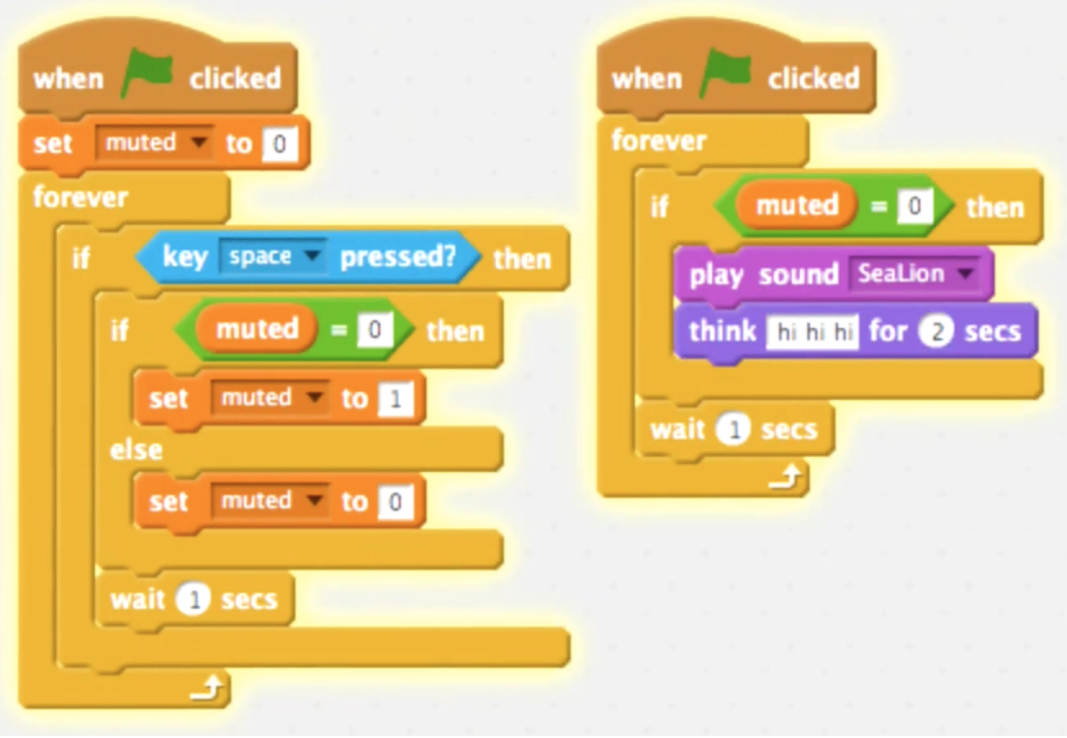


- "counting sheep" is interesting:



    # Here we have a variable called `counter` that we store a number into, and increase by 1 every time we say it. In programming we generally name variables with words that describe the value that they store, and we can use them to remember things, like a number in this case.
    # But this program probably has a bug eventually, where the counter overflows since the number becomes so big we can't fit it into the number of bits that our variable has.
    # We could start the counter at a big value, and start the program there and see what happens. We could even increase the value it increases by each time, to speed up the process.

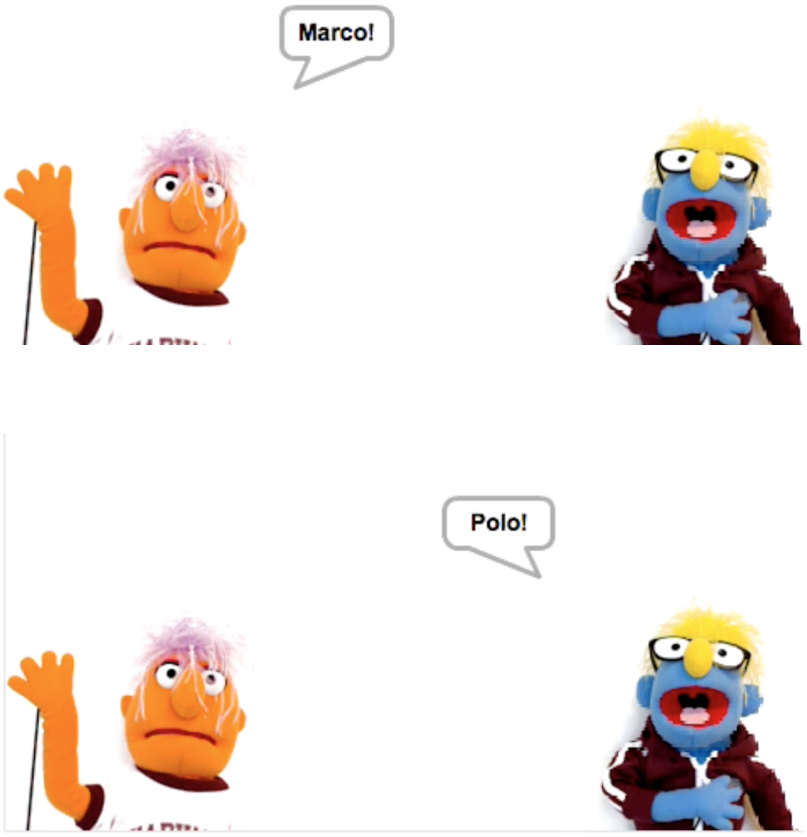
- There are ways to handle really big numbers by programming for them (for example, setting aside more bits for the number as we need them) but generally that doesn't happen automatically in many programming languages.
- The program "hi hi hi" has two sets of blocks running in parallel when the green flag is clicked:



    # The set on the left sets the `muted` variable to `0` or `1` each time we press space, and the set on the right plays a sound if `muted` has a value of `0`, since `0` is a

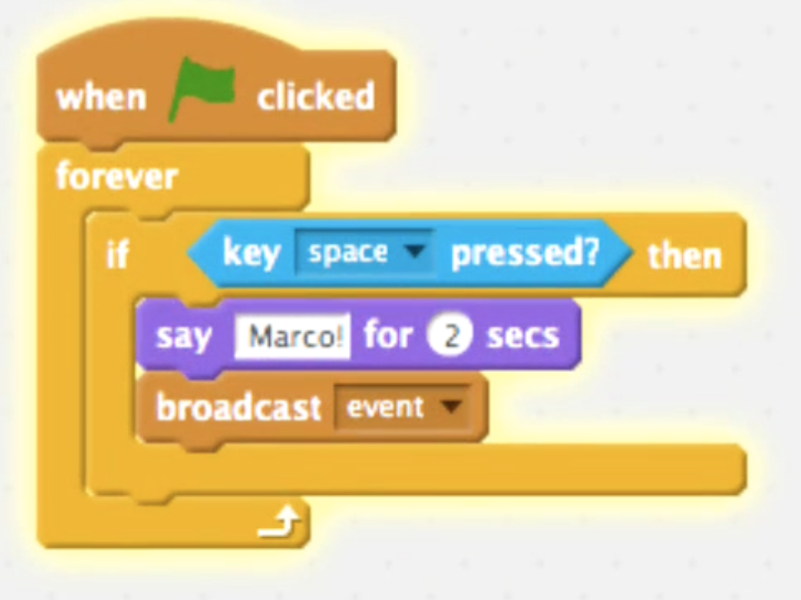
reasonable way to represent "off" or "false". And by convention, `1` represents "on" or "true".

- One core in a CPU can actually only do one thing at a time, but they are so fast these days that they are switching constantly between working on tasks so we have the illusion of them doing two or more things at once. And most CPUs these days have multiple cores, so we can actually do multiple things at once (which is faster). But the software needs to written to be multithreaded, or able to support splitting tasks into multiple cores.
- Let's look at the program "events":



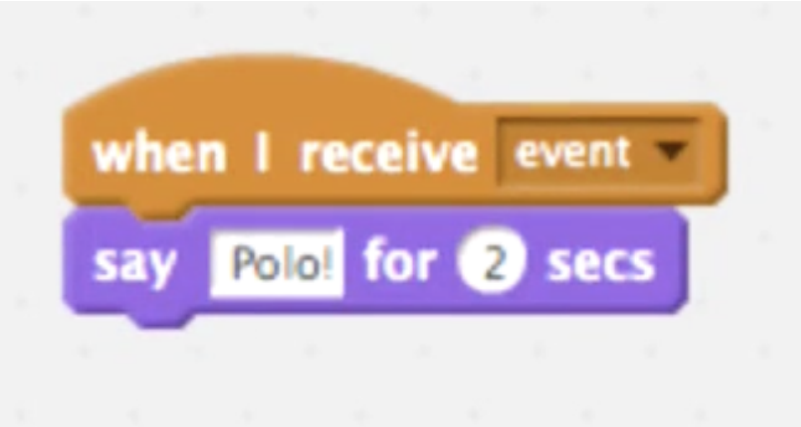
When we start our program and press space, the muppet on the left says "Marco!" and the one on the right says "Polo!" without us having to do anything else.

- The blocks for our orange muppet look like this:



Now we have this `broadcast` block that has this sprite "tell" every other sprite that `event` happened (and like variables, we can name this `event` anything we'd like). We can think of this as a digital message behind-the-scenes that other sprites can listen for.

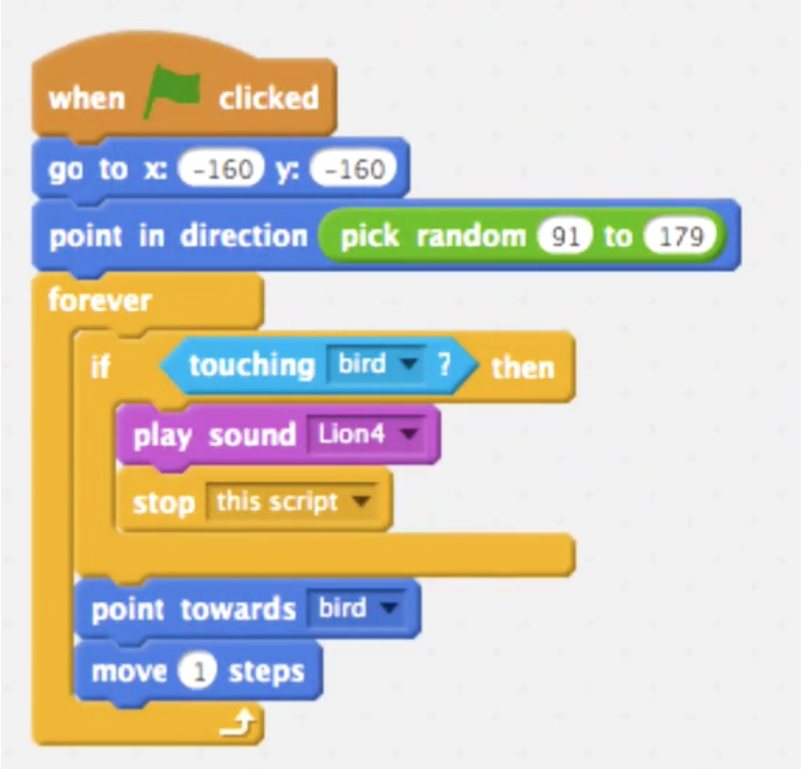
- The blue muppet has just this set of blocks:



When it receives the message that `event` occurs, then it says "Polo!"

- Now let's look at "threads":

The cat follows the bird as it moves around the screen. Here we see the logic for the bird, which instructs it to start at a certain location on the stage marked by coordinates, and move around until the cat touches it.



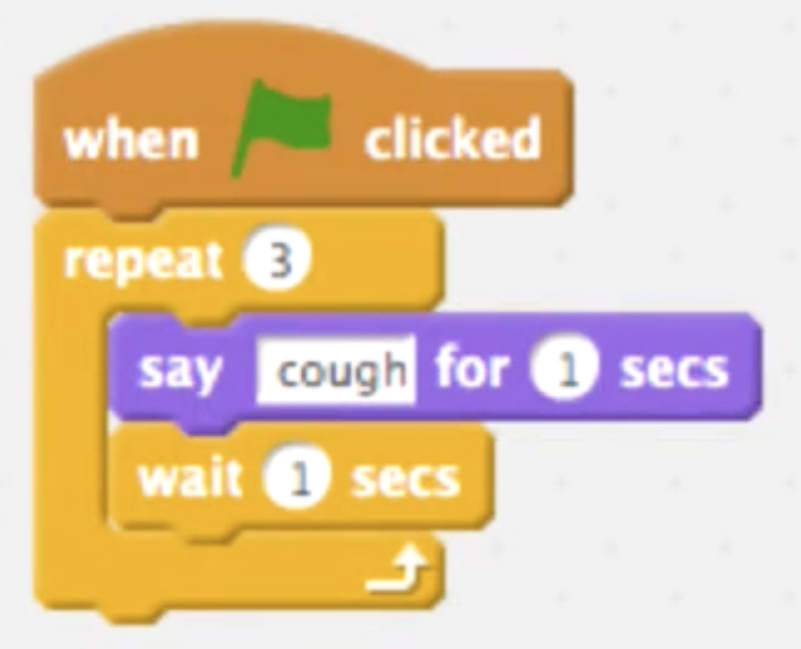
As for our cat, we start it at a location facing a random direction, and then point it towards the bird while making fewer steps, until it touches it.

If we changed that to `10000` steps, the cat jumps to being off the screen, and even though it passed over the bird, the movement happened at once so it didn't detect when it touched the bird.

- Now let's look at "cough-0":


```
# We notice that we're repeating a set of two blocks three times, which is a clue that
  the quality, or the design, of our code might be improved.
```

- We can see that "cough-1" is a natural improvement:




```
# Now if we want to change something we only have to change it in one place, like
  the number of seconds we wait between coughs.
```

- With "cough-2", we begin to see a little abstraction:

    # `cough` is now just a function, or a purple block that we can use like any other, that we've defined to be saying cough and waiting 1 second.

    # With functions, more complicated software can reuse pieces of functionality without having to rewrite them over and over again.

- We can add a feature where we tell our `cough` function how many times to actually cough, in "cough-3":


```
# When we create our own function blocks, we can specify that they take in some input. In this case, we call whatever input cough is given n, and use it in our loop so that we can cough any number of times.
```

- And our final version, "cough-4", is even fancier:


when clicked
cough 3 times
sneeze 3 times
define say word n times
repeat n
say word for 1 secs
wait 1 secs
define cough n times
say cough n times
define sneeze n times
say achoo 3 times

    # We've defined functions within functions, such that `cough` and `sneeze` both rely on the `say (word) (n) times` function.
    # And there's a bug in our program, whereby `sneeze` says `achoo` 3 times no matter what `n` is!

- We look at a few final demos, "gwalker" and "Ivy's Hardest Game remix", that are more complex but infer that events like arrow key presses help control movement and random numbers help place items on the screen in different places every time.
- And these demos were probably built one component at a time, perhaps with items falling first, then different items each time, then different items in random positions each time.
- So in Project 0, you'll come up with some interactive program of your own. And together we build a quick program that has our cat walk back and forth across the screen.
- Until next time!