Week 1; Lecture 1

Introduction to Programming

Class

- Lecture will introduce new concepts
- Tutorials will give you a chance to make sure you know the answers to the questions
- Labs will teach to to program
 - One learns programming only by programming
 - We can tell you things, but the words only have meaning when you program

Materials

- For this class you will need
- 1) Access to a computer
- 2) A pen drive

Acquire these materials as soon as possible.

Lectures and Tutorial

- Each week, you will get a list of questions, which are answered in the lecture.
- During tutorial, you will have the opportunity to check your answers
 - Turn in your answers at the end of the tutorial
 - Answer quality determines your tutorial score
- The objective section of the test will comprise a selection of the tutorial questions

Lab

- The lab is where you learn programming
- During the lab you will work on your program
 - You will probably have to work outside of class as well
 - A laptop will help you move programs to and from home.
 - The pen drive will let you store your programs in a safe place.
- The essay part of the tests will ask that you write portions of the code you write during the lab.

Programing

- A program is a description of behavior such as
 - A recipe
 - Instructions to assemble furniture
- A computer program is a description of a computer's behavior such as
 - Sending email
 - Executing a game
- The goals of this class are
 - To teach you to write programs
 - To teach you how to learn to program

Failure teaches Programming (and everything else)

- If you are not failing, you are not learning.
- Efficient learning is efficient failure. To wit:
 - Fail early: Try your best guess. It is probably wrong, but you will learn something if it is.
 - Fail often: The more you try, the more you learn.
 - Fail safely: Contain your failure so they don't break anything else. Try small changes.
 - Fail forward: Try to avoid repeating mistakes.
 Remember your failures.

Test Driven Development

- 1) Write a test
- 2) Run all tests to make sure the old ones pass and the new one fails
- 3) Write the smallest program that make the test pass
- 4) Run all test to make sure that all tests pass
- 5) Refactor

Why Test Driven Development

- Test Driven Development formalizes learning by failure.
 - Writing a test reminds you of your goal.
 - It is surprisingly easy to forget when face multiple failures
 - Making sure if fails tests the test.
 - If it succeeds before the code, it does not test the code
 - Writing the code is trying.
 - It will probably fail a few times. (That's good.)
 - Seeing the test succeed shows you have learned
 - You have written it down in the test and the code

1. Write a test

- Story: As a user, I want to move Scratch forward 250 pixels so I can see how to do it.
- Example: Scratch moves forward 250 pixels
- Test:
 - 1) Note Scratch's position
 - 2) Hit the green flag
 - 3) See: Scratch move forward 250 pixels
- What is the difference between the example and the test?
- Can testing tell your if a program is correct?

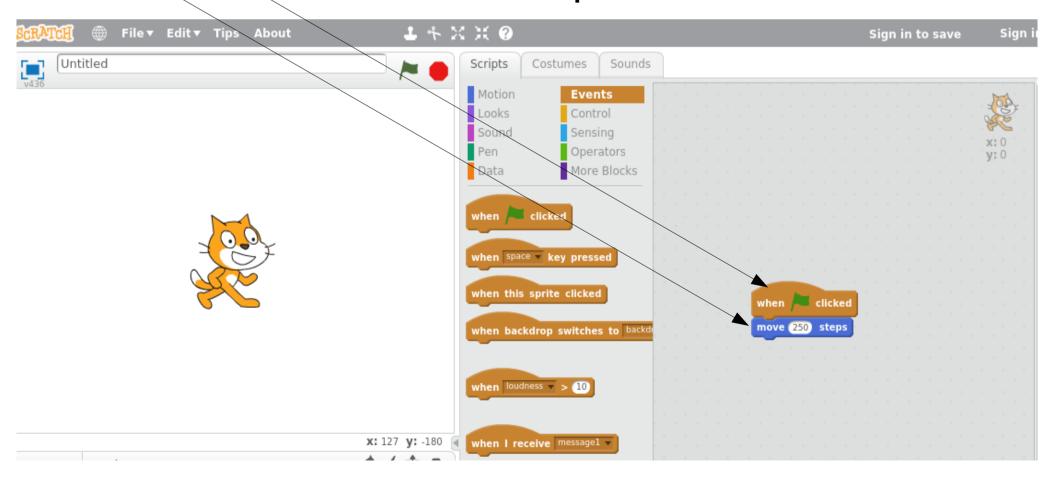
2. Run the Failing Test

- We hit the green flag. Scratch does not move.
 - So far there is only one test: see Scratch move.



3. Write the program

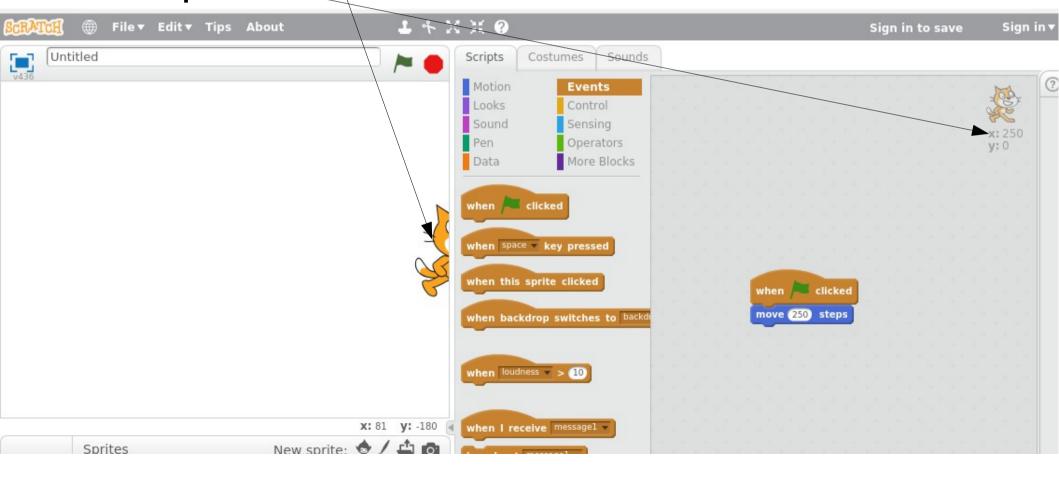
- Event: When green flag clicked
- Statement: Move 250 steps



4. Run a successful test

Scratch moves

• 250 pixels



5. Refactor

- Refactoring changes the description of behavior without changing the behavior.
- Refactoring makes the program
 comprehensible to another programming.
- No need to refactor a two line program

 How many descriptions of the same behavior are there?

Test Every Line

- Each time you write line of code:
 - 1. Think: "What will the program do when the line works?"
 - 2. Write the line of code.
 - 3. Compile the line of code. Does it compile?
 - 4. Run the line of code. Does it do what you expected?
 - 5. Make it as clear as possible.

Why test every line of code

- If you get a compiler error, you know what to fix.
- If you get a run time error, you know what to fix.
- Each additional line of code you write doubles to difficulty of finding errors.

Programming Languages

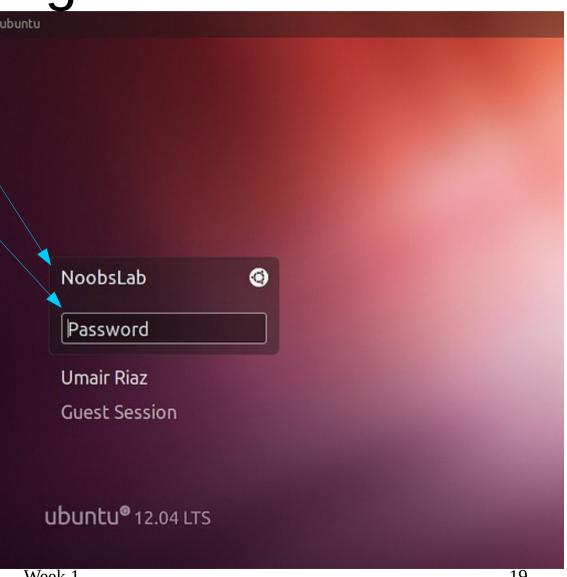
- High-level languages such as Scratch and C are written for people
- Computers do not understand high-level languages; they only understand machine code
- We write programs in high-level languages for other people. Often ourselves.

Why Unix

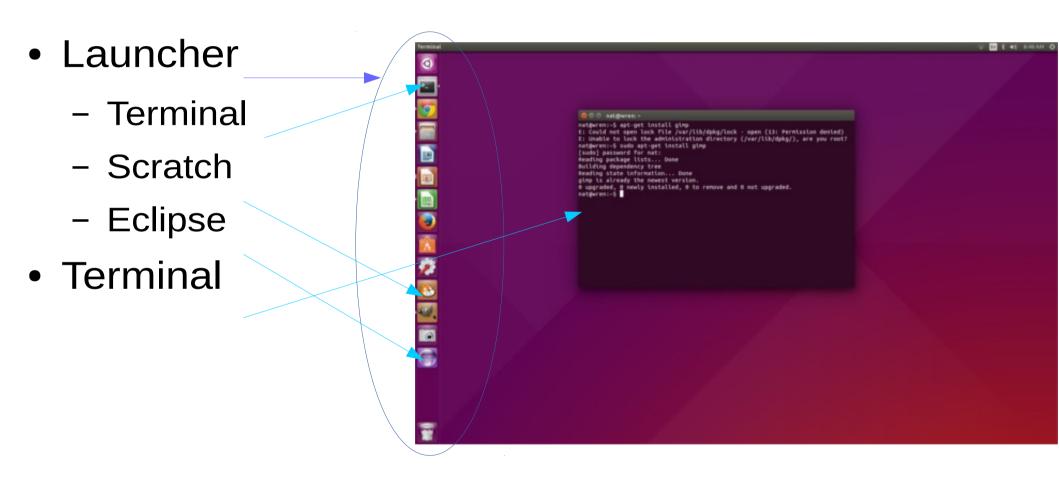
- Unix is a family of Operating System
 - Includes: OSX, iOS, Android, Linux
 - Alternative to Microsoft Windows
- We use Ubuntu Linux
 - Freely available—you can put it on your laptop
 - Comes with many programming tools
- More open than Windows
 - You can look at the code, which is written in C
 - Common in the Academy and Research

Login

- Select username
- Enter password



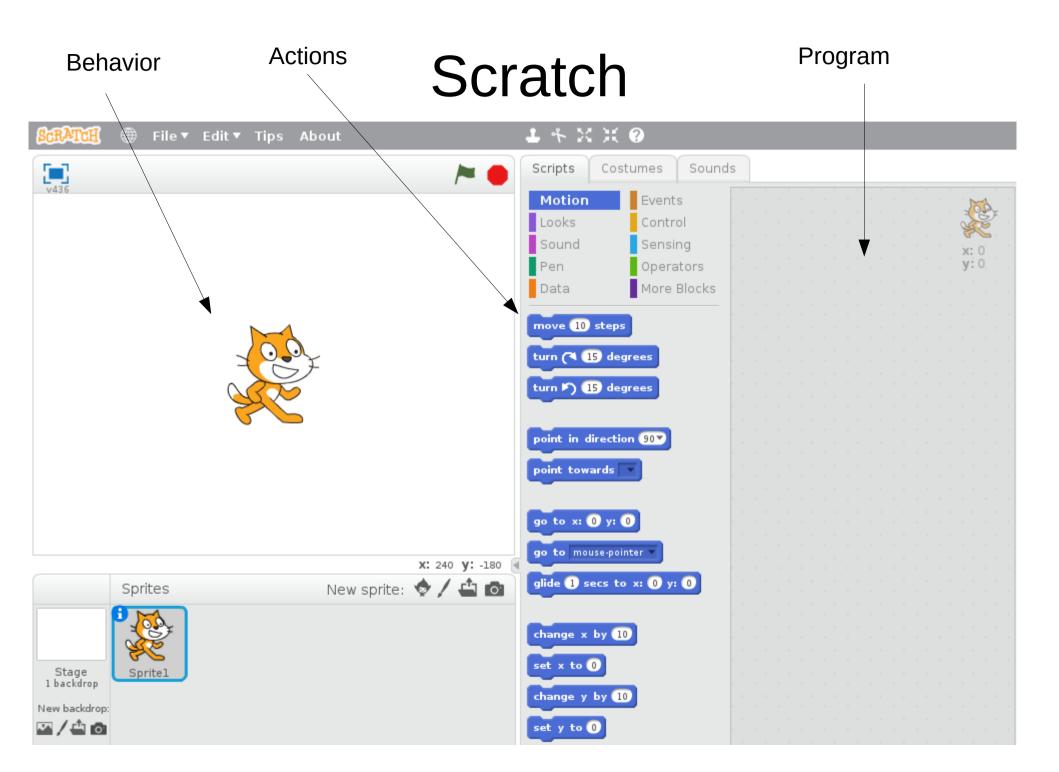
Ubuntu Linux Desktop



Terminal

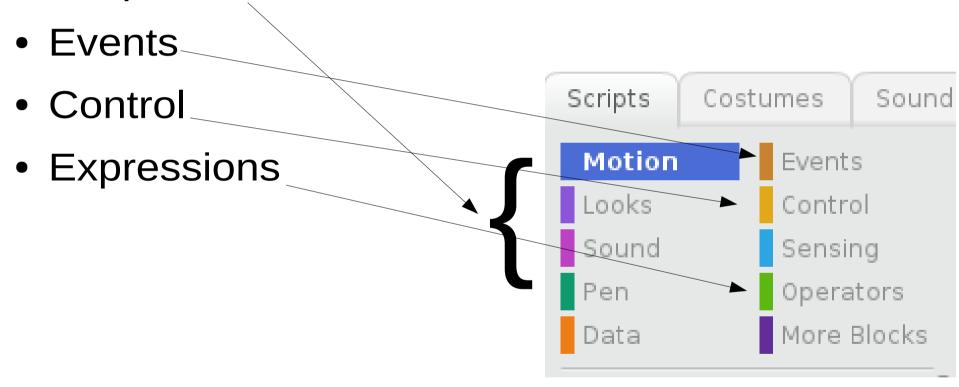
- Prompt
- Command
- Argument
- Result

```
nat@wren: ~
nat@wren:~$ ls -la
total 272
drwxr-xr-x 36 nat nat
                        4096 May 28 08:36 .
drwxr-xr-x 4 root root
                        4096 May 5 09:59 ...
drwxrwxr-x 3 nat nat
                        4096 May 17 10:57 .adobe
rwxrwxr-x 1 nat nat
                        9987 May 7 09:46 adobe-air.sh
                        4096 May 26 15:55 .appdata
           4 nat nat
                        6363 May 28 08:29 .bash history
                  nat
                         220 May 5 09:59 .bash_logout
           1 nat
                  nat
                 nat
                        3637 May 5 09:59 .bashrc
                        4096 May 28 08:32 .cache
drwxrwxr-x 2 nat nat
                        4096 May 14 15:09 chrome
drwxrwxr-x 3 nat nat
                        4096 May 28 08:20 classes
drwx----- 3 nat nat
                        4096 May 5 14:45 .compiz
drwx----- 22 nat nat
                        4096 May 28 08:44 .config
                 nat
                        4096 May 23 19:23 CUnit-2.1-3
drwxr-xr-x 11 nat
           3 root root
                        4096 May 5 14:14 .dbus
drwxr-xr-x 5 nat nat
                        4096 May 28 08:36 Desktop
rw-r--r-- 1 nat nat
                          25 May 5 10:00 .dmrc
                        4096 May 19 14:31 Documents
                        4096 May 28 08:44 Downloads
drwxr-xr-x 3 nat nat
drwxrwxr-x 3 nat
                        4096 May 5 19:39 .eclipse
                 nat
-rw-r--r-- 1 nat nat
                        8980 May 5 09:59 examples.desktop
drwx----- 4 nat nat
                        4096 May 23 09:38 .gconf
```



Statements

Simple Statements



Simple Statements: Motion

- Lets you move the sprite
- You can type numbers into the white areas
- You can select from the drop down menus

```
move 10 steps
turn (4 15 degrees
turn 🖍 15 degrees
point in direction 90
point towards
go to x: -182 y: 141
go to mouse-pointer
glide 1) secs to x: -182 y: 141
change x by 10
set x to 0
change y by 10
set y to 0
if on edge, bounce
set rotation style left-right
   x position
   y position
   direction
```

Simple Statements: Looks

 Lets you change the appearance of the stage and the sprite

```
say Hello! for 2 secs
say Hello!
think Hmm... for 2 secs
think Hmm...
show
switch costume to costume2 >
next costume
switch backdrop to backdrop1
change color v effect by 25
set color effect to 0
clear graphic effects
change size by 10
set size to 100 %
go to front
go back 1 layers
   costume #
   backdrop name
```

Simple Statements: Sound

Lets you play sounds

```
play sound meow
play sound meow v until done
stop all sounds
play drum 1 for 0.25 beats
rest for 0.25 beats
play note 60♥ for 0.5 beats
set instrument to 1
change volume by -10
set volume to 100 %
  volume
change tempo by 20
set tempo to 60 bpm
tempo
```

Simple Statements: Pen

Lets you play draw on the stage

```
clear
stamp
pen down
pen up
set pen color to
change pen color by 10
set pen color to 0
change pen shade by 10
set pen shade to 50
change pen size by 1
set pen size to 1
```

Events: Starting Programs

- Starts programs
- Events are actions that occur outside the program.
- Programs can generate events using "broadcast"
 - Broadcast is an asynchronous event
 - Broadcast and wait is a synchronous event

```
when space very pressed

when this sprite clicked

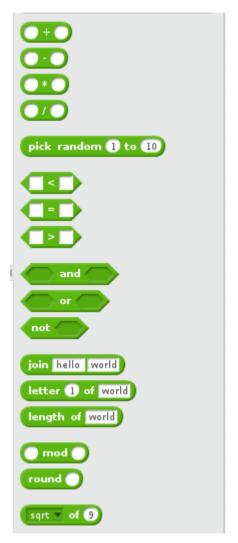
when backdrop switches to backd

when loudness very > 10

when I receive messagel very broadcast messagel very and wait
```

Expressions: Testing and Calculating

- Perform calculations
- Arithmetic expressions calculate numbers
 - e.g. 1 + 2, 3* 4
- Boolean expressions evaluate to true or false
 - e.g. 1 < 2, 5 = 6
 - e.g. 1 < 2 or 5 = 6



Control: Choosing next action

- Selection and Iteration
- Selection (if) chooses between one of two actions
- Iteration (repeat and forever) repeat actions
 - Repeat n times
 - Repeat until

```
wait 1 secs
stop all
 vhen I start as a clone
delete this clone
```

Draw a Square 1

