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

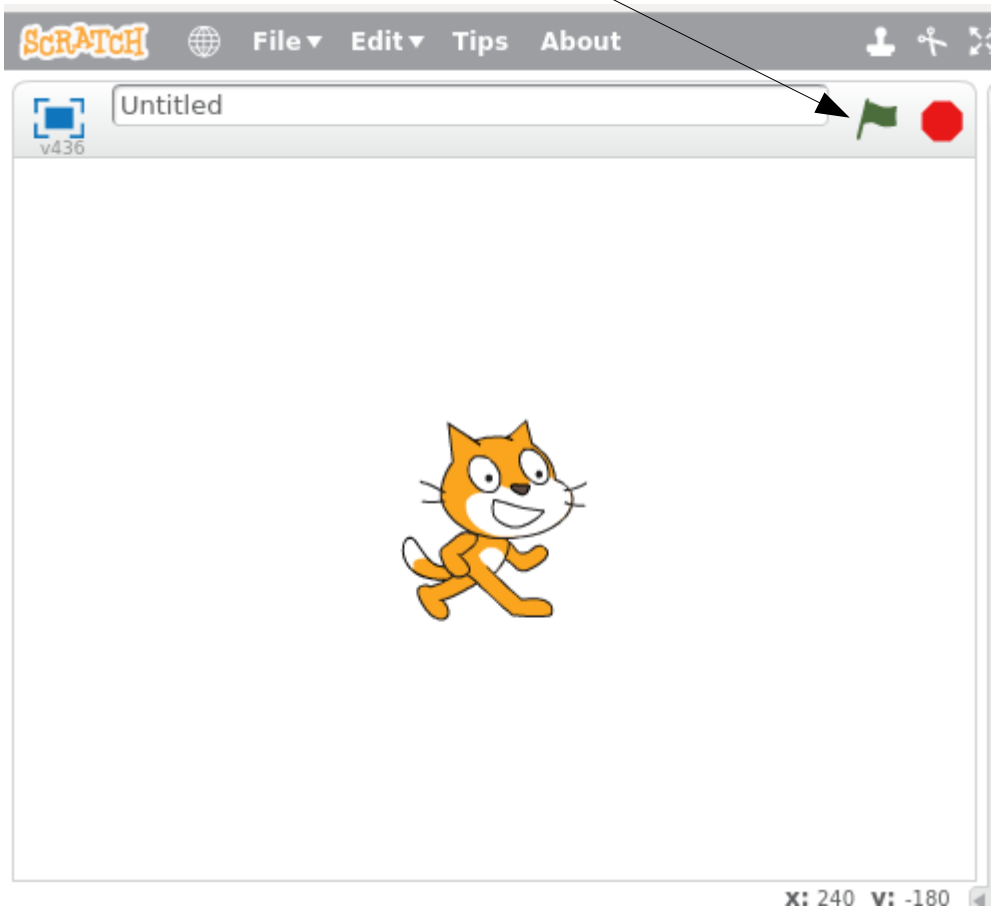
- **Story**: As a user, I want to move Scratch forward 250 pixels so I can see how to do it.
- **Example**: Scratch moves forward 250 pixels
- **Test**:

  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?

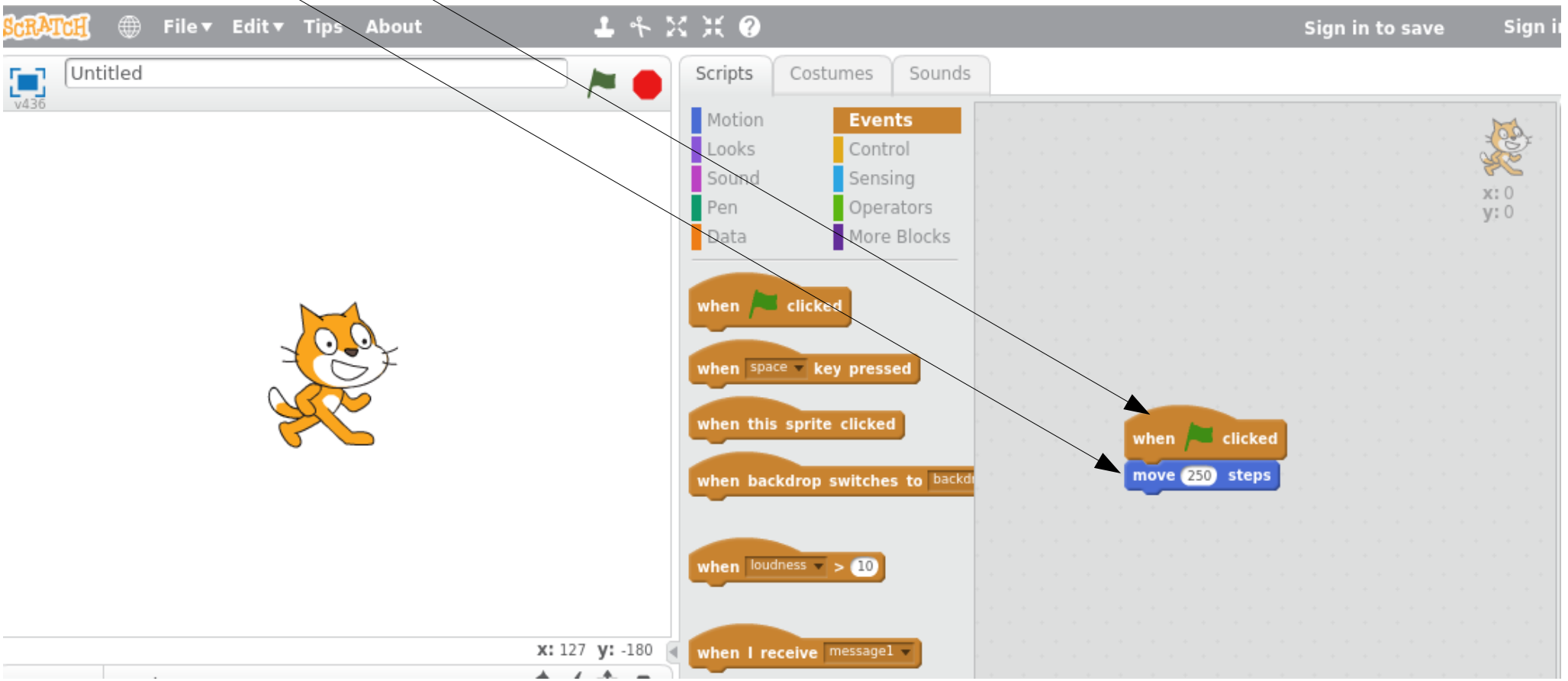
2. Run the Failing Test

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



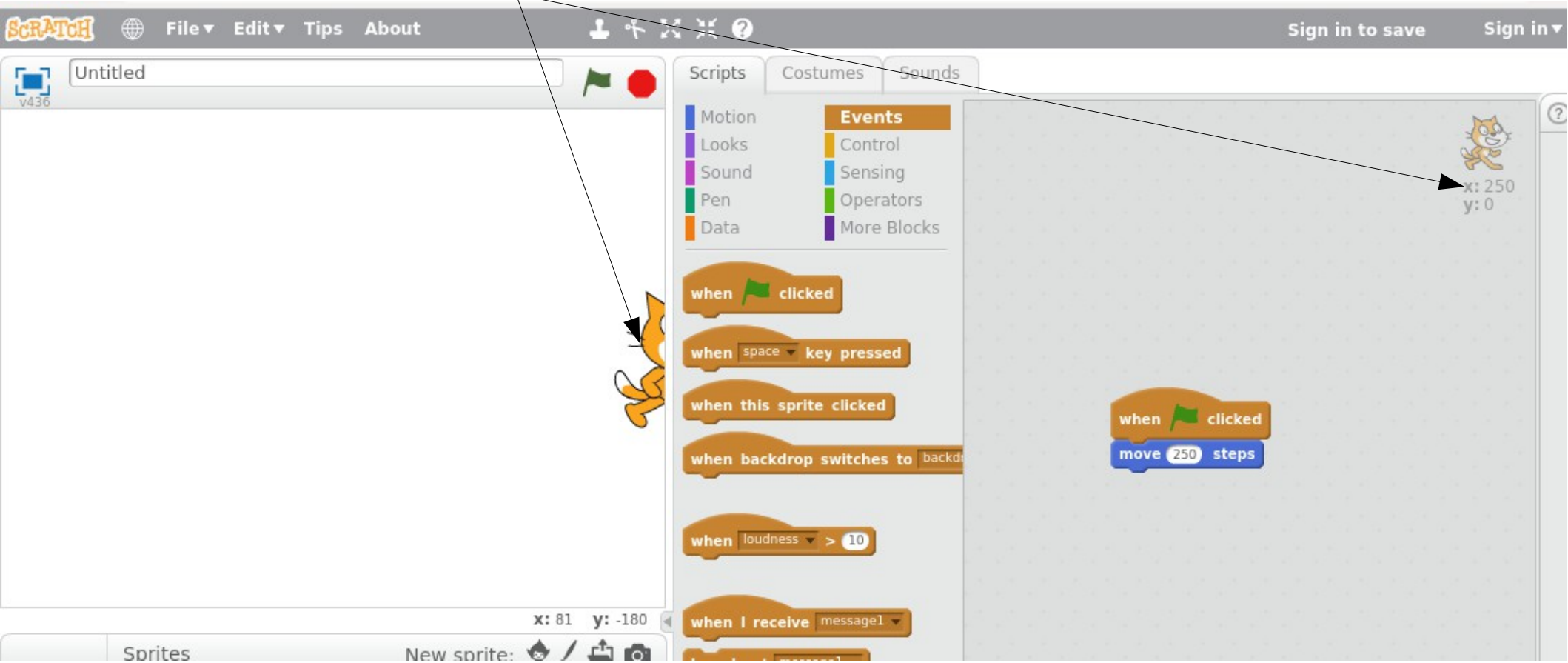
3. Write the program

- 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

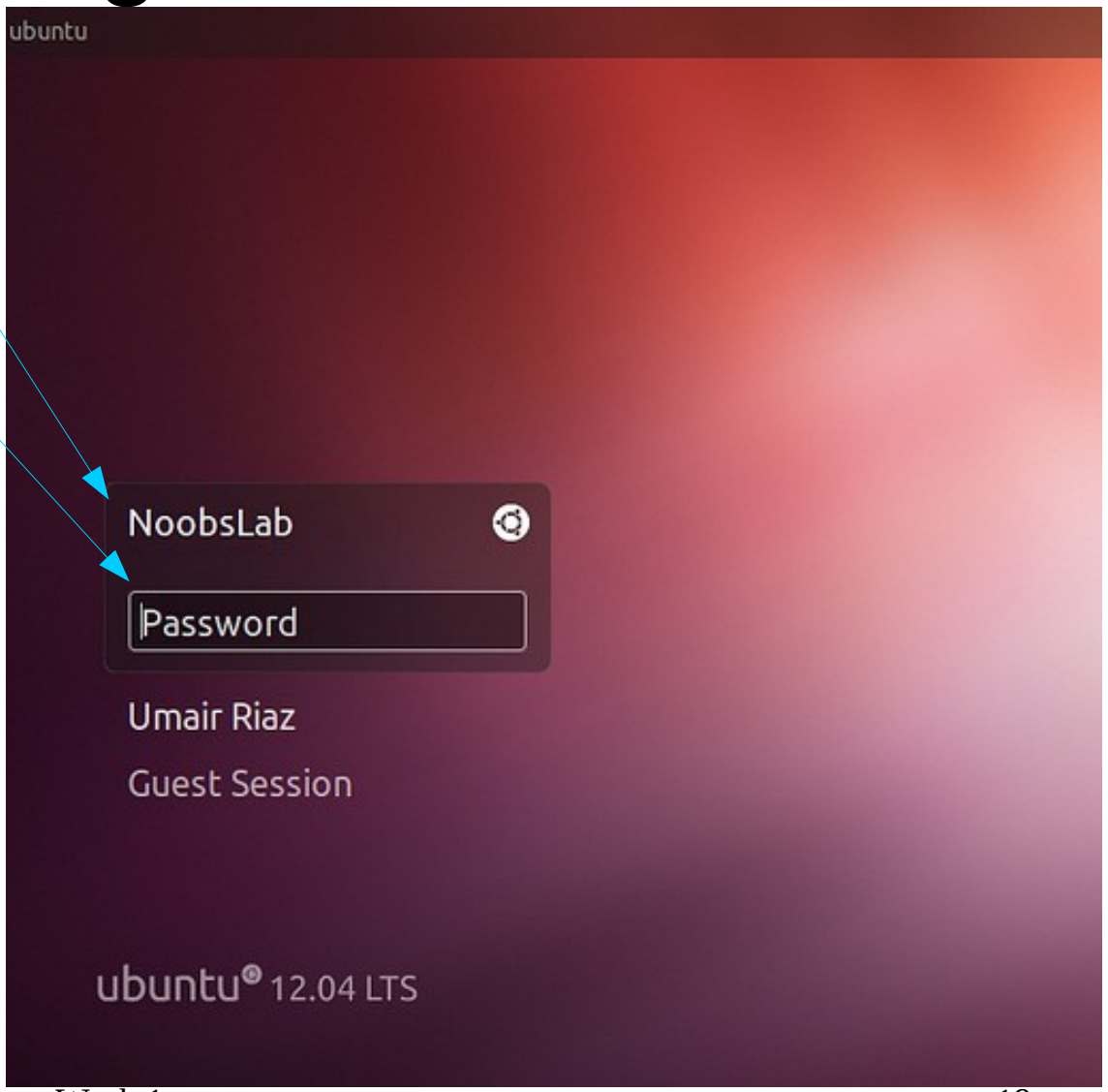
- High-level languages such as Scratch and C are written for people
- Computers do not understand high-level languages; they only understand machine code
- We write programs in high-level languages for other people. Often ourselves.

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

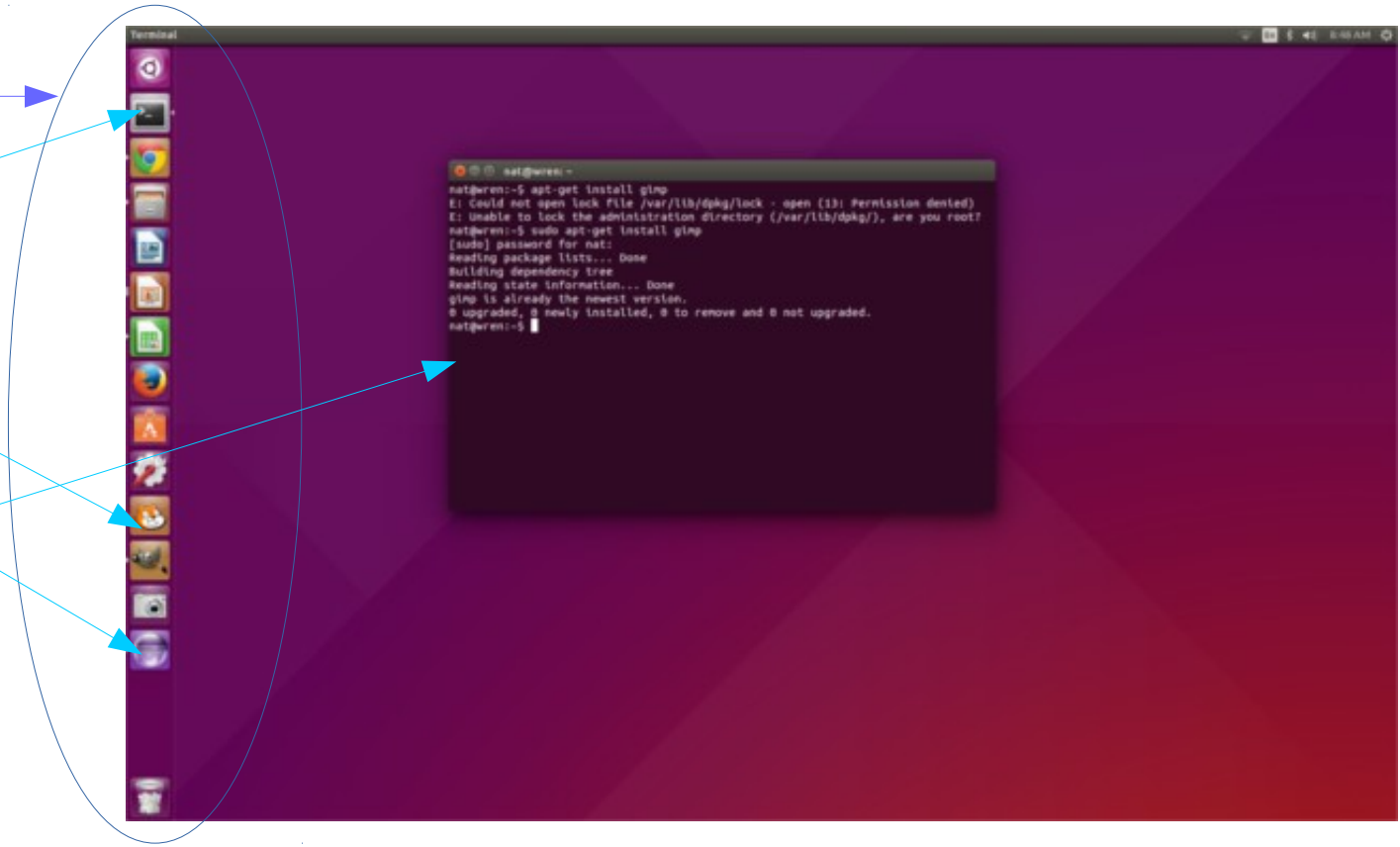
Login

- Select username
- Enter password



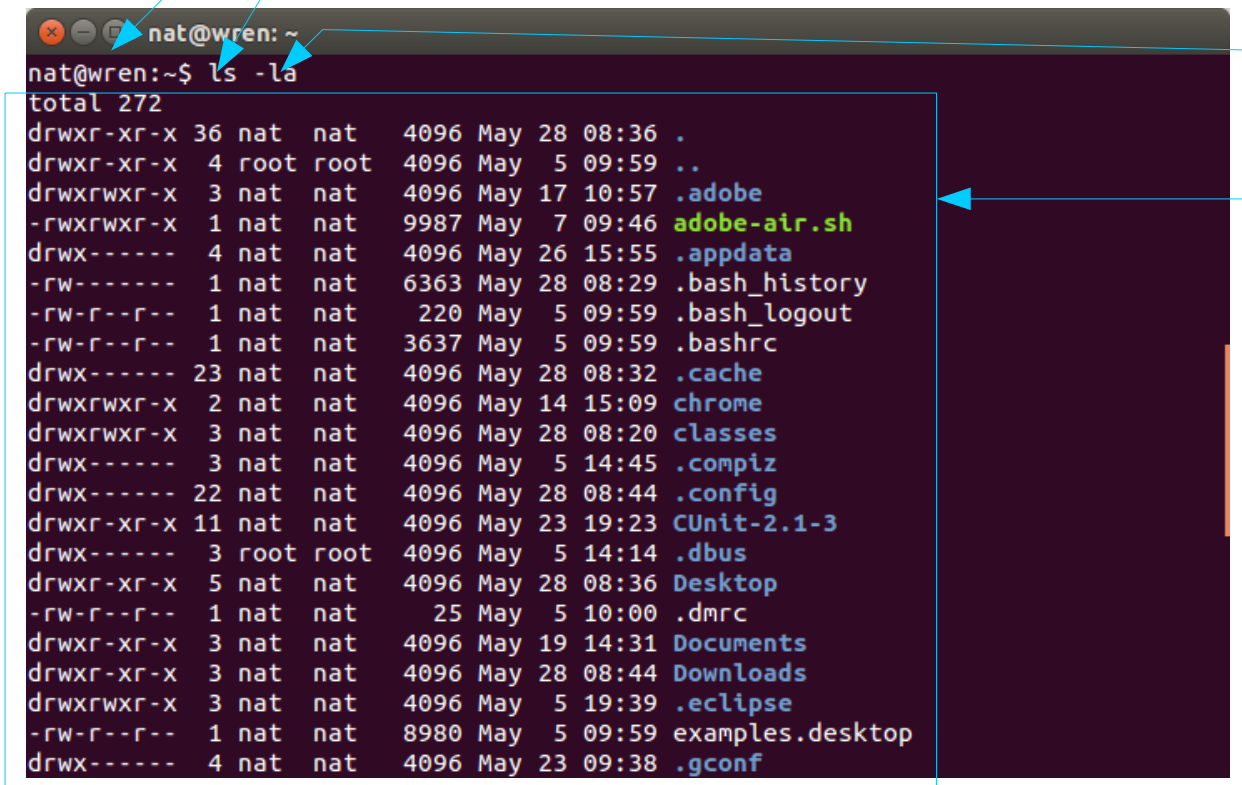
Ubuntu Linux Desktop

- Launcher
  - Terminal
  - Scratch
  - Eclipse
- Terminal



Terminal

- Prompt
- Command
- Argument
- Result

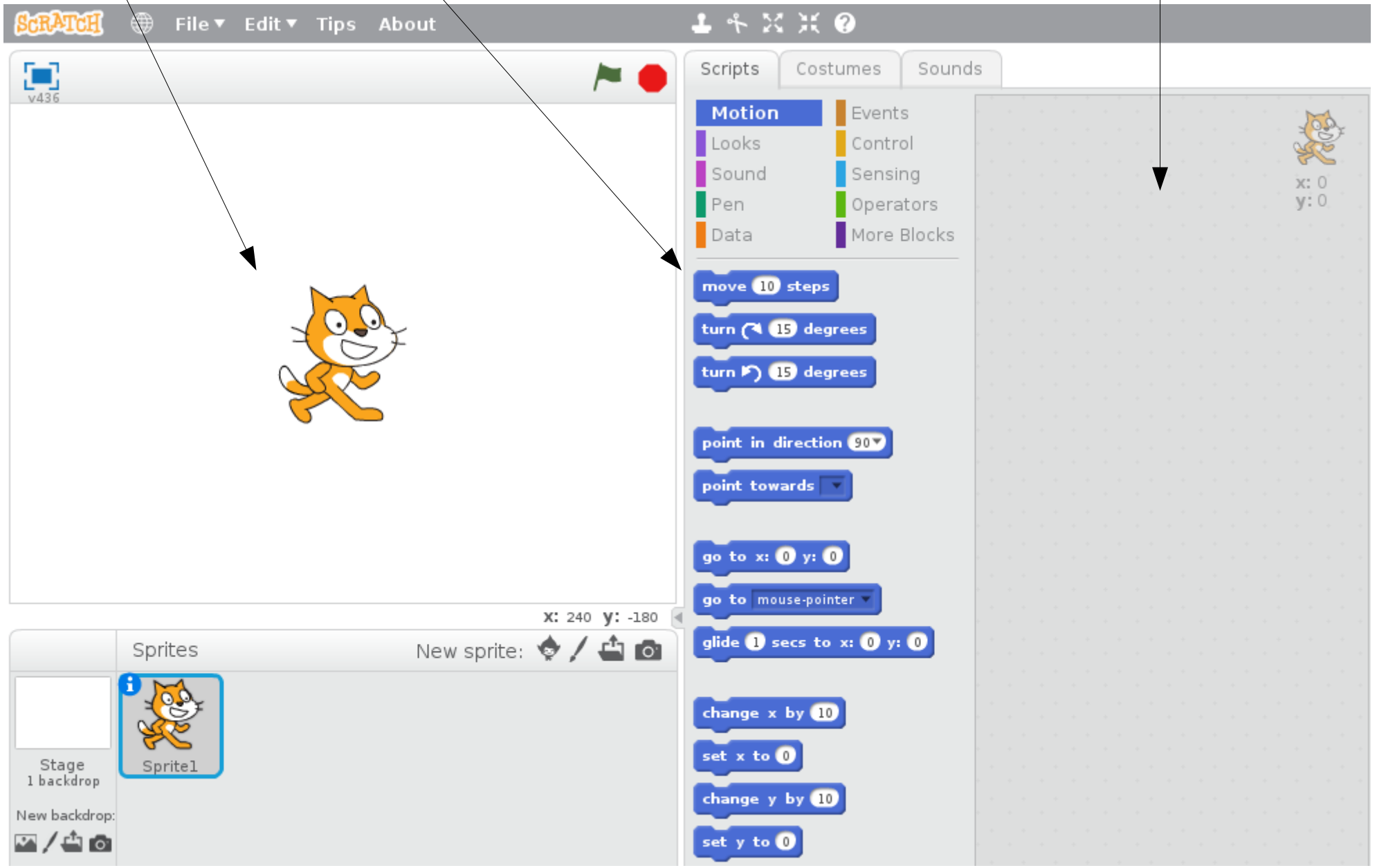


Scratch

Behavior

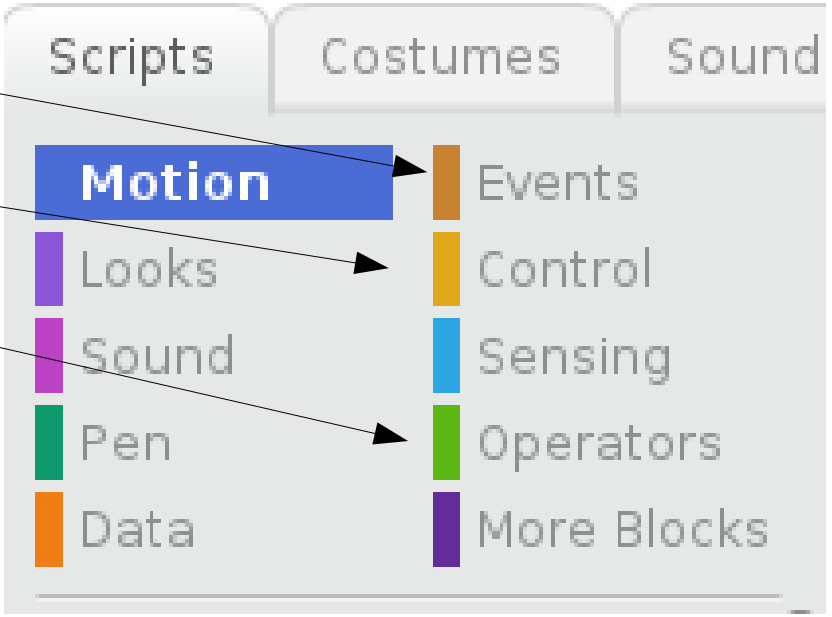
Actions

Program



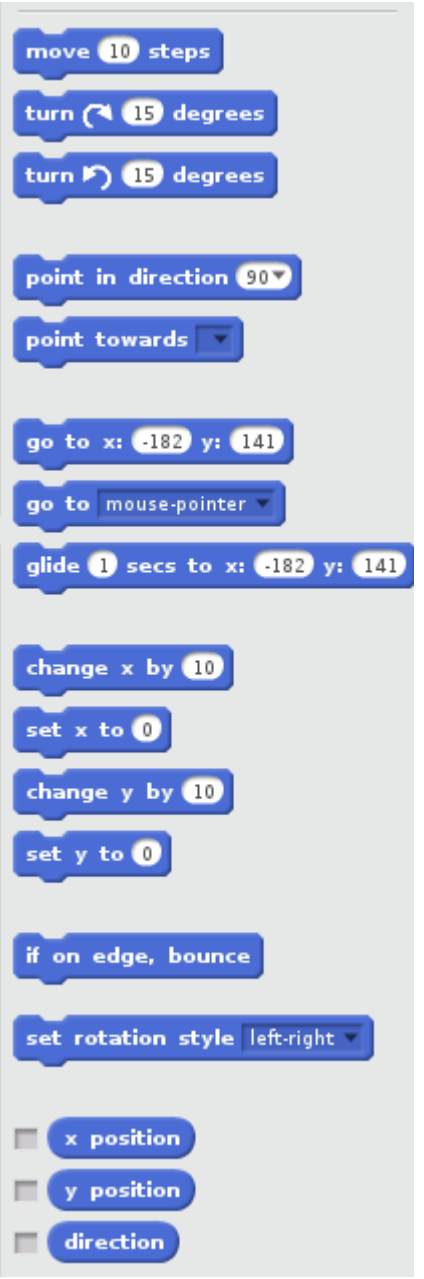
Statements

- Simple Statements
- Events
- Control
- Expressions



Simple Statements: Motion

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

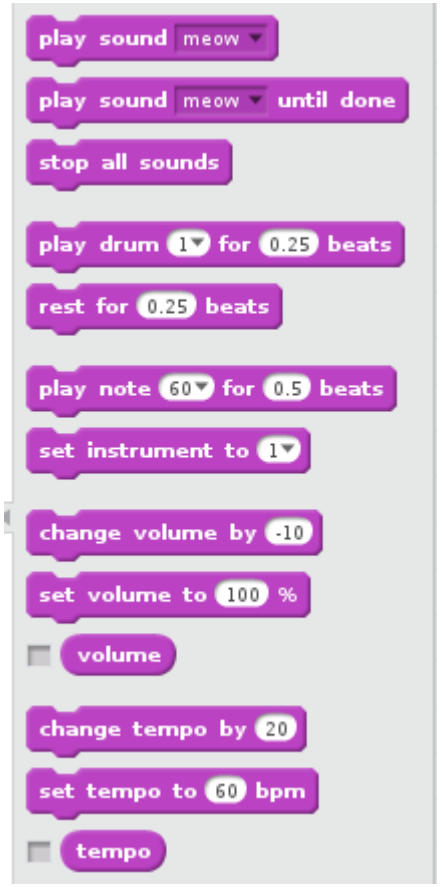


Simple Statements: Looks

- Lets you change the appearance of the stage and the sprite

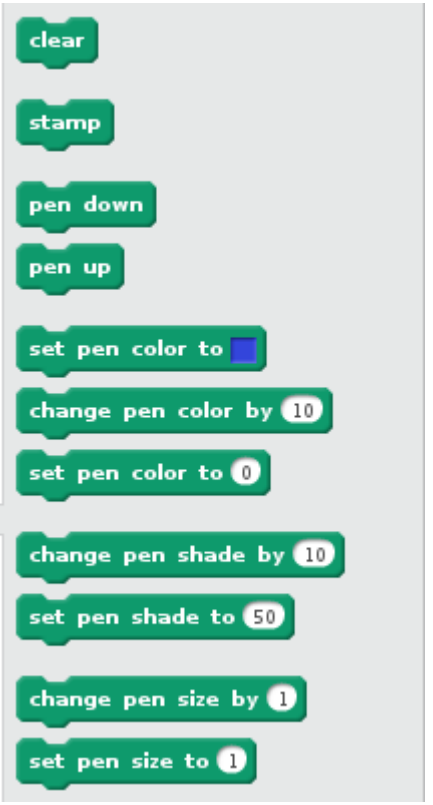
Simple Statements: Sound

- Lets you play sounds



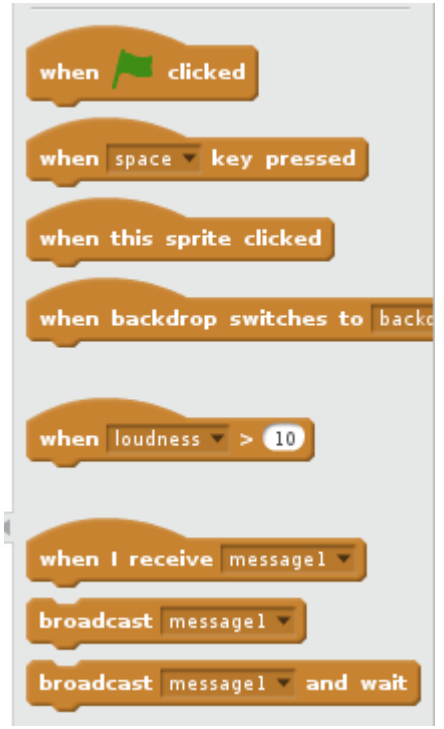
Simple Statements: Pen

- Lets you play draw on the stage



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

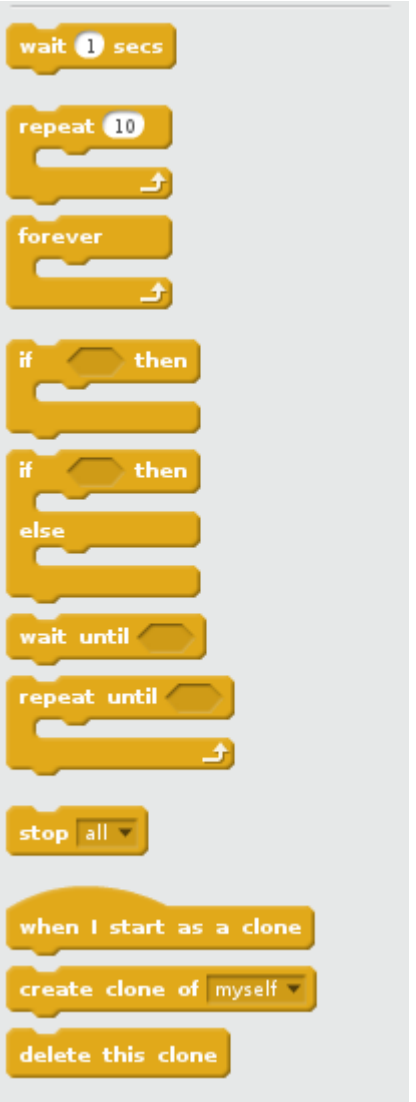


Expressions: Testing and Calculating

- 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



Draw a Square 1

- Reset
- Draw a line
- Turn 90 degrees
- Draw and turn three more times