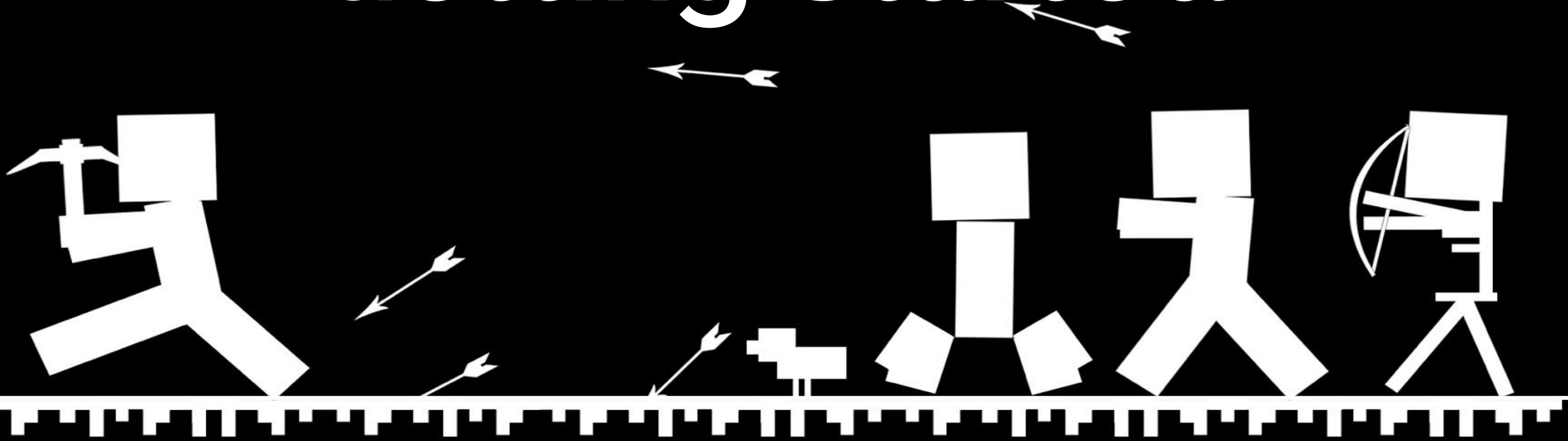
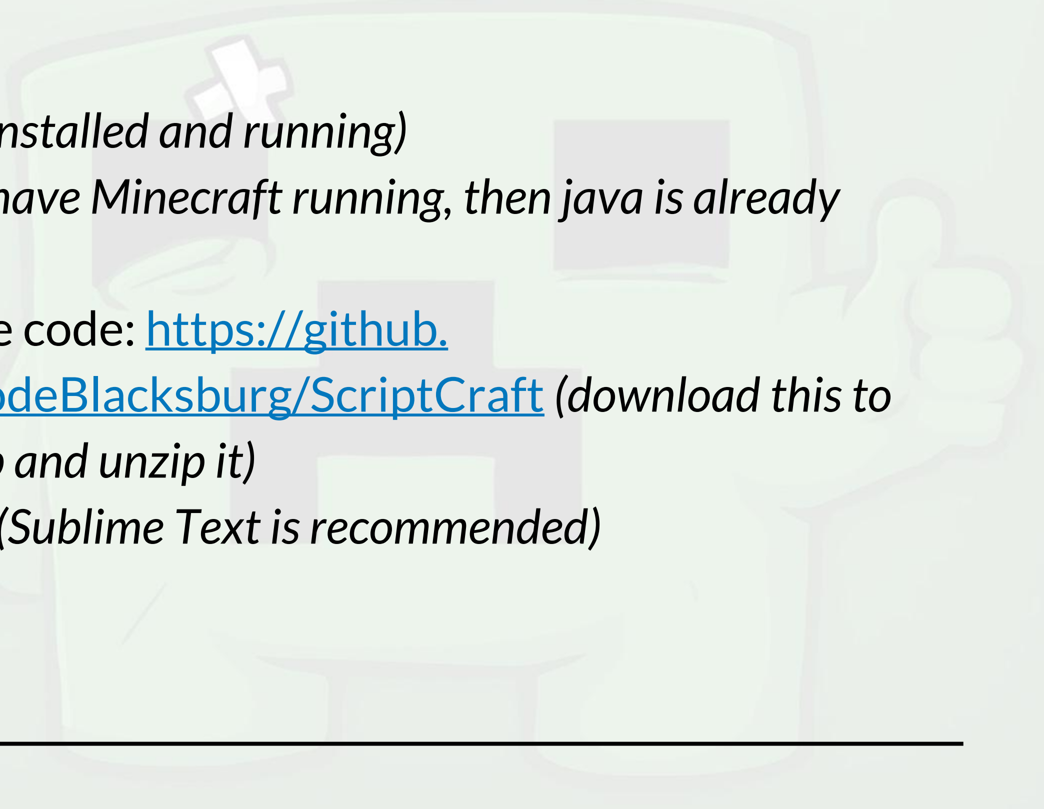
Writing Plugins in Minecraft with JavaScript

Getting Started

Getting Started

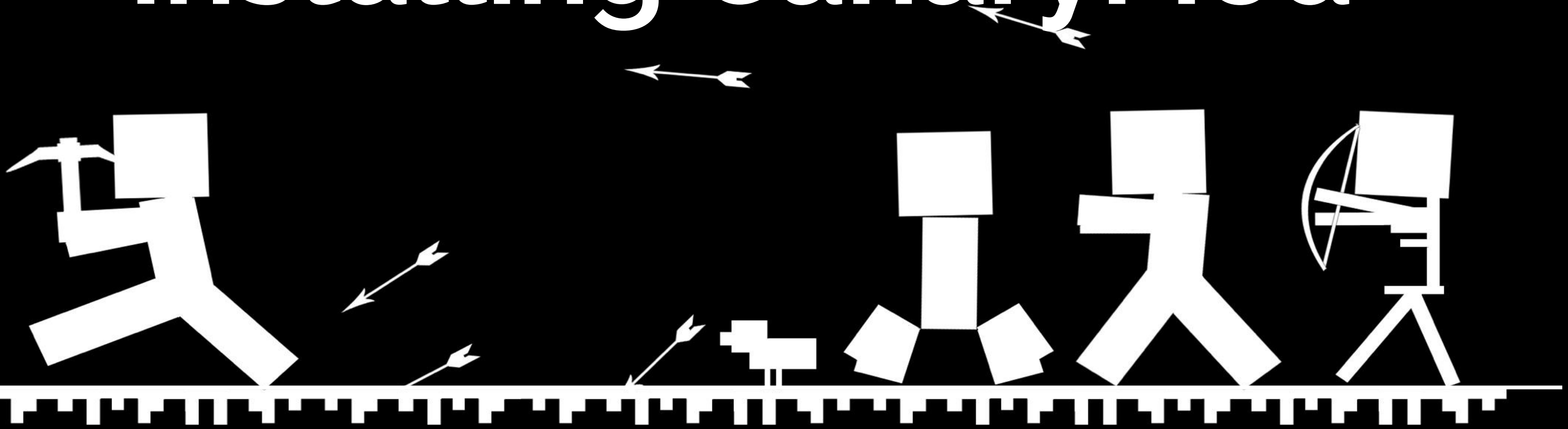


What You’ll Need

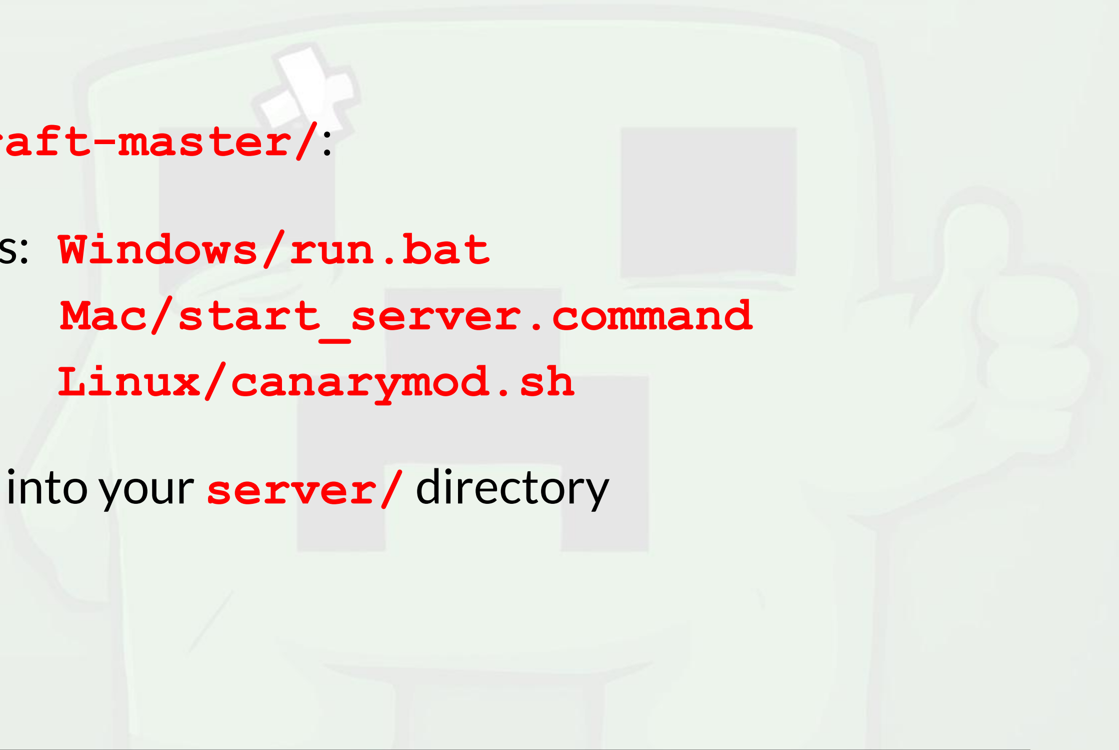


1. Minecraft *(installed and running)*
2. Java *(if you have Minecraft running, then java is already installed)*
3. Class source code: https://github.com/LetsCodeBlacksburg/ScriptCraft *(download this to your desktop and unzip it)*
4. Text editor *(Sublime Text is recommended)*

Installing CanaryMod



Find your OS's launcher script



In `ScriptCraft-master/`:

- Windows: `Windows/run.bat`
- Mac: `Mac/start_server.command`
- Linux: `Linux/canarymod.sh`

Copy this file into your `server/` directory

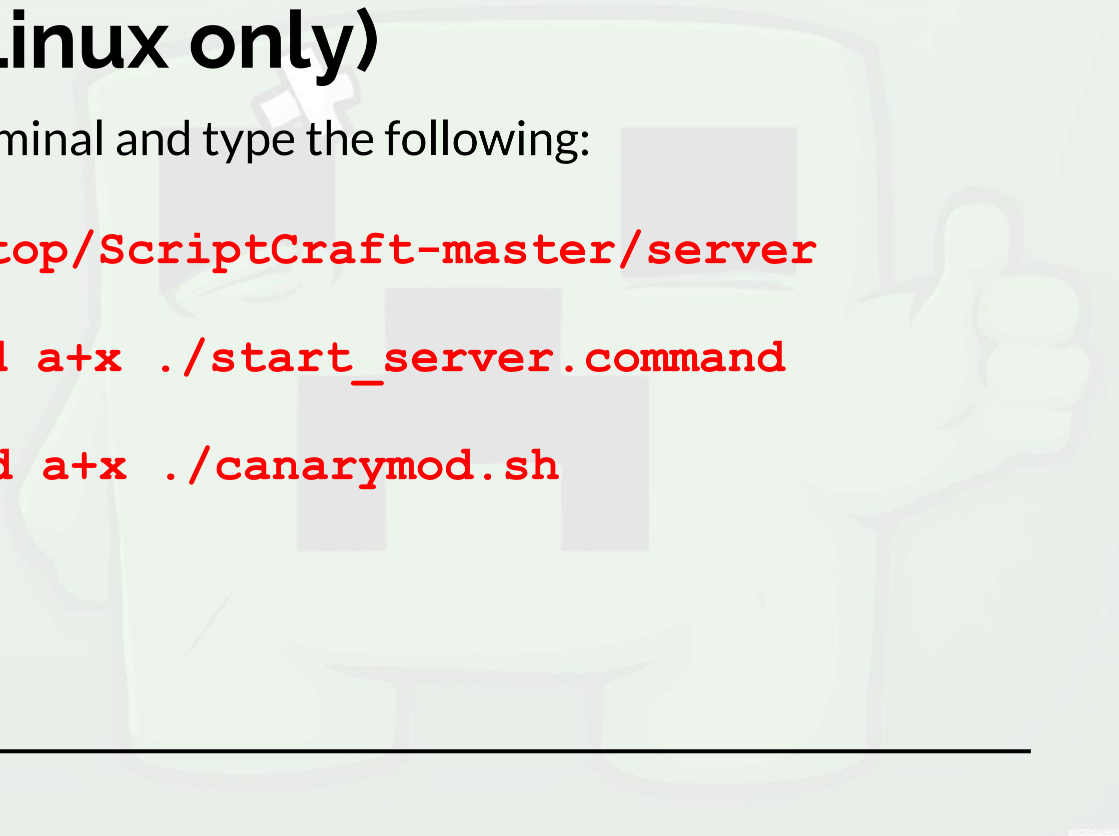
Making the script executable (Mac/Linux only)

Open the terminal and type the following:

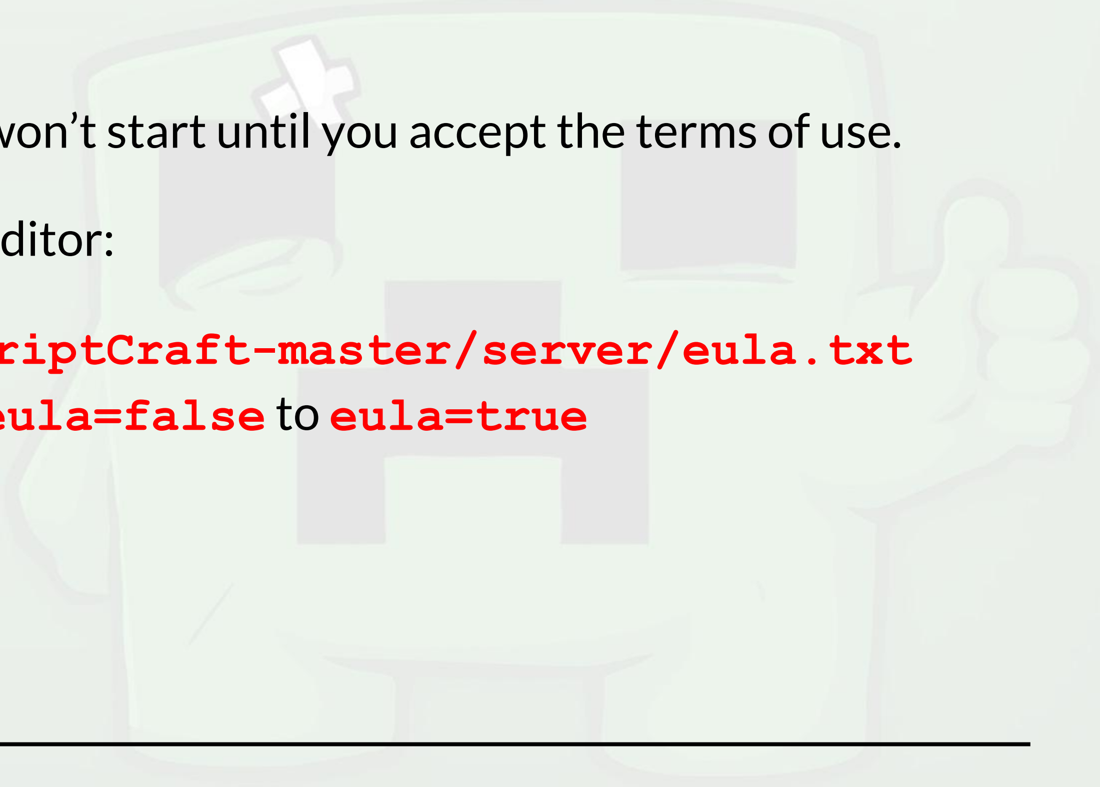
```
cd ~/Desktop/ScriptCraft-master/server
```

Mac:
```
chmod a+x ./start_server.command
```

Linux:
```
chmod a+x ./canarymod.sh
```



Accepting the EULA

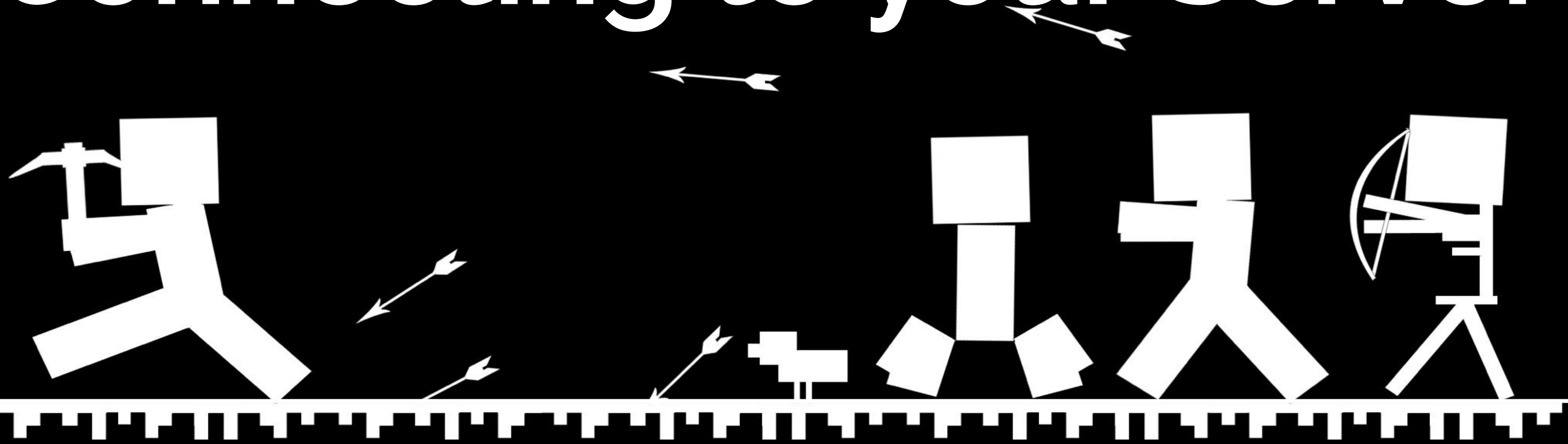


Your server won’t start until you accept the terms of use.

In your text editor:

- open `ScriptCraft-master/server/eula.txt`
- change `eula=false` to `eula=true`

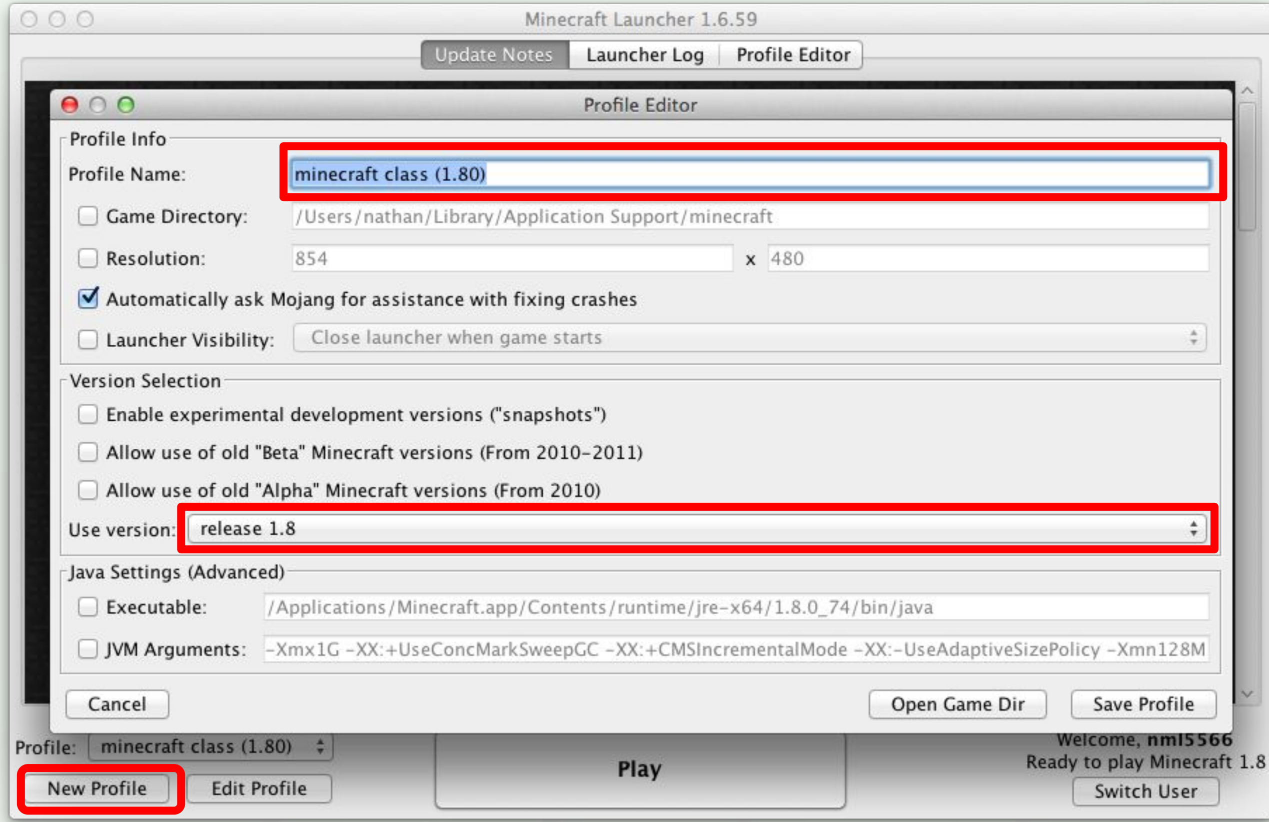
Connecting to your Server



Minecraft Profile Editor

Click on the *New Profile* button

Create a profile that uses release version 1.8.0



Minecraft Launcher 1.6.59

Profile Editor

Profile Info

Profile Name: minecraft class (1.80)

[ ] Game Directory: /Users/nathan/Library/Application Support/minecraft

[ ] Resolution: 854 x 480

[✓] Automatically ask Mojang for assistance with fixing crashes

[ ] Launcher Visibility: Close launcher when game starts

Version Selection

[ ] Enable experimental development versions ("snapshots")

[ ] Allow use of old "Beta" Minecraft versions (From 2010-2011)

[ ] Allow use of old "Alpha" Minecraft versions (From 2010)

Use version: release 1.8

Java Settings (Advanced)

[ ] Executable: /Applications/Minecraft.app/Contents/runtime/jre-x64/1.8.0_74/bin/java

[ ] JVM Arguments: -Xmx1G -XX:+UseConcMarkSweepGC -XX:+CMSIncrementalMode -XX:-UseAdaptiveSizePolicy -Xmn128M

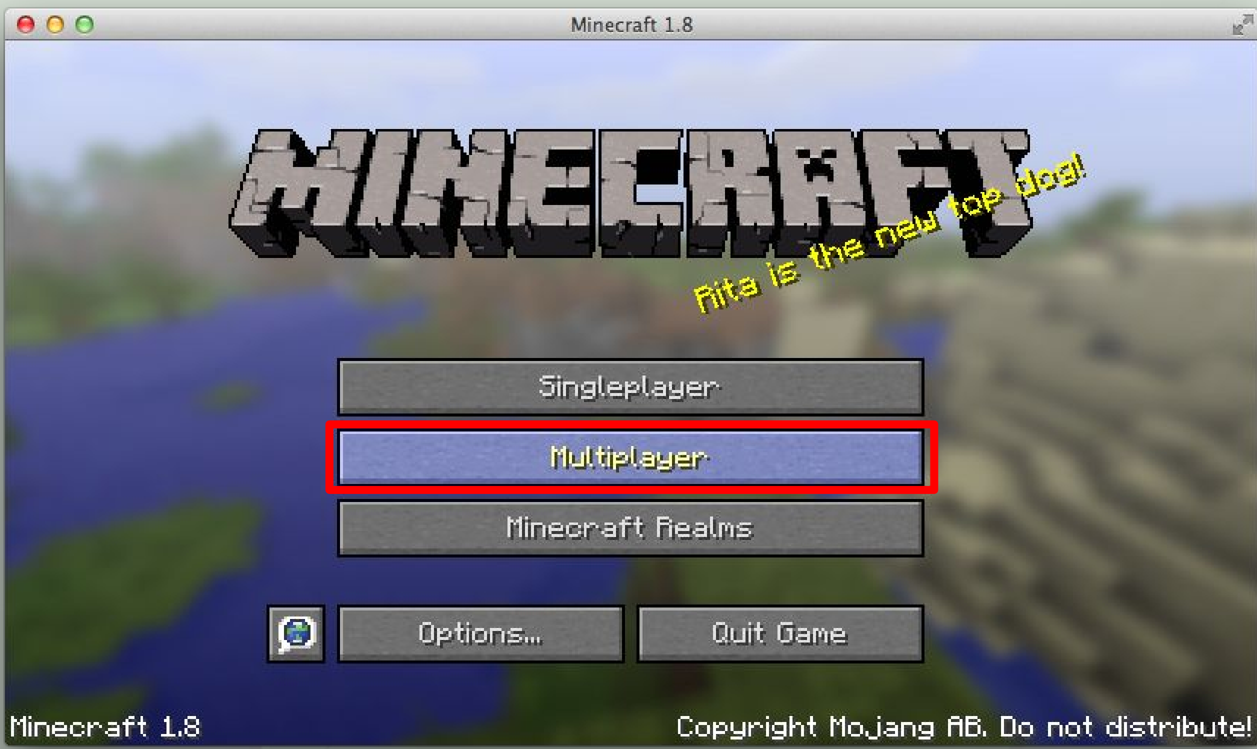
Profile: minecraft class (1.80)

Welcome, nml5566
Ready to play Minecraft 1.8

Connecting Client to Server

Launch the game and click on *Multiplayer*.

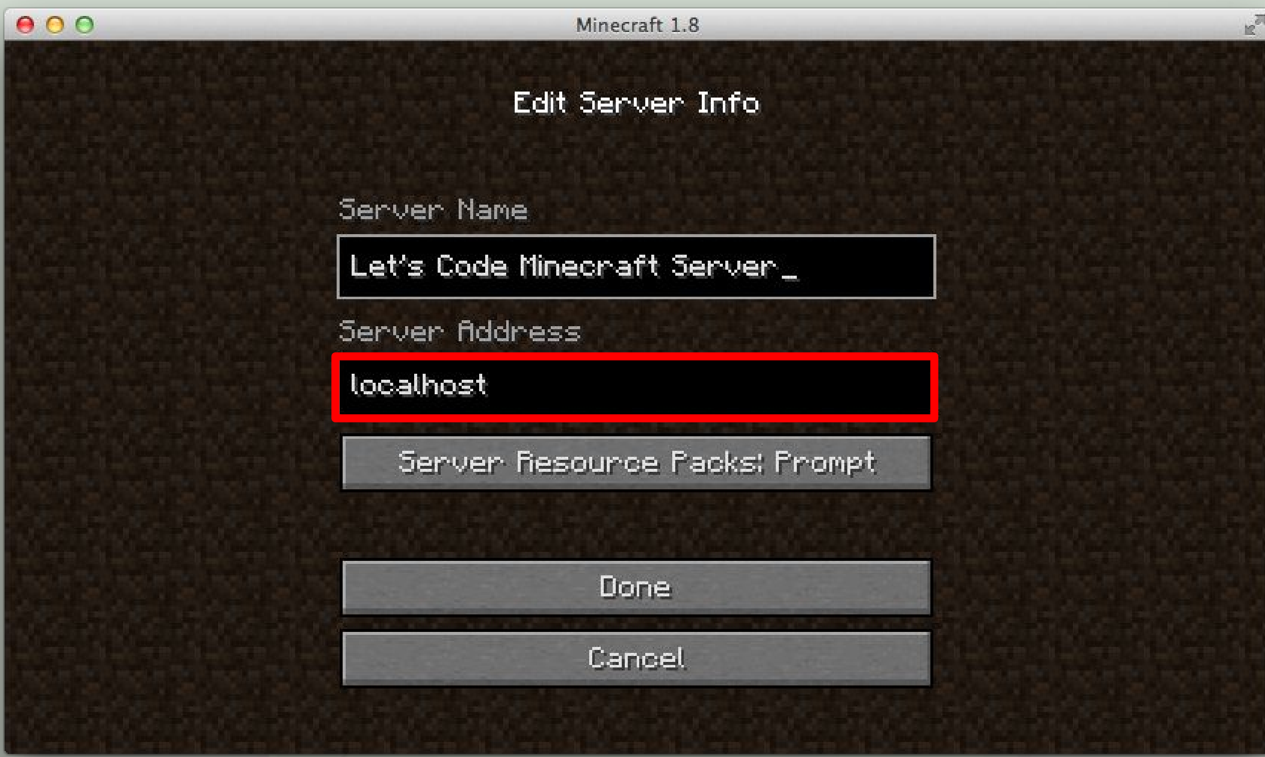
Next, click *Add Server* and type your server's name



Adding Your Server

Give your server a distinct name.

Type **localhost** in the *Server Address* field.

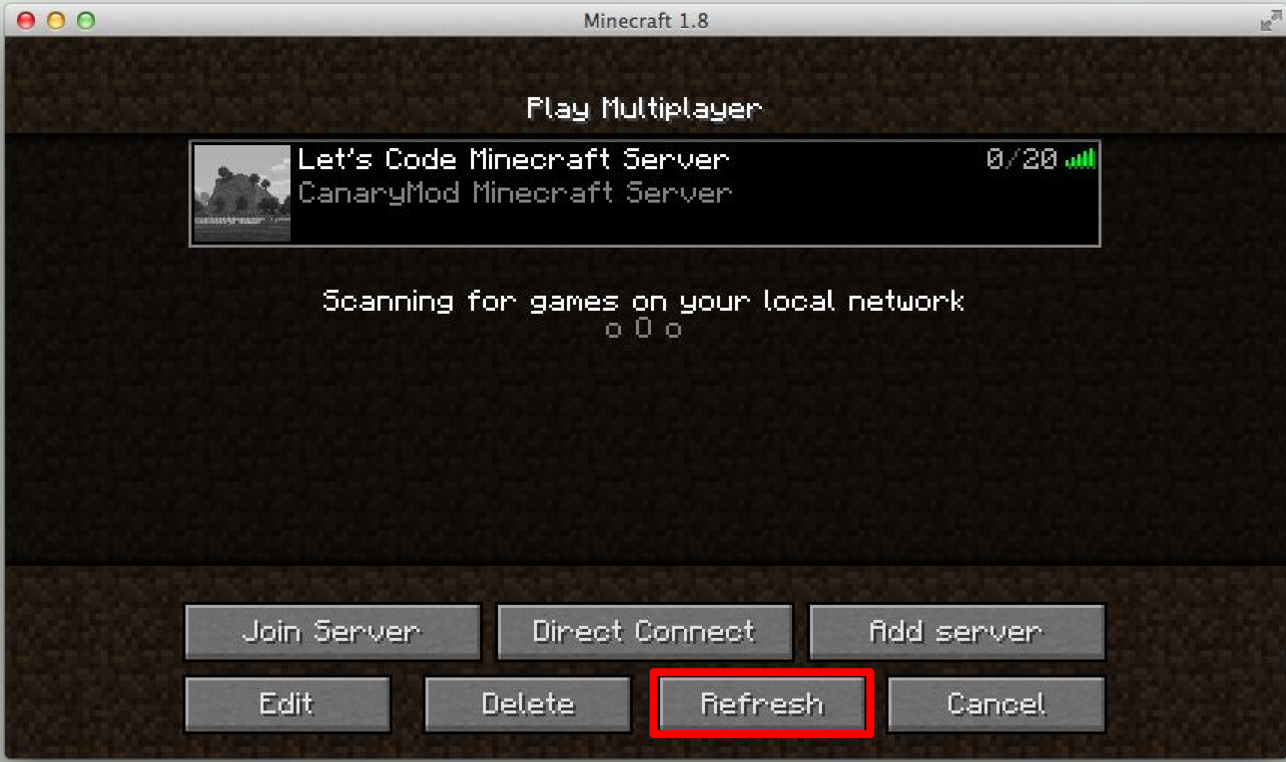


Server Name
Let's Code Minecraft Server_

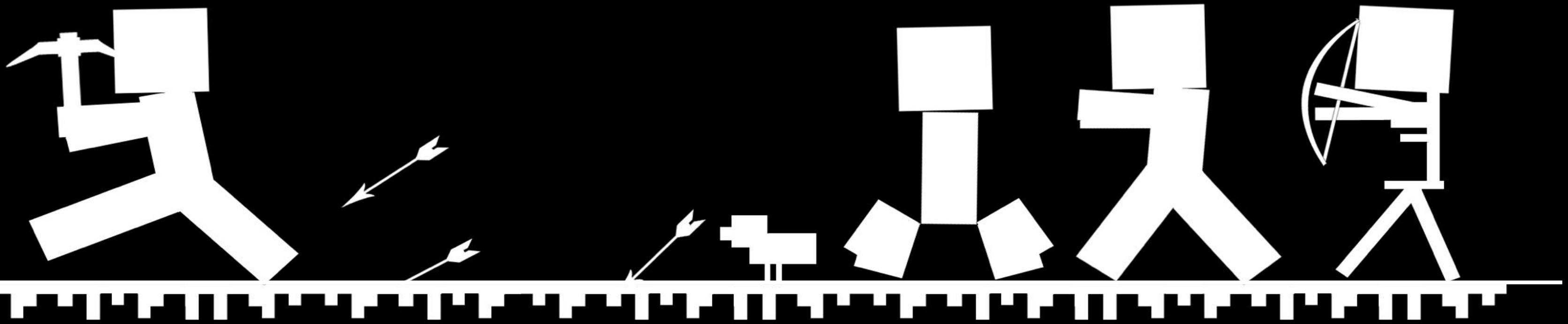
Server Address
localhost

Joining Your Server

Try clicking *Refresh* if nothing shows up.


Minecraft 1.8
Play Multiplayer
Let's Code Minecraft Server
CanaryMod Minecraft Server
0/20
Scanning for games on your local network
Join Server
Direct Connect
Add server
Edit
Delete
Refresh
Cancel

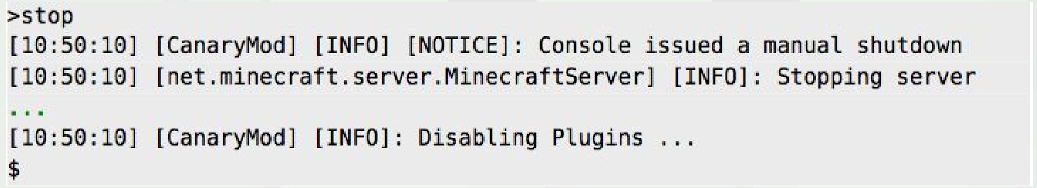
Installing ScriptCraft



Adding ScriptCraft to Plugins

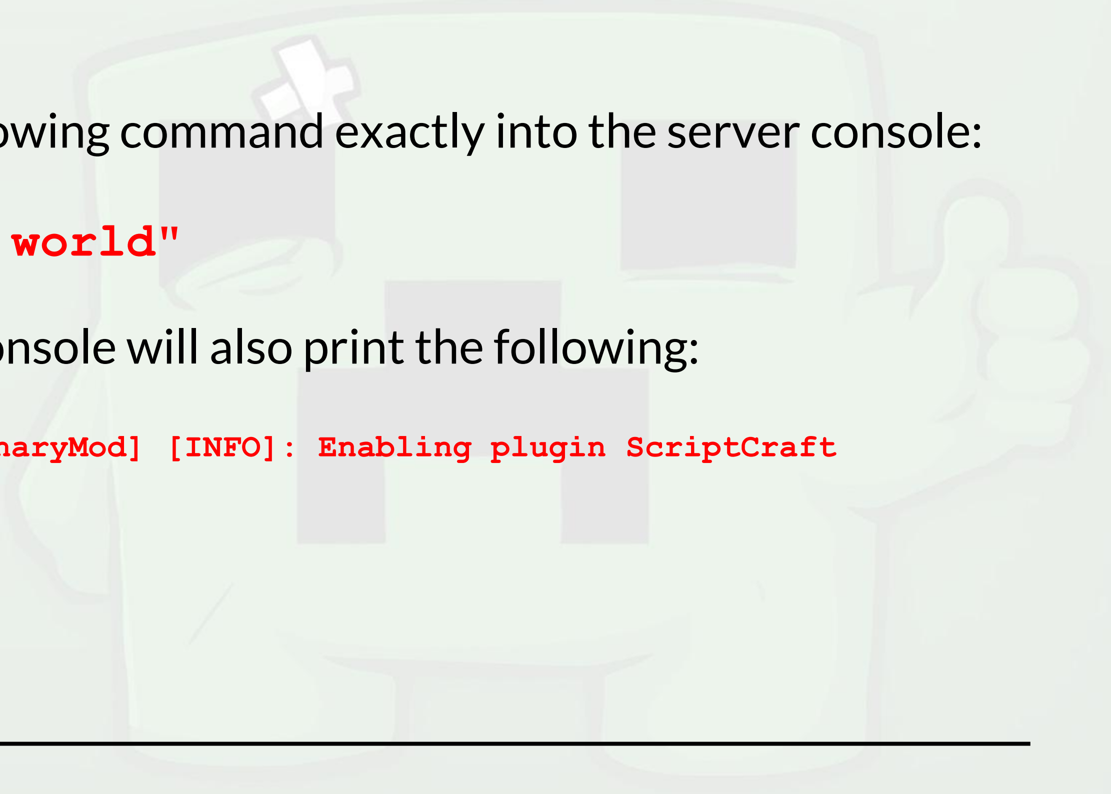
In `ScriptCraft-master/plugins/`:

1. Find `scriptcraft.jar`
2. Copy this file into your `server/plugins/` directory
3. Restart the server
   (type the `stop` *command into the server console, then relaunch it by double-clicking on your startup script)*



```
>stop
[10:50:10] [CanaryMod] [INFO] [NOTICE]: Console issued a manual shutdown
[10:50:10] [net.minecraft.server.MinecraftServer] [INFO]: Stopping server
...
[10:50:10] [CanaryMod] [INFO]: Disabling Plugins ...
$
```

Verifying ScriptCraft is Installed



Type the following command exactly into the server console:

```
js "Hello world"
```

The server console will also print the following:

```
[19:22:21] [CanaryMod] [INFO]: Enabling plugin ScriptCraft
```

Giving yourself OP

This is necessary to run JavaScript commands in-game and break blocks. You can only do this *after* you've logged into your server.

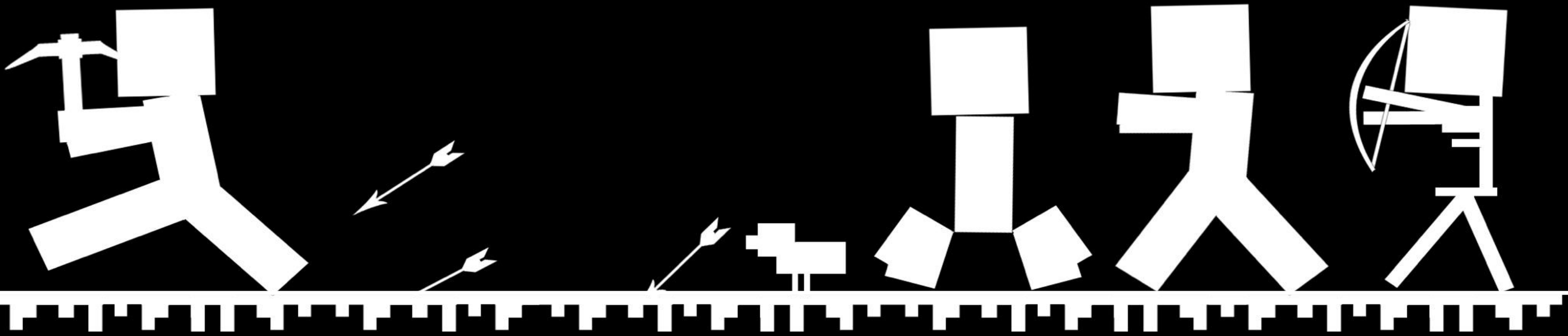
Type the following command exactly into the server console:

```
op <username>
```

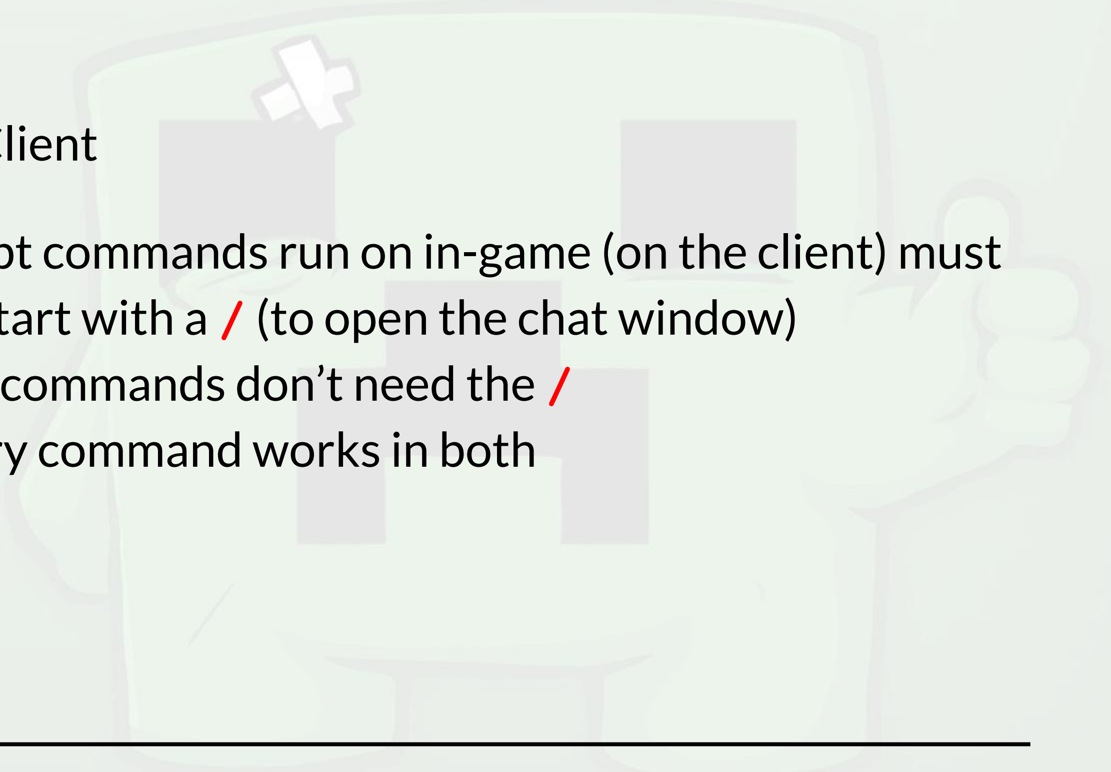
The server console will print the following:

```
[11:31:09] [CanaryMod] [INFO]: [SERVER] Opped <username>
```

Exploring JavaScript in Minecraft



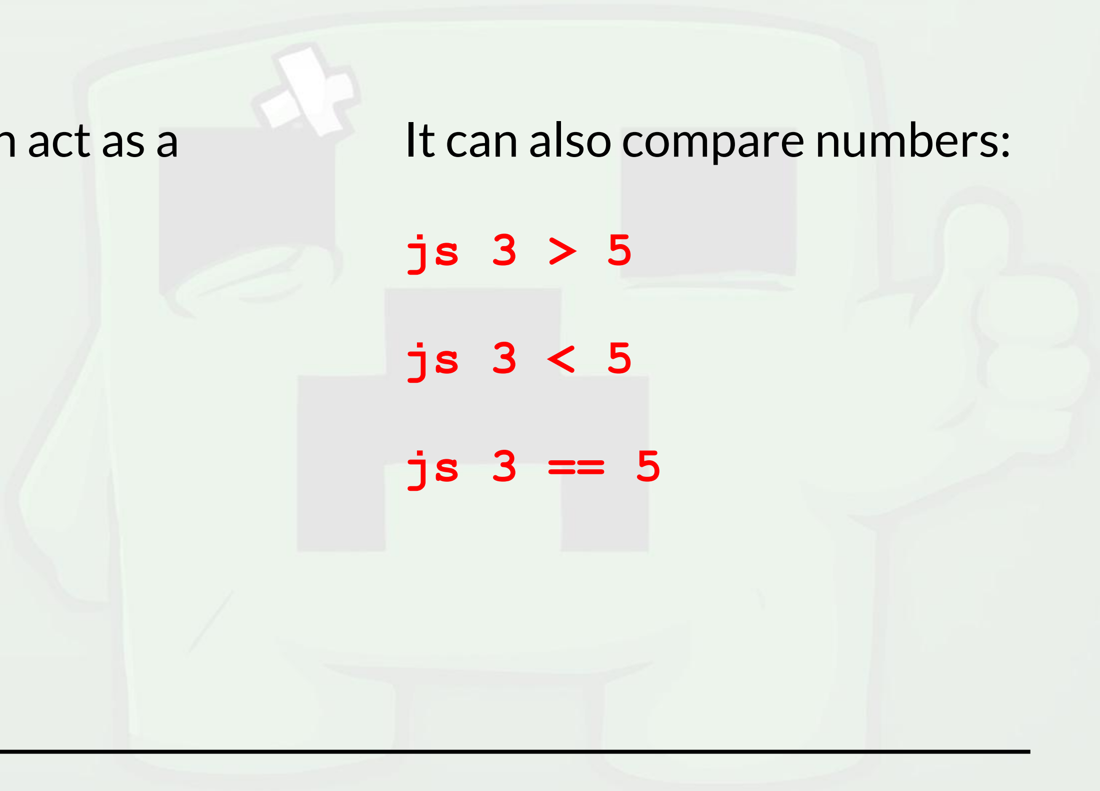
Running Commands



Console VS Client

- Javascript commands run on in-game (on the client) must always start with a / (to open the chat window)
- Console commands don’t need the /
- Not every command works in both

Basic Math



Javascript can act as a calculator:

```
js 2 + 3

js 2 * 3

js 2 - 3
```

It can also compare numbers:

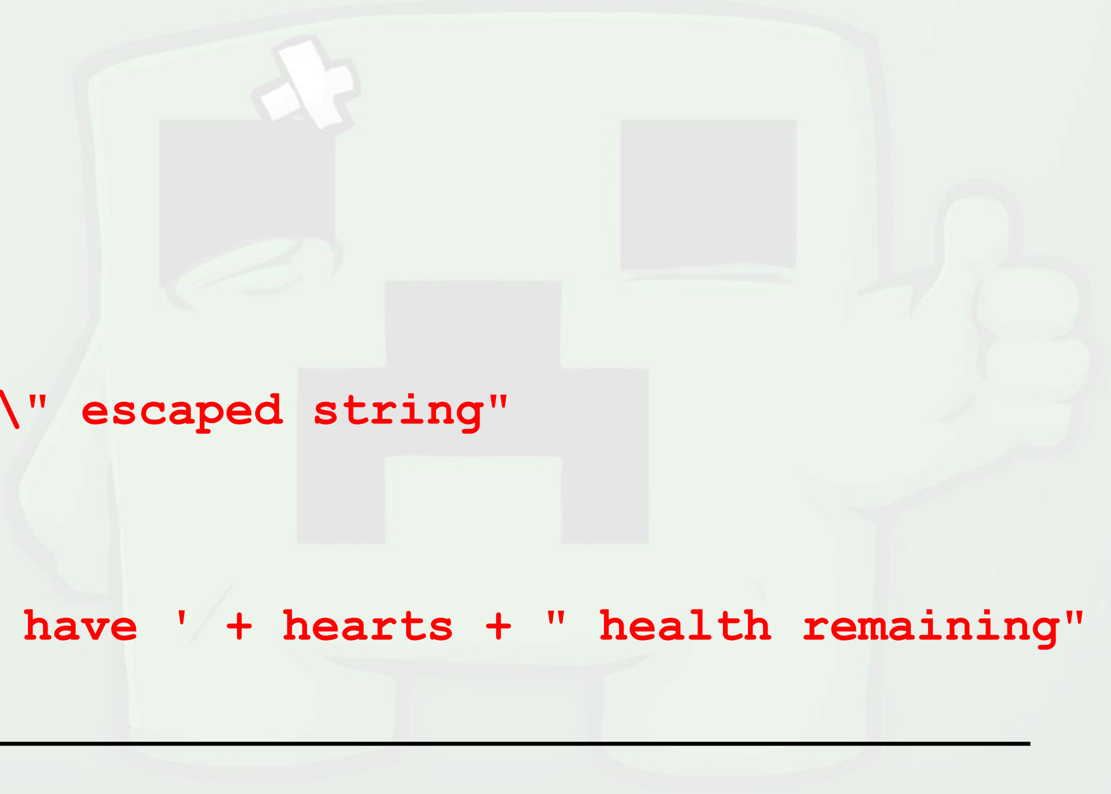
```
js 3 > 5

js 3 < 5

js 3 == 5
```

Storing Data in Variables

Start with a variable:	`js var hearts`
Set it to a value:	`js hearts = 8`
Check the current value:	`js hearts`
Change the value:	`js hearts = 9`
Do math with it:	`js hearts + 5`
	`js hearts - 2`
	`js hearts * 1`
	`js hearts / 3`

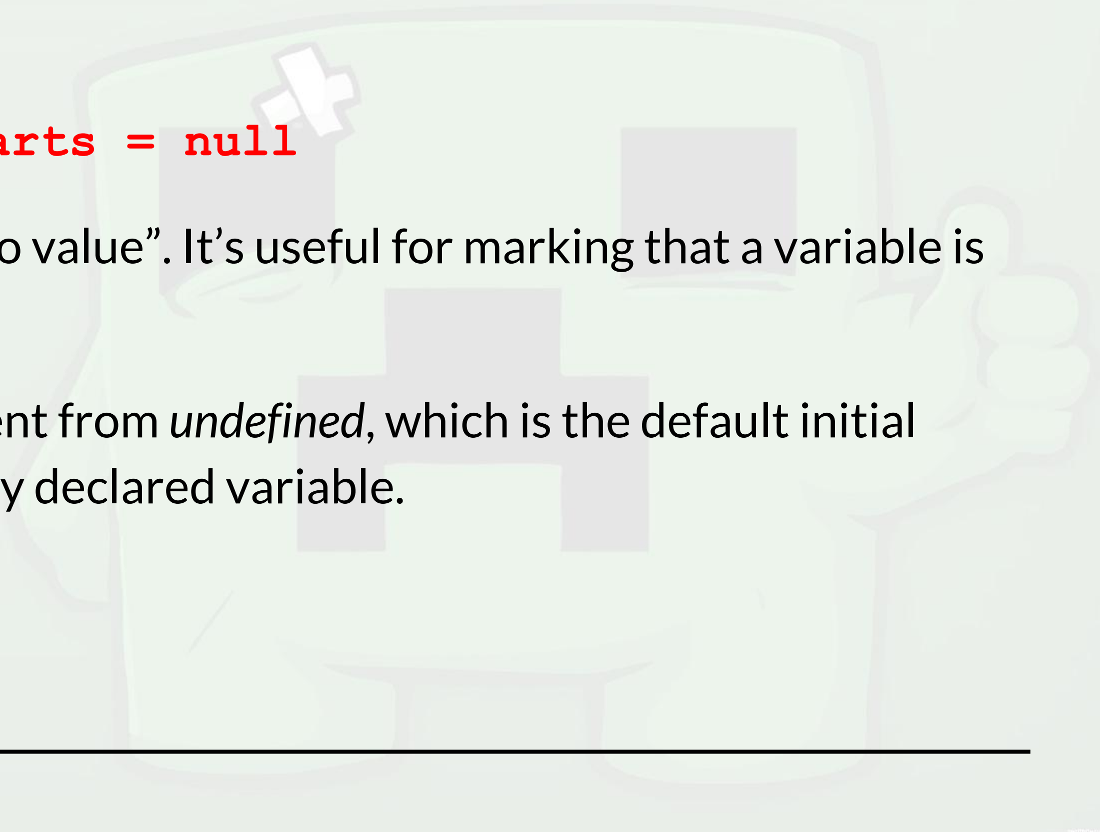
NOTE: variables can’t begin with numbers

Strings



```
js "double string"

js 'single string'

js 'I\'m an escaped string'

js "Here’s a \"double-quote\" escaped string"

js "I'm un-escaped"

js var healthMessage = 'You have ' + hearts + " health remaining"
```

The null Keyword

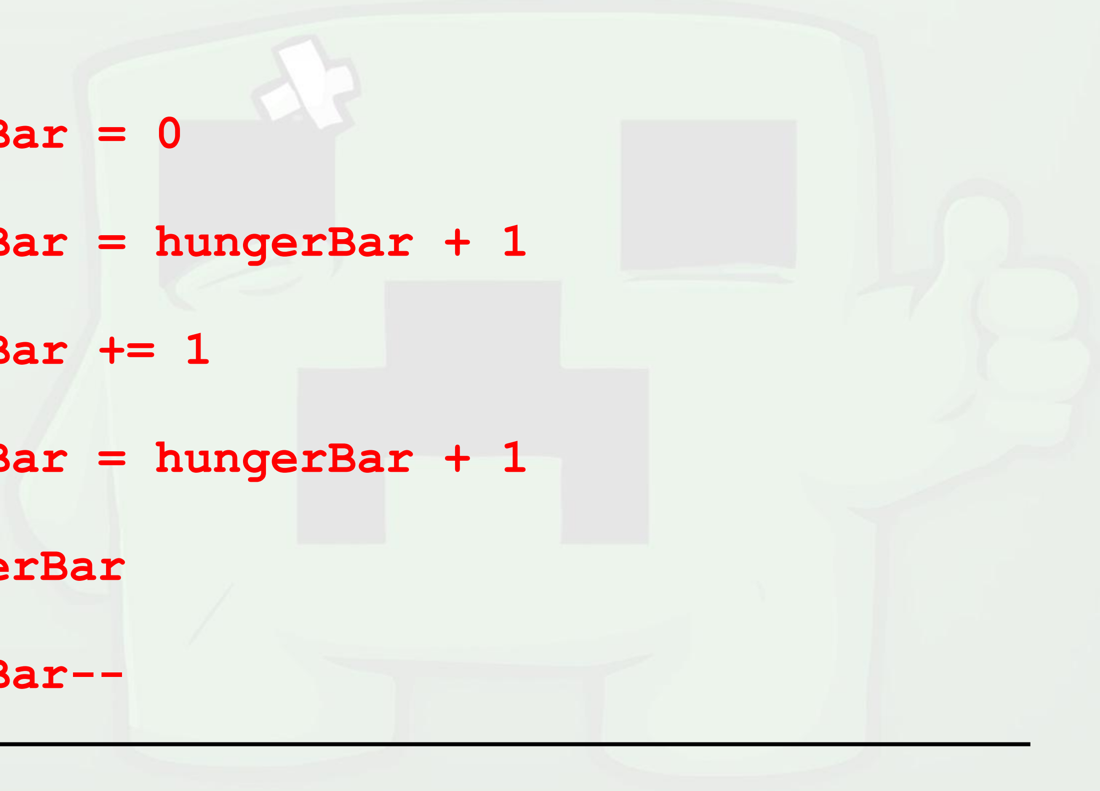


```
js var hearts = null
```

null means “no value”. It’s useful for marking that a variable is empty.

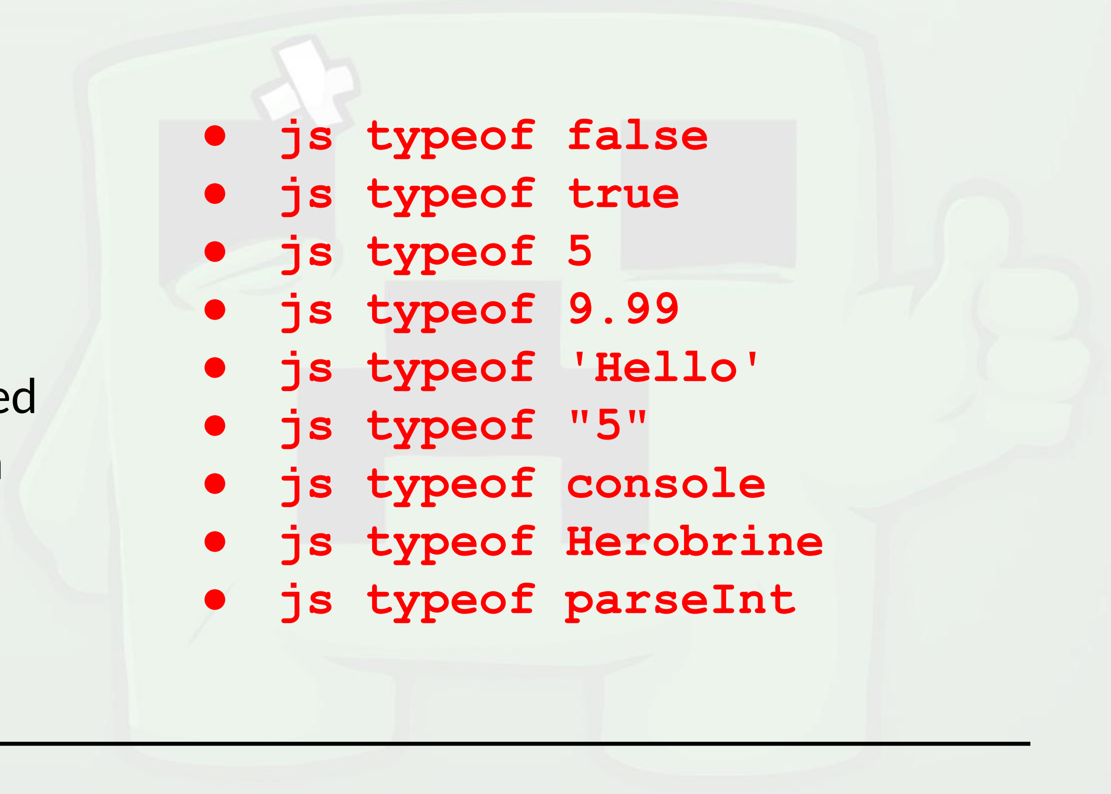
This is different from *undefined*, which is the default initial setting for any declared variable.

Adding and Subtracting



```
js hungerBar = 0

js hungerBar = hungerBar + 1

js hungerBar += 1

js hungerBar = hungerBar + 1

js ++hungerBar

js hungerBar--
```

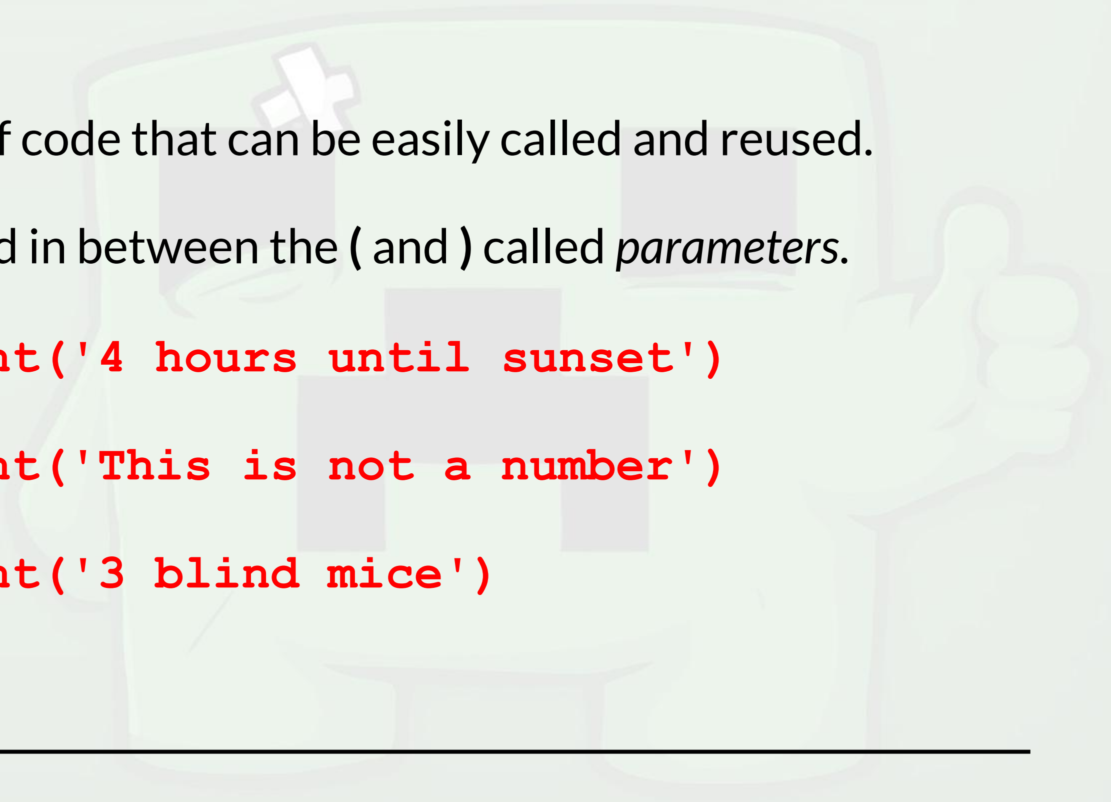
Data Types



- Number
- String
- Boolean
- Object
- Undefined
- Function

- `js typeof false`
- `js typeof true`
- `js typeof 5`
- `js typeof 9.99`
- `js typeof 'Hello'`
- `js typeof "5"`
- `js typeof console`
- `js typeof Herobrine`
- `js typeof parseInt`

Functions

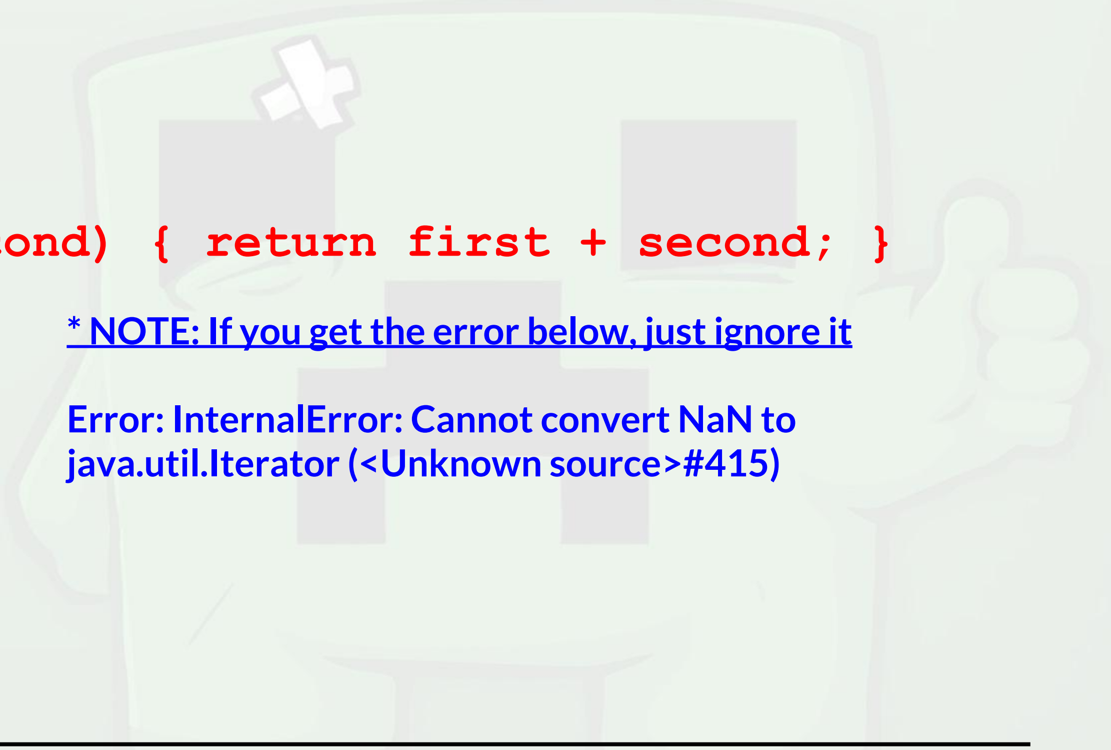


Collections of code that can be easily called and reused.

Values passed in between the **(** and **)** called *parameters*.

```
js parseInt('4 hours until sunset')

js parseInt('This is not a number')

js parseInt('3 blind mice')
```

Writing Your Own Functions



Type the following on one line *:

```
js function add(first, second) { return first + second; }
```

Call your new function:

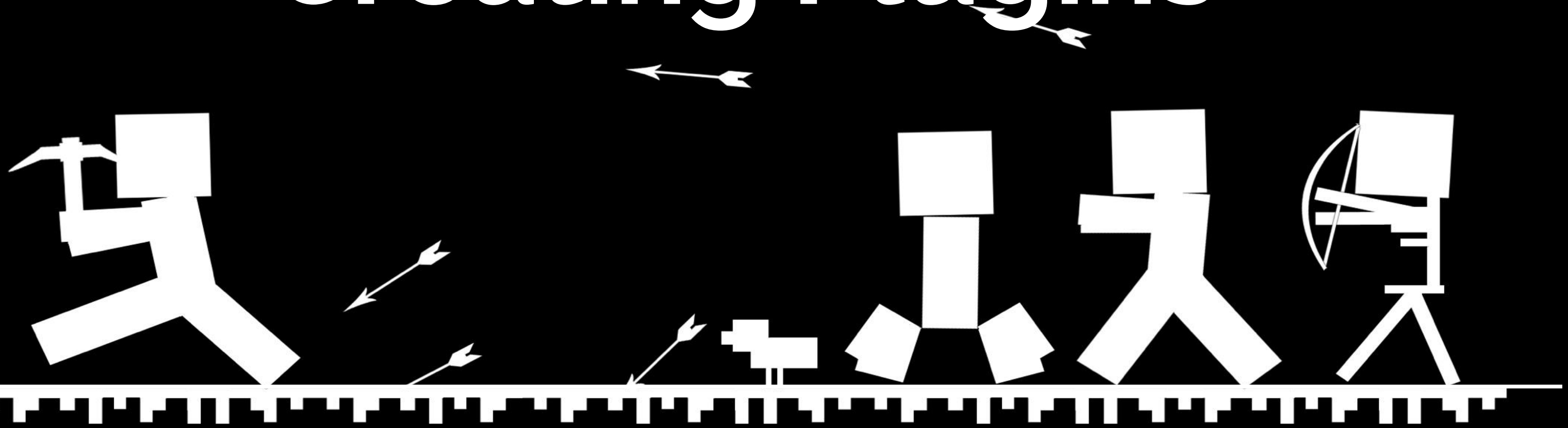
```
js add(5, 6)
```

```
js add(9, 1)
```

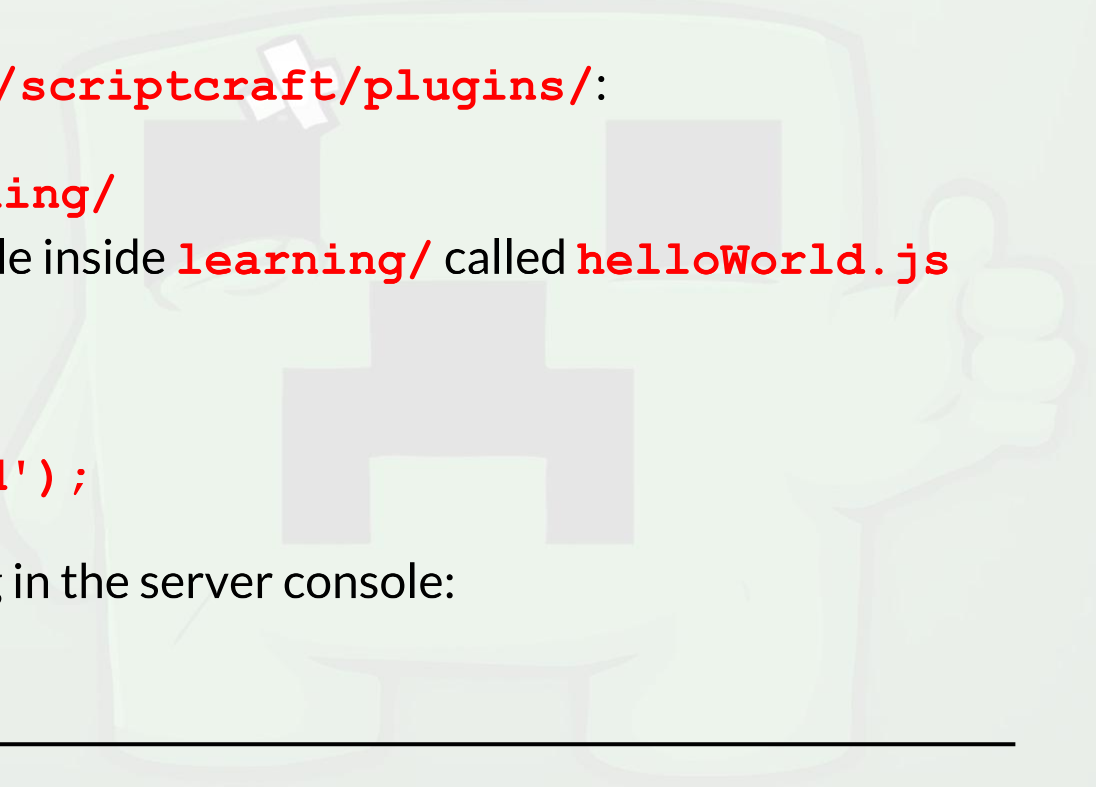
* NOTE: If you get the error below, just ignore it

Error: InternalError: Cannot convert NaN to java.util.Iterator (<Unknown source>#415)

Creating Plugins



Your First Minecraft Plugin



In `ScriptCraft-master/server/scriptcraft/plugins/`:

- create a new folder called `learning/`
- use your text editor to create a file inside `learning/` called `helloWorld.js`

Add the following inside your file:

```
console.log('Hello World');
```

Save your file, then type the following in the server console:

```
js refresh()
```

Making Your Code Reusable

Let’s put our helloWorld.js code into a function:

```
function helloWorld() {
    console.log('Hello World');
}
```

And refresh our server:

```
js refresh()
```

What Happened to Our Message?

Add the new code and refresh:

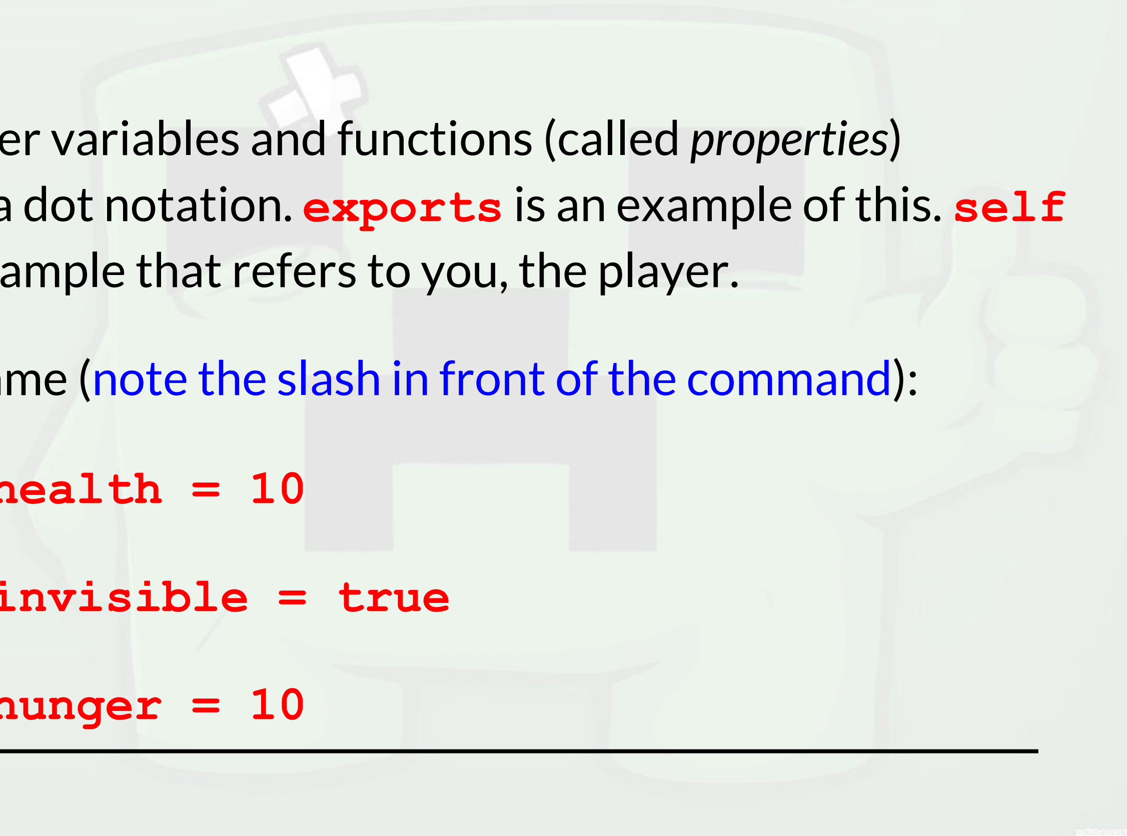
```
function helloWorld() {
    console.log('Hello World');
}
helloWorld();
```

Making helloWorld() public

To call functions directly, we must first export them:

```
function helloWorld() {
    console.log('Hello World');
}
helloWorld();
exports.helloWorld = helloWorld;
```

Objects

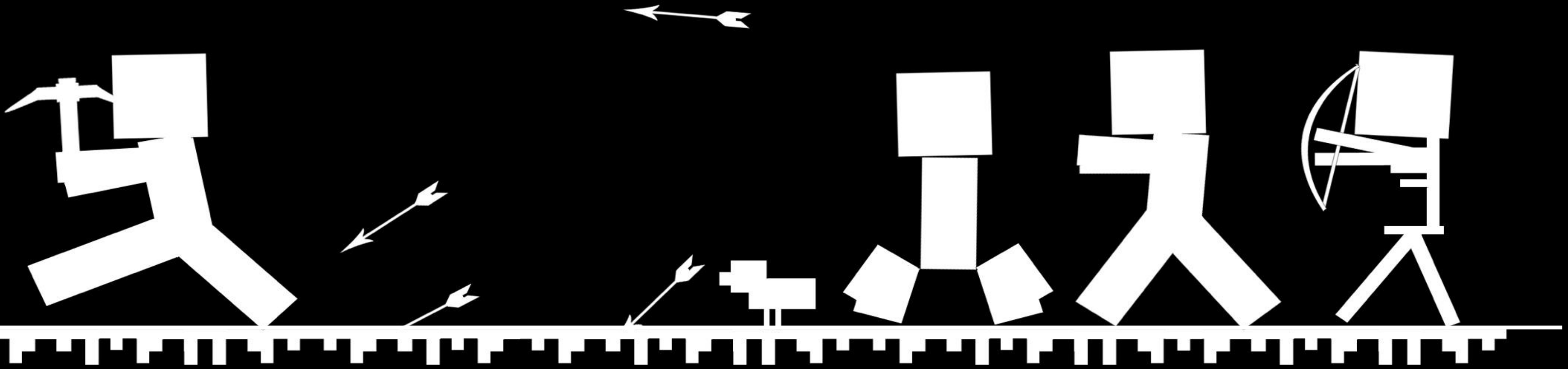


Can hold other variables and functions (called *properties*) accessible via dot notation. `exports` is an example of this. `self` is another example that refers to you, the player.

Try this in-game (note the slash in front of the command):

```
/js self.health = 10

/js self.invisible = true

/js self.hunger = 10
```

Making A Dice Plugin



Rolling Dice

In `ScriptCraft-master/server/scriptcraft/`

- create a new file in `plugins/` called `dice.js`
- add the code on the right
- save and refresh

```
function roll(){

    var result = Math.random();

    result = result * 6;

    result = Math.floor(result);

    return result;

}

exports.roll = roll;
```

Multi-sided Die

You *could* define multiple functions for different dice sides (e.g. `rollSixSides()`, `rollFourSides()`, etc...), but that gets tiring.

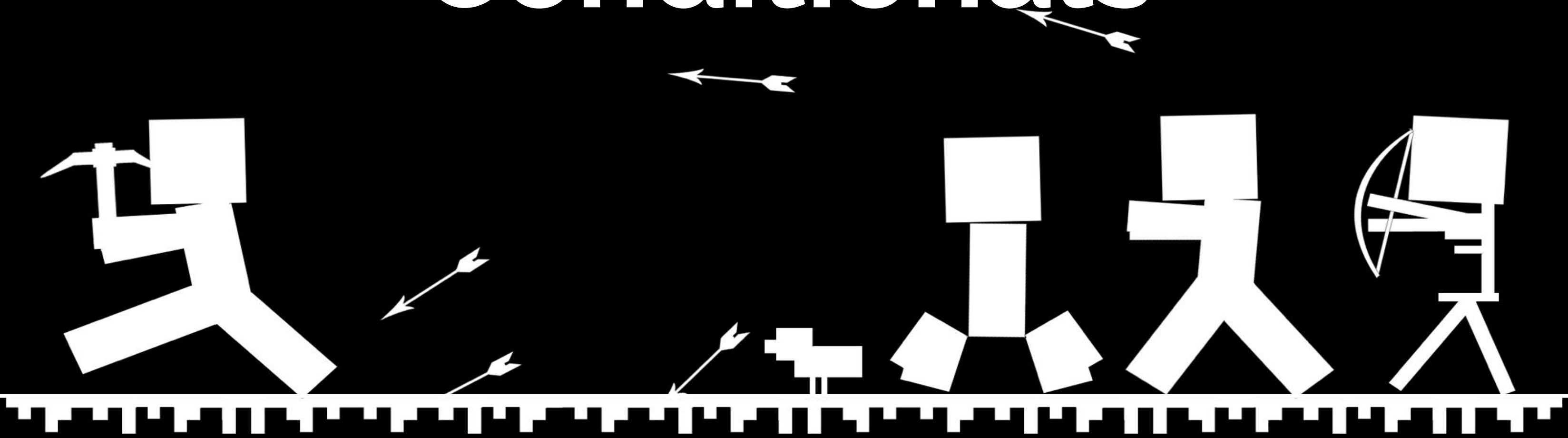
It's easier to pass the number of sides as a **parameter**.

Edit `dice.js` to add the changes on the ***right***, refresh, and try it out using the code ***below***:

```
js var dice = require("dice")

js dice.roll(6)

js dice.roll(20)
```

```
function roll( sides ){

    var result = Math.random();

    result = result * sides;

    result = Math.floor(result);

    return result;

}

exports.roll = roll;
```

Conditionals



If/Else/Else If

Useful for changing code based on different events:

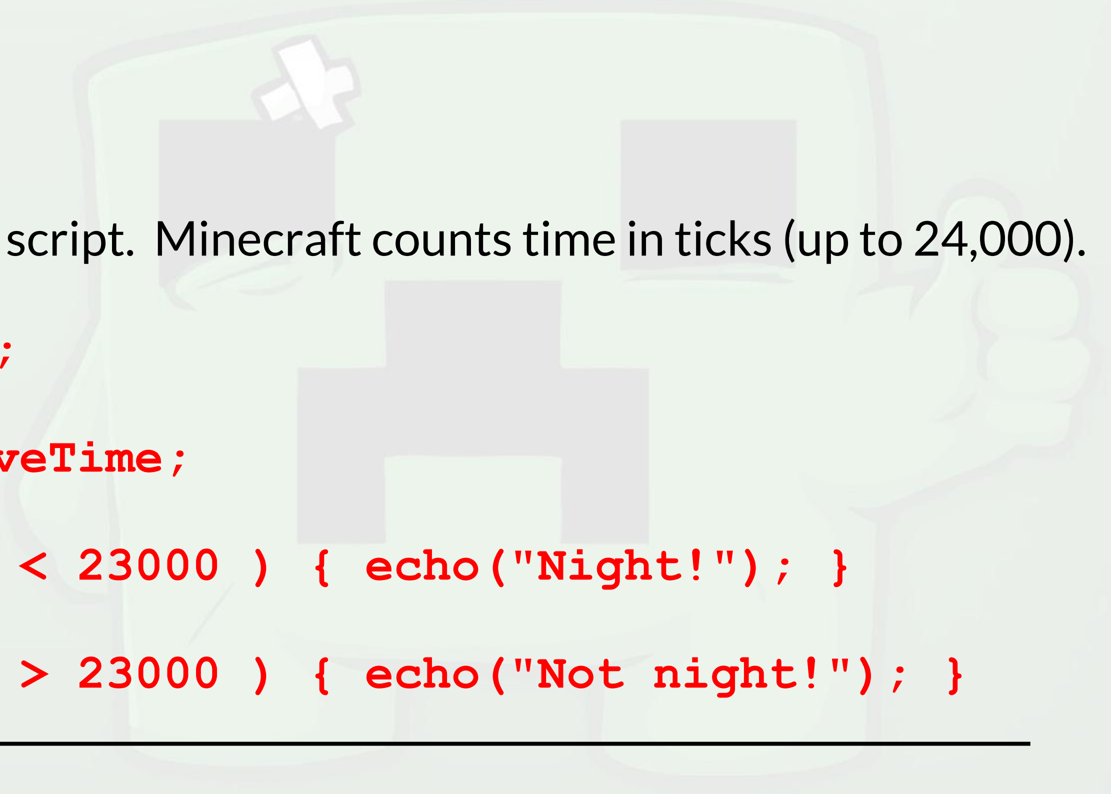
```
var time = "noon";

if ( time == "morning" ) { echo("Time for breakfast!"); }

else if (time == "noon") { echo("Time for lunch!"); }

else if (time == "night") { echo("Time for dinner!"); }

else { echo(“Time for snacks!”); }
```

Combining Conditionals

`&&` represents *and*. `||` represents *or*.

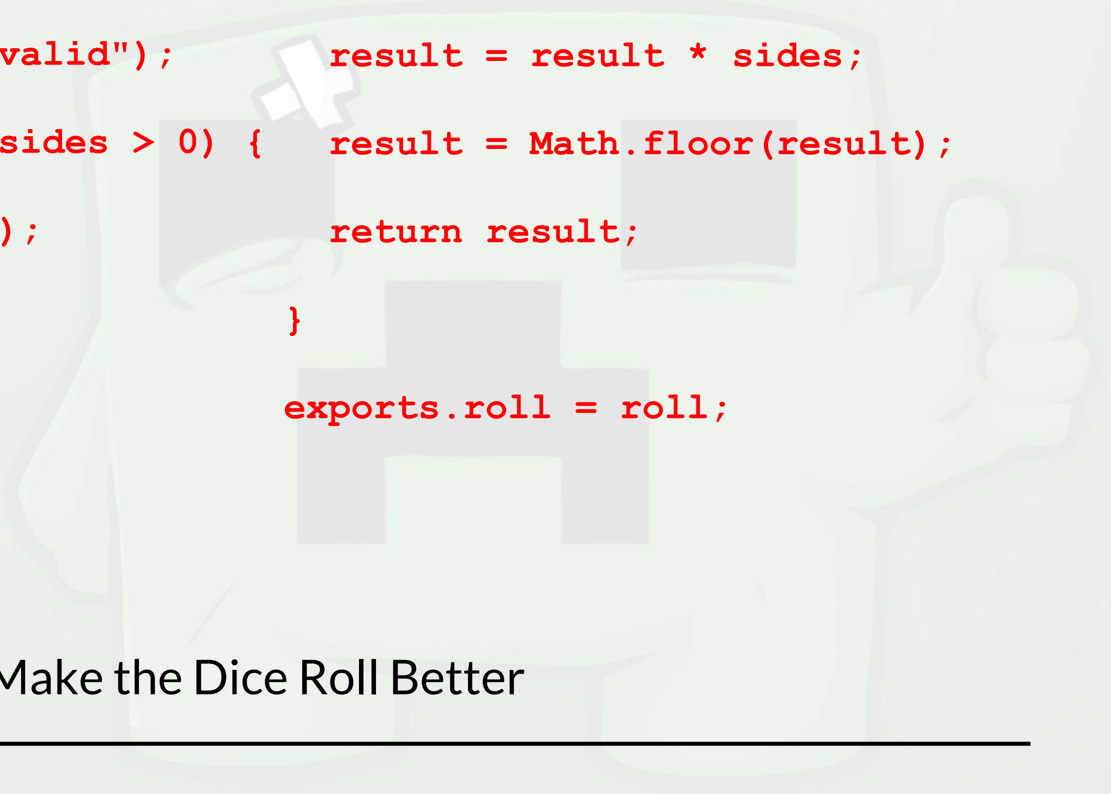
Below is a more accurate time check script. Minecraft counts time in ticks (up to 24,000).

```
/js var world = self.world;

/js var now = world.relativeTime;

/js if (now > 13000 && now < 23000 ) { echo("Night!"); }

/js if (now < 13000 || now > 23000 ) { echo("Not night!"); }
```



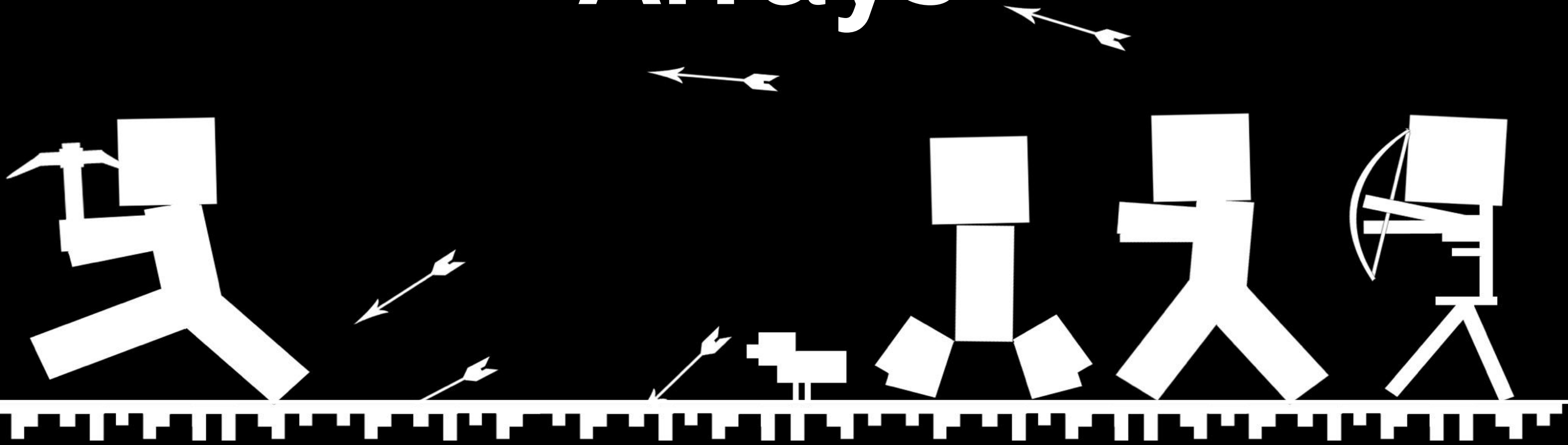
```
function roll( sides ){
  if ( isFinite(sides) && sides < 0 ) {
    throw("Negative numbers not valid");
  } else if (isFinite(sides) && sides > 0) {
    return rollValidNumber(sides);
  } else {
    throw("Not a number");
  }
}
```

```
function rollValidNumber( sides ){
  var result = Math.random();
  result = result * sides;
  result = Math.floor(result);
  return result;
}
exports.roll = roll;
```

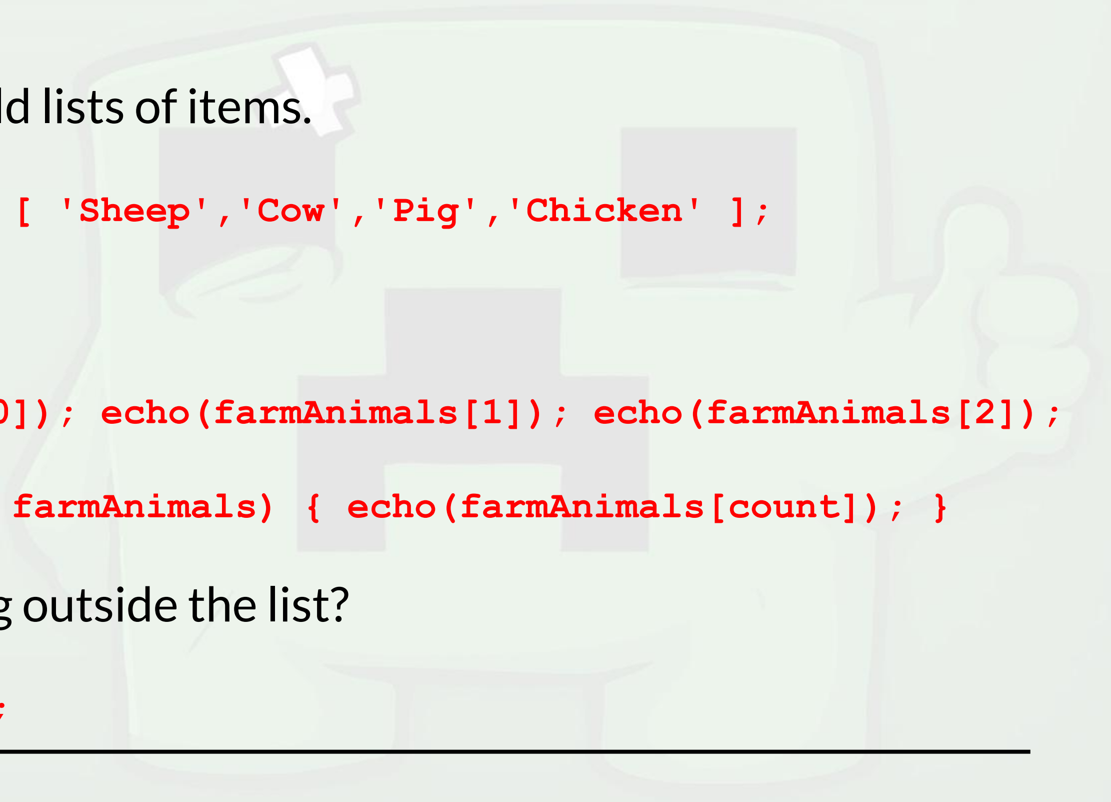


Example of Using Conditionals to Make the Dice Roll Better

Arrays



Using Arrays



Arrays are objects than hold lists of items.

```
js var farmAnimals = [ 'Sheep','Cow','Pig','Chicken' ];
```

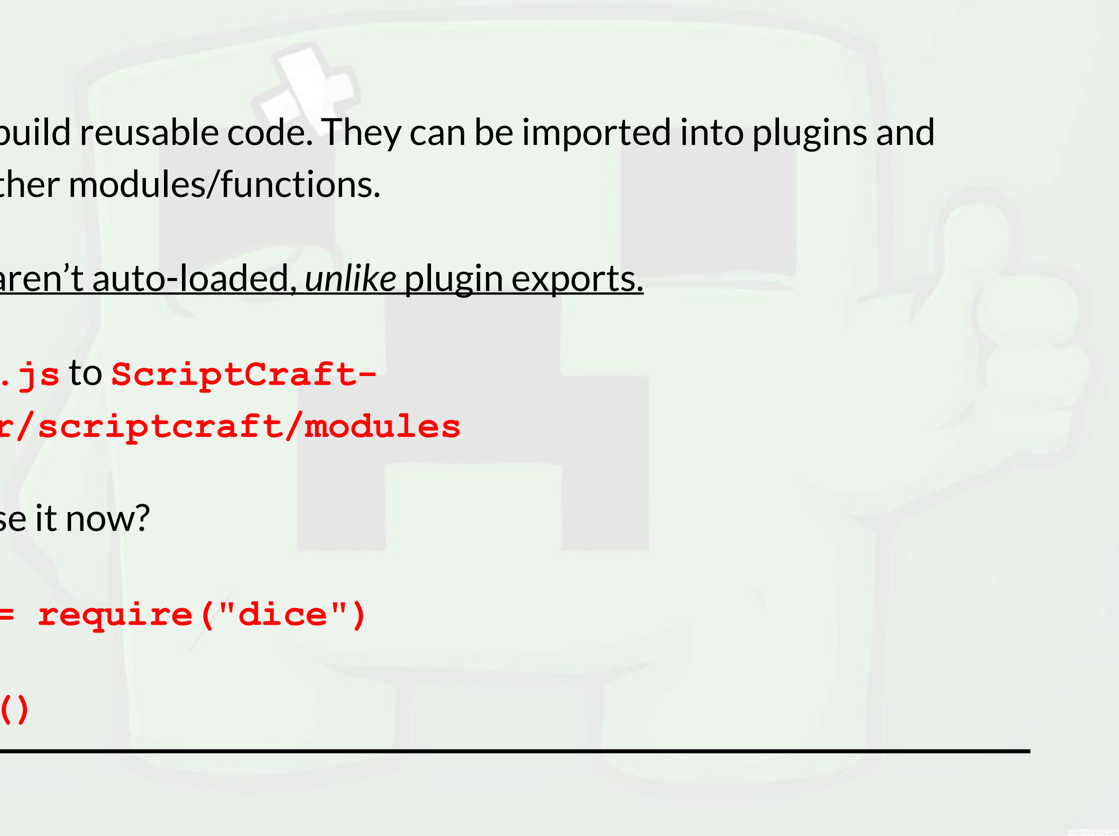
How do we access the list?

```
js echo(farmAnimals[0]); echo(farmAnimals[1]); echo(farmAnimals[2]);

js for (var count in farmAnimals) { echo(farmAnimals[count]); }
```

What if we print something outside the list?

```
echo(farmAnimals[5]);
```

Using Modules



Modules let you build reusable code. They can be imported into plugins and combined with other modules/functions.

<u>Module exports aren't auto-loaded, *unlike* plugin exports.</u>

Let's move `dice.js` to `ScriptCraft-master/server/scriptcraft/modules`

But how do we use it now?

```
js var dice = require("dice")
```

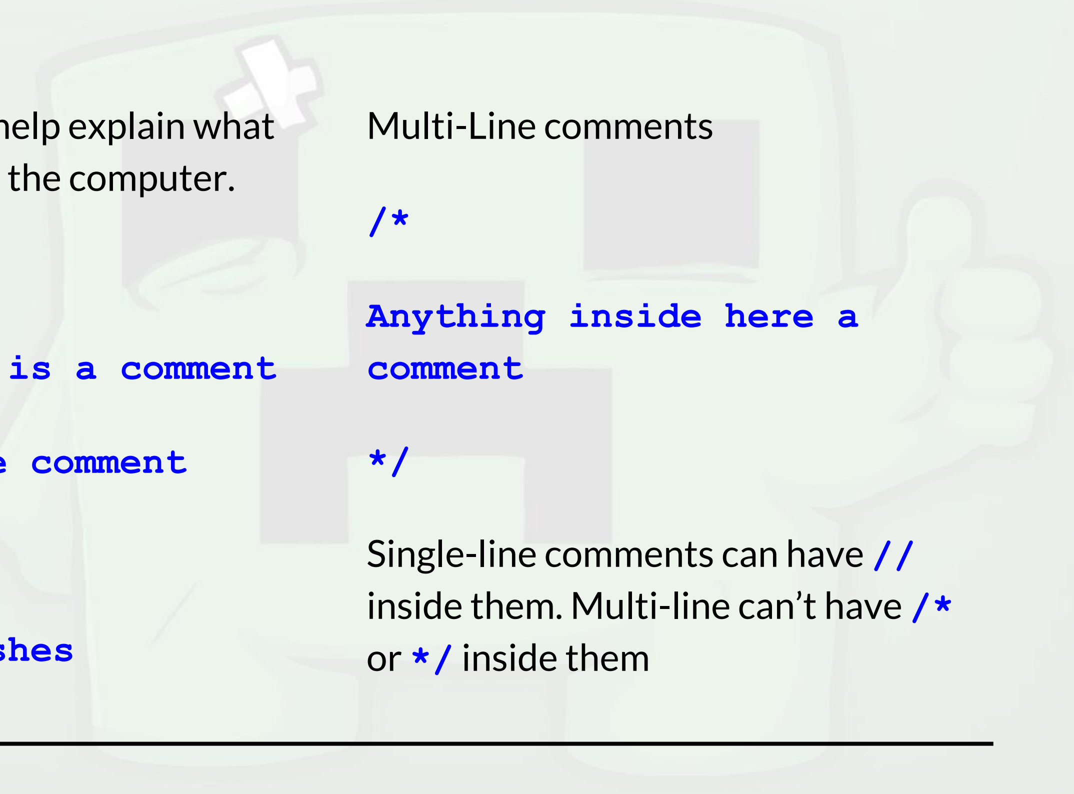
```
js dice.roll()
```

Random Spawner Plugin

Create a file called `randomSpawner.js` in your `plugins/` folder:

```
var dice = require('dice');
var spawn = require('spawn');
var farmAnimals = ["cow", "chicken", "pig", "sheep"];
var total = farmAnimals.length; // .length gets # of elements

function randomSpawn() {
  var result = dice.roll(total);
  spawn(farmAnimals[result], self.location);
}

/* This command will only work in-game */
exports.randomSpawn = randomSpawn;
```

A Quick Note About Comments



Comments are notes in code that help explain what is happening. They are not read by the computer.

Single-Line comments

```
// anything after the // is a comment

var a = 1; // end of line comment

// multiple lines ...

// ... need multiple slashes
```

Multi-Line comments

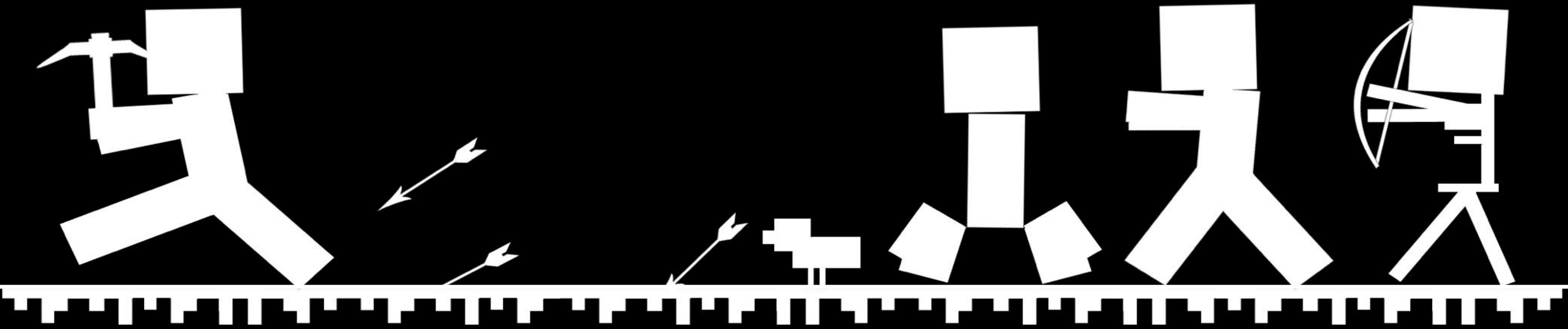
```
/*

Anything inside here a
comment

*/
```

Single-line comments can have `//` inside them. Multi-line can't have `/*` or `*/` inside them

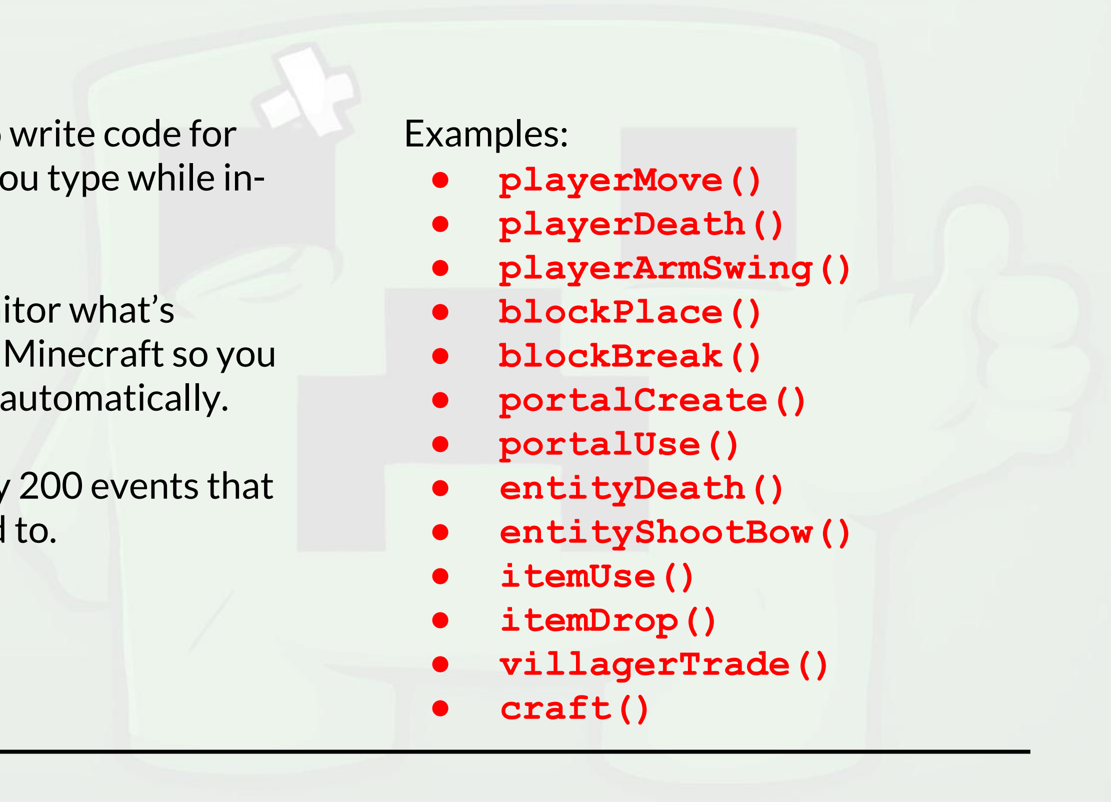
Even More Randomness

```
var dice = require('dice');
var spawn = require('spawn');
var entities = require('entities');
var entityNames = []; // empty array
/* push object properties into array to get total */
for (var name in entities){ entityNames.push(name); }
var total = entityNames.length;

function randomSpawn() {
  var result = dice.roll(total);
  echo("Spawning "+entityNames[result]); //show what spawned
  spawn(entityNames[result], self.location);
}

exports.randomSpawn = randomSpawn;
```

Event-Driven Programming



Why Events?



You know how to write code for commands that you type while in-game.

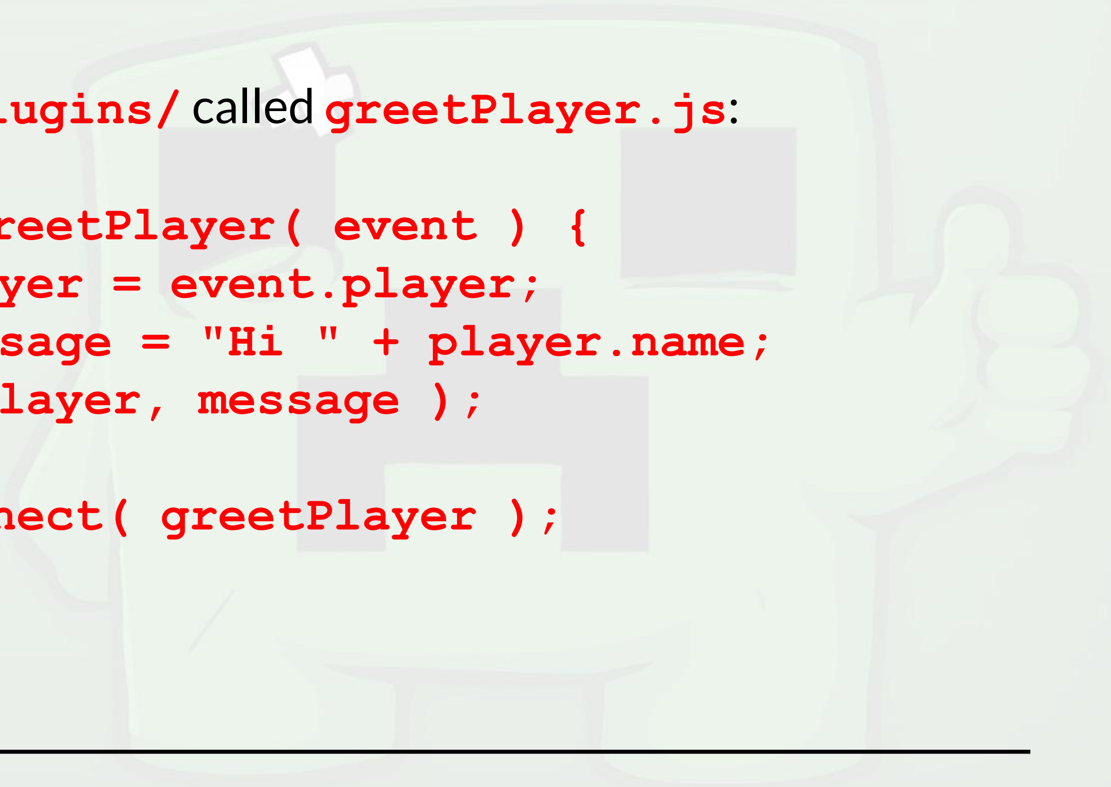
You can also monitor what's happening inside Minecraft so you can respond to it automatically.

There are roughly 200 events that can be responded to.

Examples:

- `playerMove()`
- `playerDeath()`
- `playerArmSwing()`
- `blockPlace()`
- `blockBreak()`
- `portalCreate()`
- `portalUse()`
- `entityDeath()`
- `entityShootBow()`
- `itemUse()`
- `itemDrop()`
- `villagerTrade()`
- `craft()`

Greet Players On Server Join



Add a file in `plugins/` called `greetPlayer.js`:

```
function greetPlayer( event ) {
    var player = event.player;
    var message = "Hi " + player.name;
    echo( player, message );
};
events.connect( greetPlayer );
```

Stay Tuned for Part 2!

To be continued...