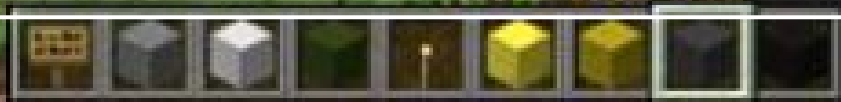


---

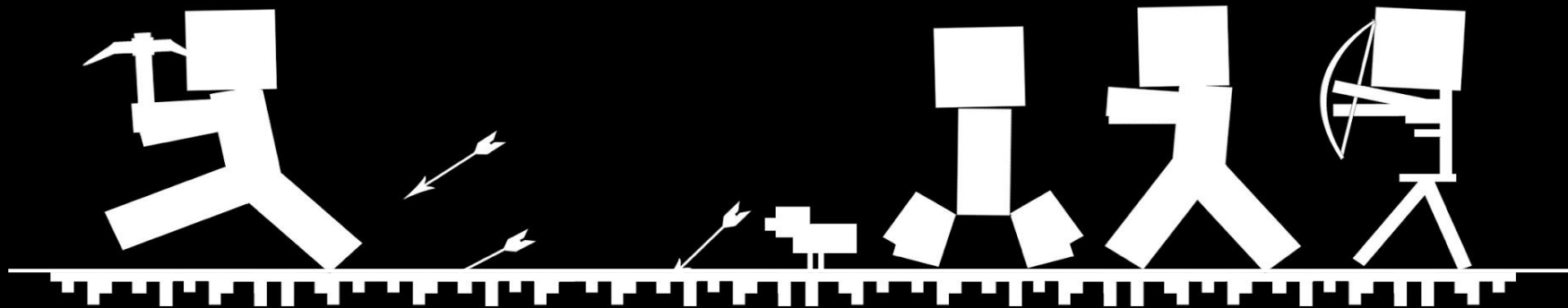
# Writing Plugins in Minecraft with JavaScript

Getting Started



---

# Getting Started



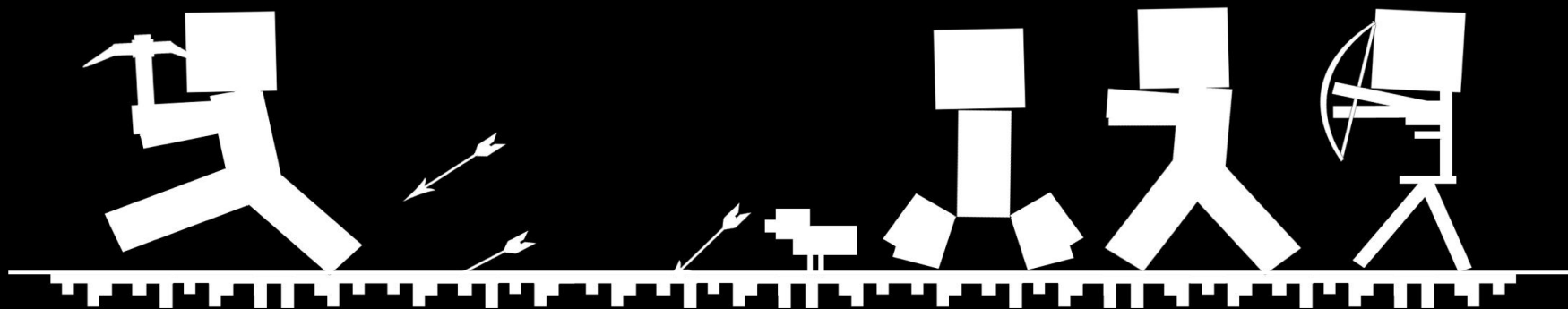
---

# What You'll Need

1. Minecraft (*installed and running*)
  2. Oracle Java 8 (*even if you already have Java, you may need to upgrade your version*)
  3. Class source code:  
<https://github.com/LetsCodeBlacksburg/ScriptCraft-Spigot>  
(*download this to your desktop and unzip it*)
  4. Text editor (*Sublime Text is recommended*)
-

---

# The Spigot Server



---

# About

- <http://spigotmc.org>
- One of many custom servers that expose APIs
- Related examples: CraftBukkit, Glowstone, CanaryMod, etc...
- Many use the popular Bukkit API (including Spigot)
- Written in statically-typed Java language
- Java used in enterprise software development and web (decreasingly)



---

# Find the startup script for your OS

In **Scriptcraft-Spigot-master/**:

- Windows: **Windows/start.bat**
- Mac: **Mac/start.command**
- Linux: **Linux/start.sh**

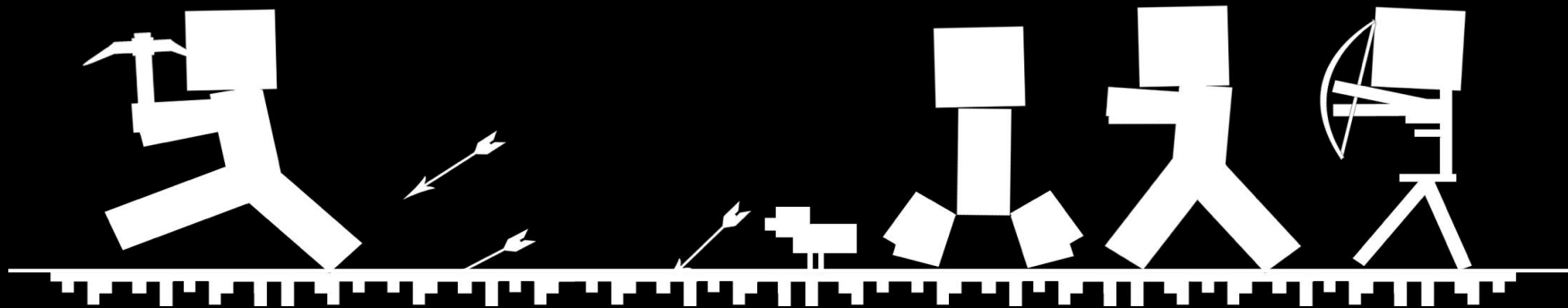
Drag this file into the **server/** directory

Start your server by double-clicking on the file you just moved

---

---

# The ScriptCraft Mod



---

# About

- <http://scriptcraftjs.org>
- Plugin for CanaryMod and Bukkit API servers
- Implements dynamically-typed JavaScript
- JavaScript used in client-side web development and server-side scripting





---

# Verifying ScriptCraft is Installed

During startup, the server console will print the following:

```
19:49:16 [INFO] [scriptcraft] Enabling scriptcraft v3.2.0-2016-03-19
```

You can also verify everything works by typing the following command exactly into the server console:

```
js "Hello world"
```

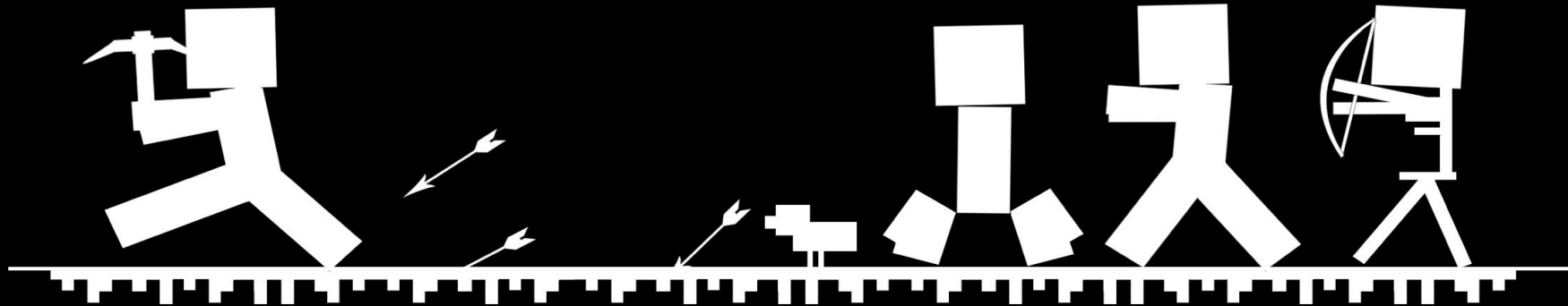
You should get something similar to the following response:

```
[22:10:23 INFO]: "Hello world"
```

---

---

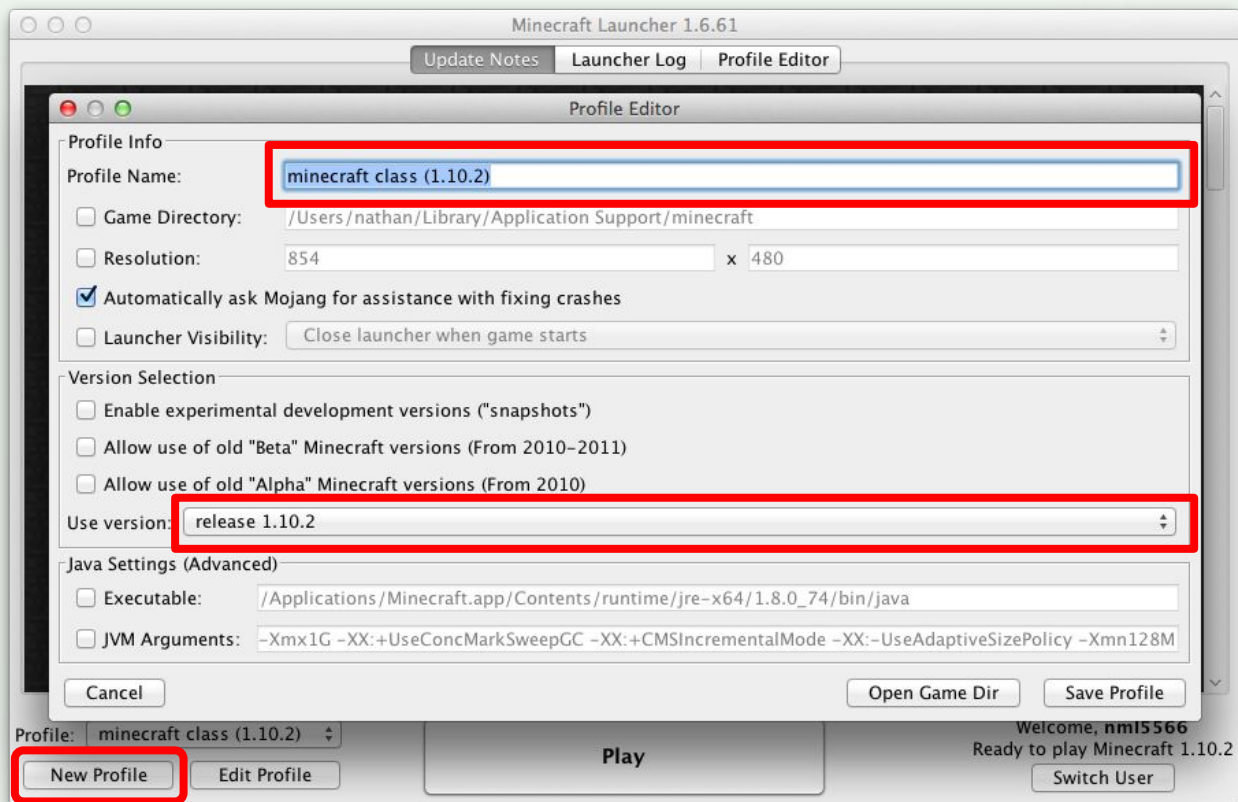
# Connecting to your Server



# Minecraft Profile Editor

Click on the New Profile button

Create a profile that uses release version 1.10.2



## 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.

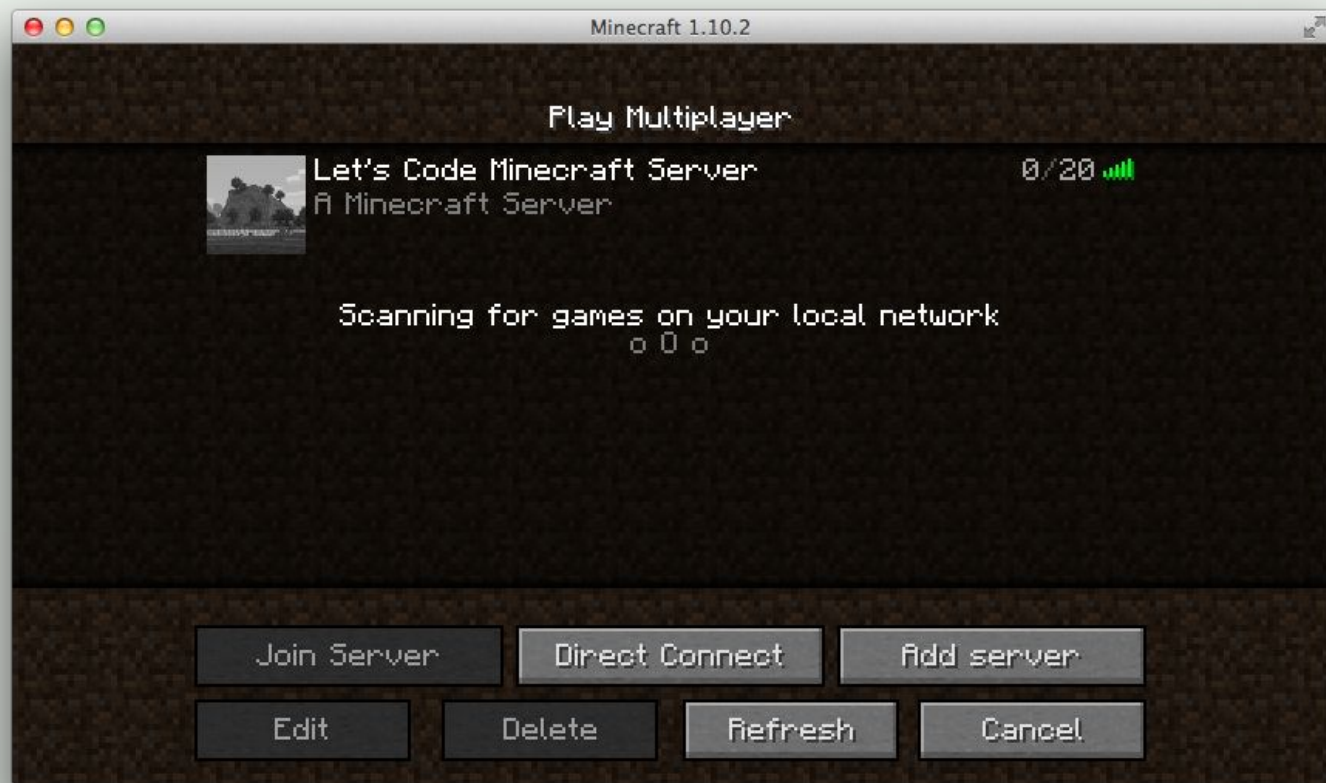
Type **localhost** in the *Server Address* field.



# —

## Joining Your Server

- Make sure your server is running
- Try clicking *Refresh* if nothing shows up.



---

# 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 your\_username**

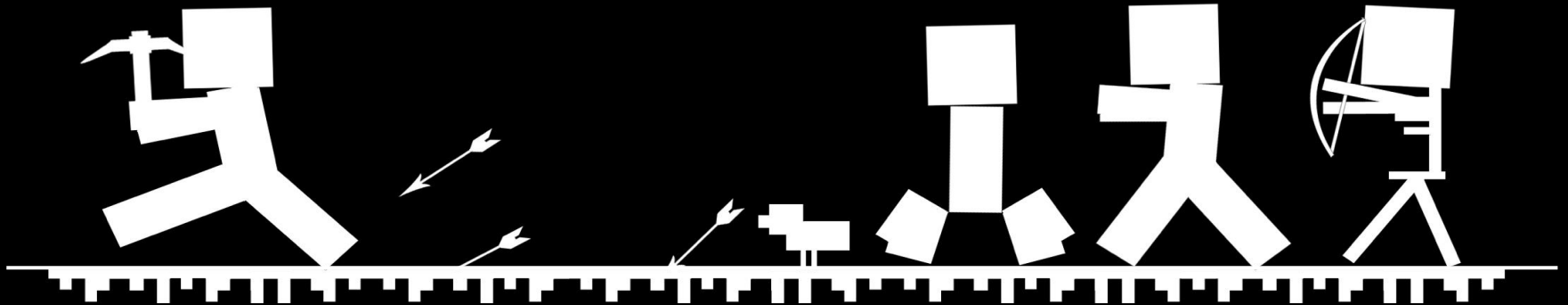
The server console will print the following:

**[22:20:23 INFO]: Opped your\_username**

---

---

# Exploring JavaScript in Minecraft





---

# Running Commands

## Console VS Client

- Javascript commands run 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, and they can sometimes have different effects

---

# 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'
```

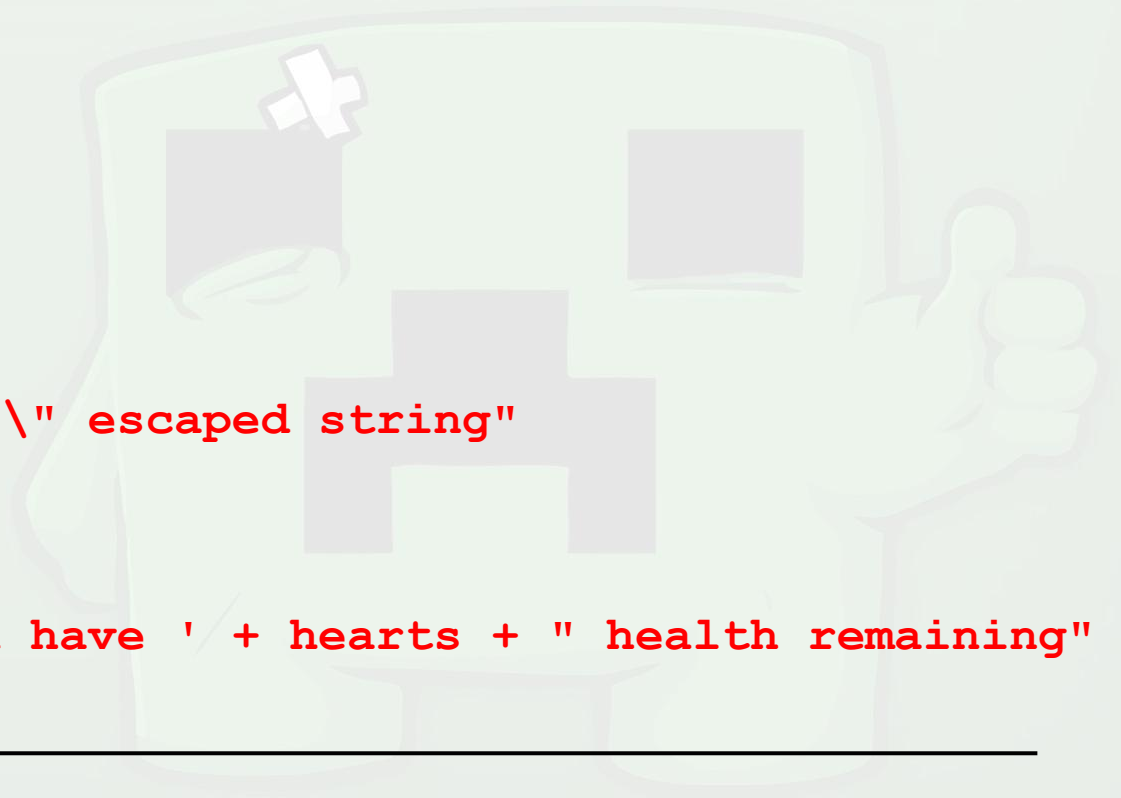
```
js 'I\'m an escaped string'
```

```
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
```

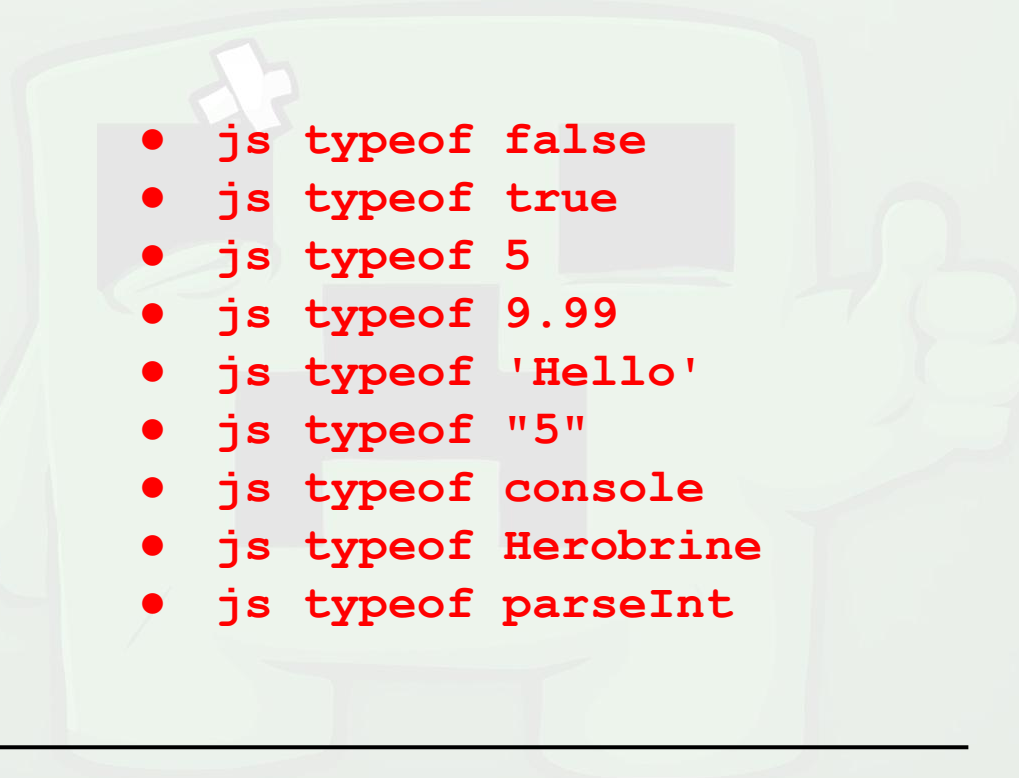
```
js hungerBar--
```

---

---

# 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; }
```

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

Call your new function:

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

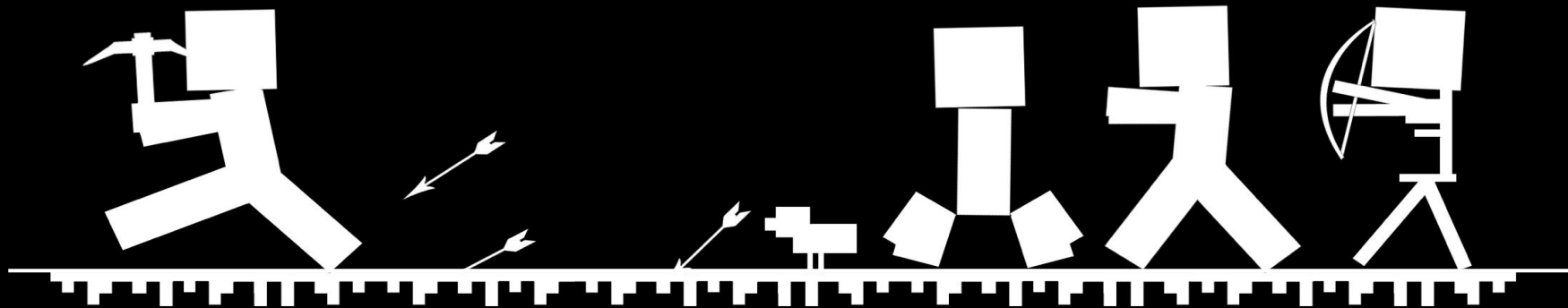
```
js add(9, 1)
```

---

22:32:52 [SEVERE] [scriptcraft] Error while  
trying to display result:

---

# Creating Plugins



---

# Your First Minecraft Plugin

In `Scriptcraft-Spigot-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.setHealth(10)
```

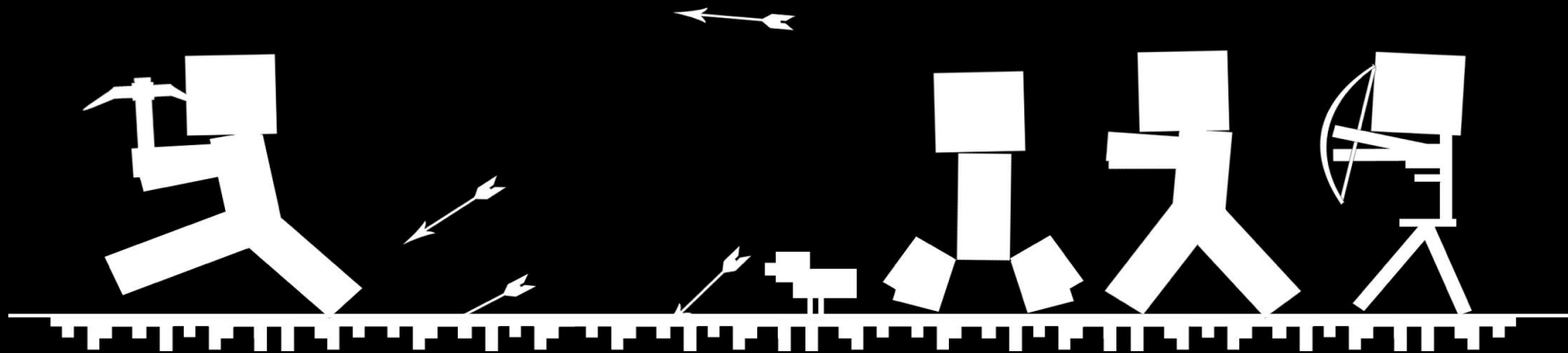
```
/js self.setFoodLevel(10)
```

```
/js self.setExp(0.5)
```

---

---

# Making A Dice Plugin





---

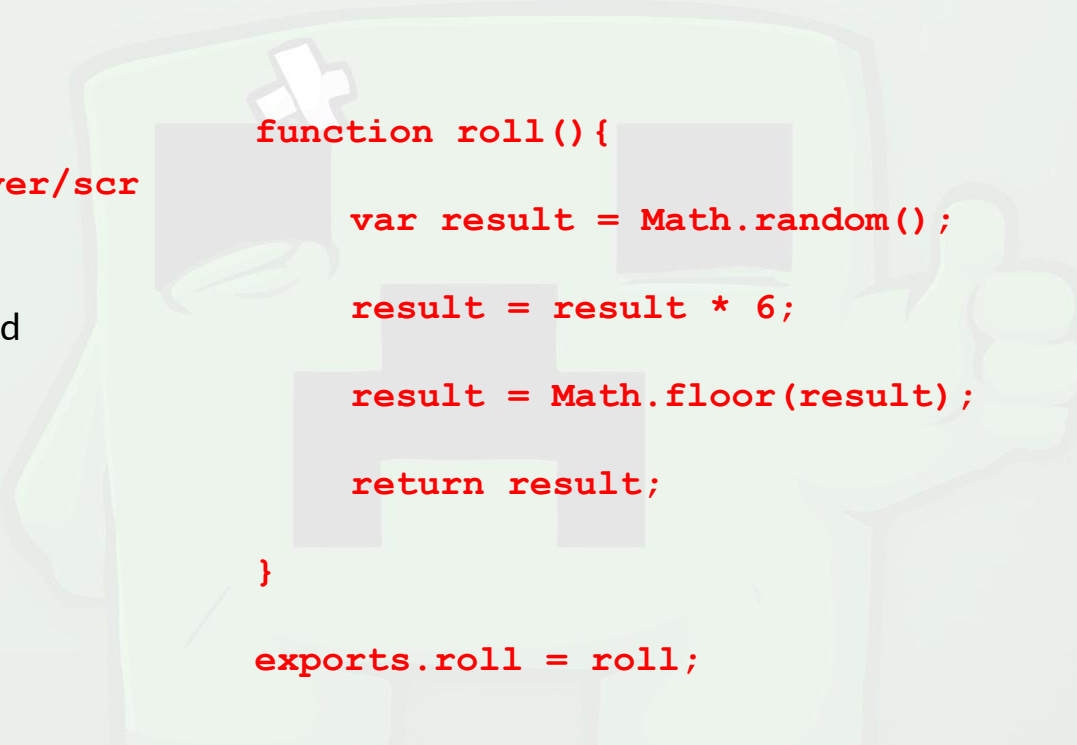
# Rolling Dice

In

Scriptcraft-Spigot-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*:

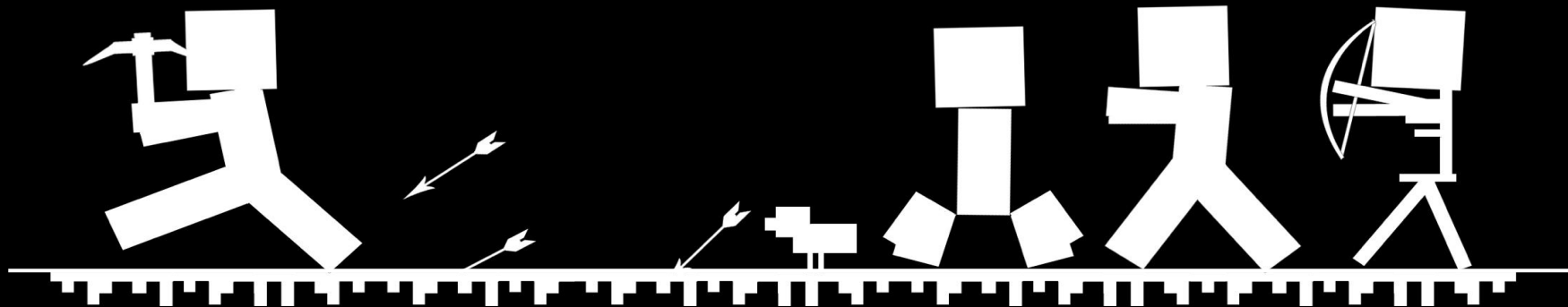
```
js roll(6)
```

```
js 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.getTime();
```

```
/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 ) {
        echo("Negative numbers not valid");
    } else if (isFinite(sides) && sides > 0) {
        return rollValidNumber(sides);
    } else {
        echo("Not a number");
    }
}

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

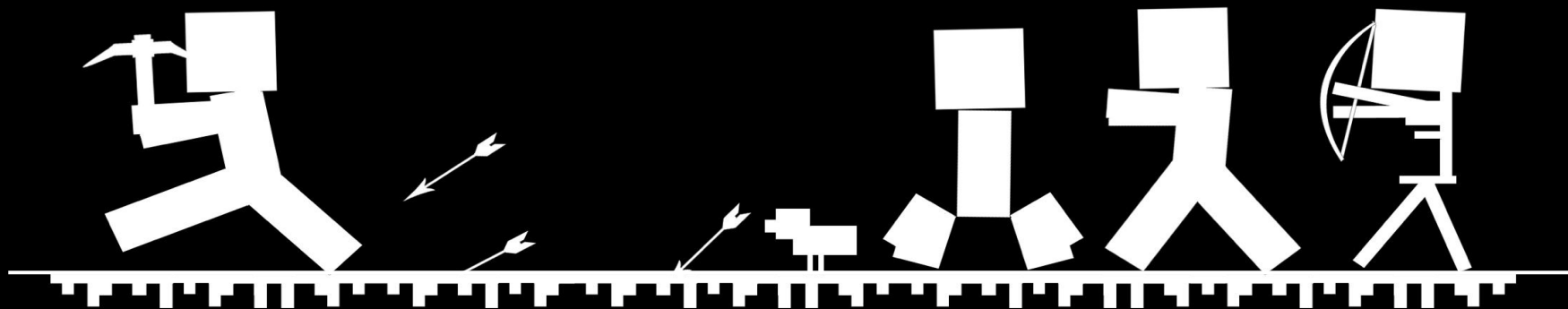
exports.roll = roll;
```

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?

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

---



---

# Using Modules

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

Module exports aren't auto-loaded, *unlike* plugin exports.

Let's move **dice.js** to

**Scriptcraft-Spigot-master/server/scriptcraft/modules**

But how do we use it now?

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

```
js dice.roll(6)
```

---

---

# 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;

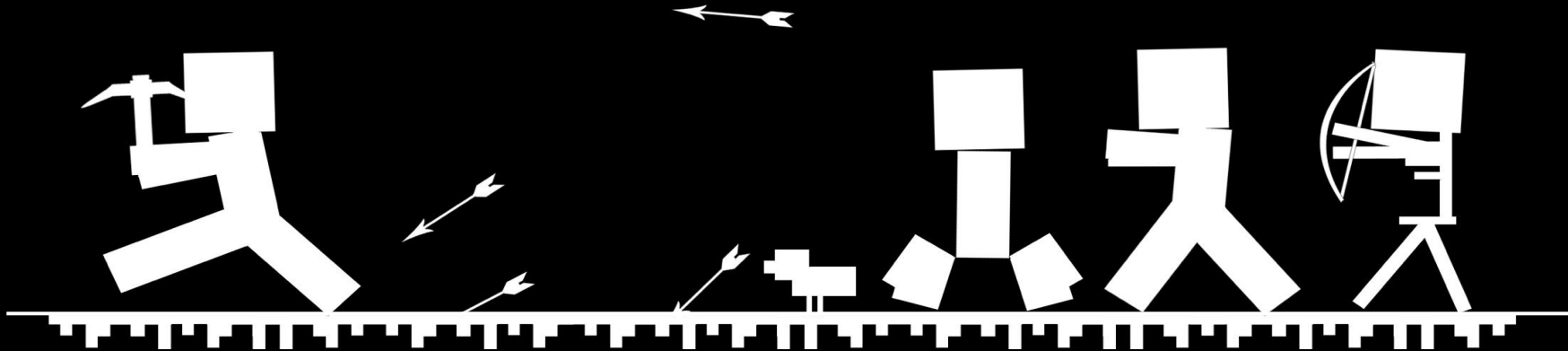
function randomSpawn() {
    var result = dice.roll(total);
    echo("Spawning "+entityNames[result]); //show what spawned
    spawn(entityNames[result], self.location);
}

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:

- Player movement
  - Player arm swing
  - Placing blocks
  - Breaking blocks
  - Creating portals
  - Using portals
  - Changing weather
  - Entity death
  - Entity shooting bow
  - Using items
  - Dropping items
  - Crafting items
  - Trading with villagers
-

---

# Greet Players On Server Join

Add a file in **plugins/** called **playerSneak.js**:

```
events.playerToggleSneak(function( event ) {  
    var player = event.player;  
    if (player.isSneaking())  
        echo( player, "STANDING TALL!" );  
    else  
        echo( player, "sneaky, sneaky..." );  
});
```

---

---

# Stay Tuned for Part 2!

To be continued...

