



## Learning Goals

- Basic Phaser development environment.
- Setting up a web server.

In order for you to start making games with Phaser you will need to have a **web browser** (Google Chrome will be used for this course), **code editor**, and a **web server**.

There are many different code editors out there that are free to use, and it doesn't matter which one you choose.

- A list of code editors that are free
  - Brackets(<https://brackets.io>)
  - Atom(<https://atom.io>)
  - Sublime text(<https://www.sublimetext.com>)
  - Visual Studio Code(<https://code.visualstudio.com/>)

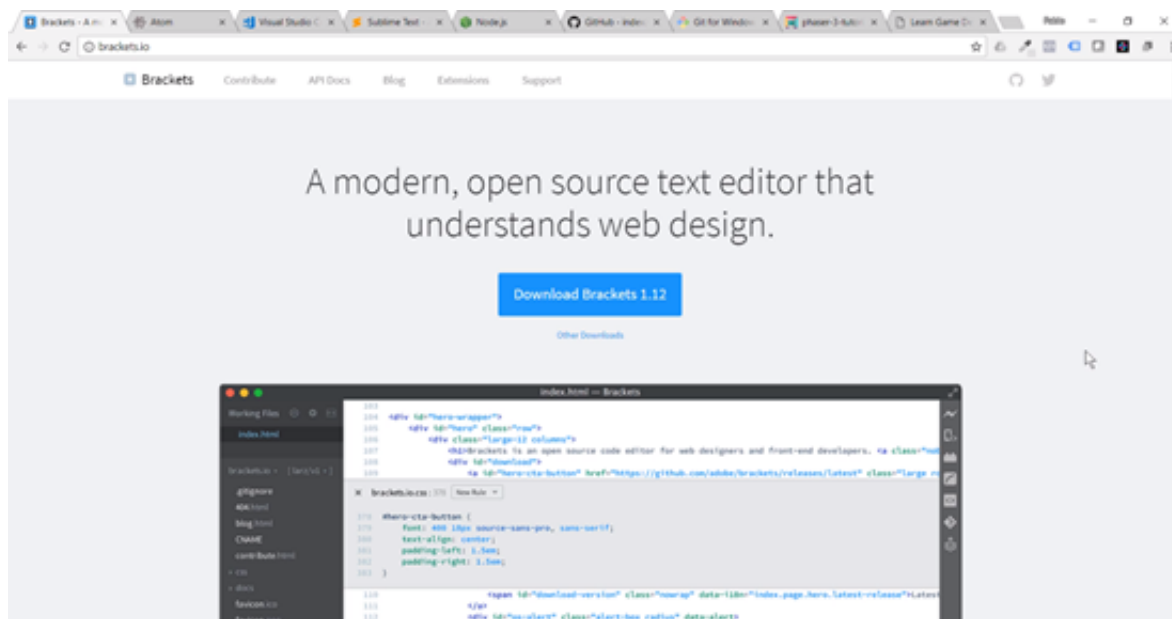
## Code Editors

- Brackets (<https://brackets.io>)
- Atom (<https://atom.io>)
- Sublime Text (<https://www.sublimetext.com/>)
- Visual Studio Code (<https://code.visualstudio.com/>)

**A web browser is a program that will receive requests from the browser, and we will send, it will serve files as a response.**

A browser doesn't allow you to load files from the file protocol due to security reasons. The web would be a very dangerous place if random websites could get access to the files on your computer. This is why you cannot just go and open a Phaser game by double clicking on the index.html file.

You have to load the game through a **web server**. The simplest way to get a web server up and running is to install the Brackets code editor.



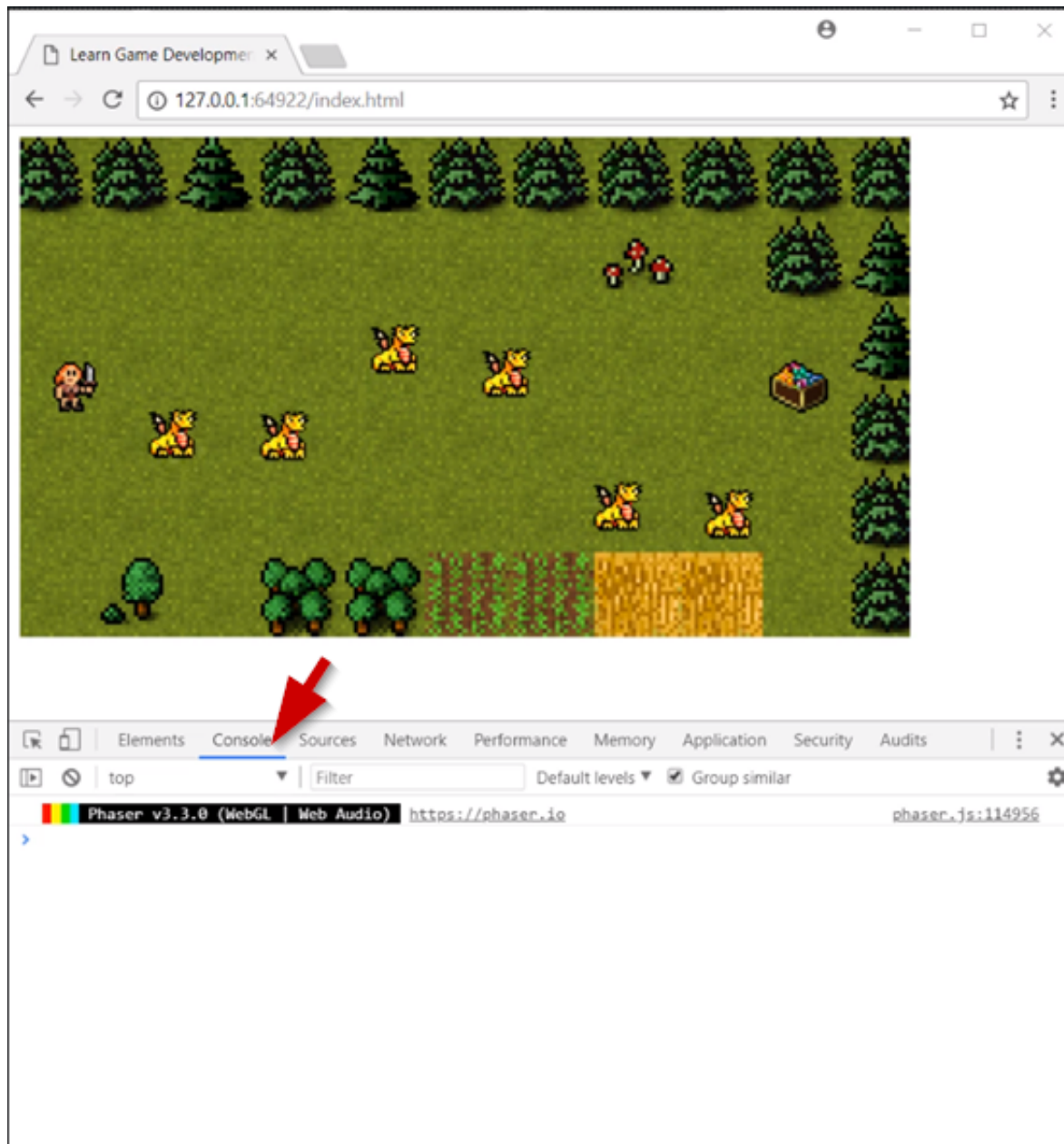
You can **download** Brackets from here: <http://brackets.io>

The **direct link** to download Brackets is here: [Download Brackets](#)

Brackets comes with a built-in web server and it is free to use.

- Web Server-Brackets
  - Download and install Brackets (<https://brackets.io>)
  - File-Open Folder
  - Live preview

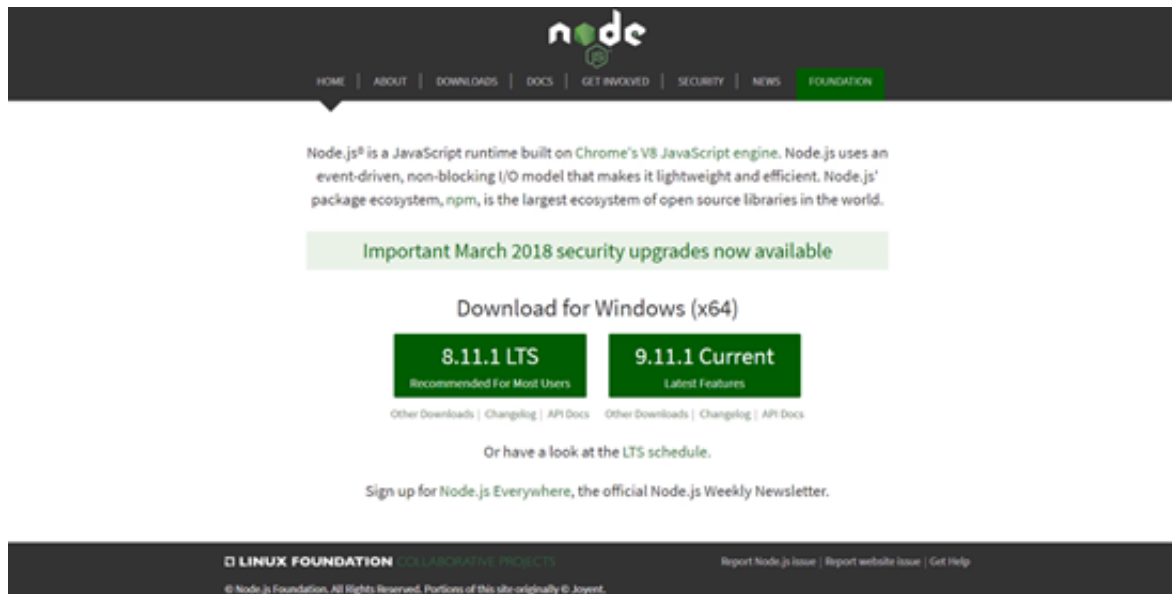
Always keep the Chrome Developer Tools open when working.



There is a more advanced way of using a web server and this involves using a called http server.

The first step is to install **Node JS**, and you can download Node JS from here: <https://nodejs.org/en/>

The direct link to **download** Node JS is here: [Node JS Download](#)



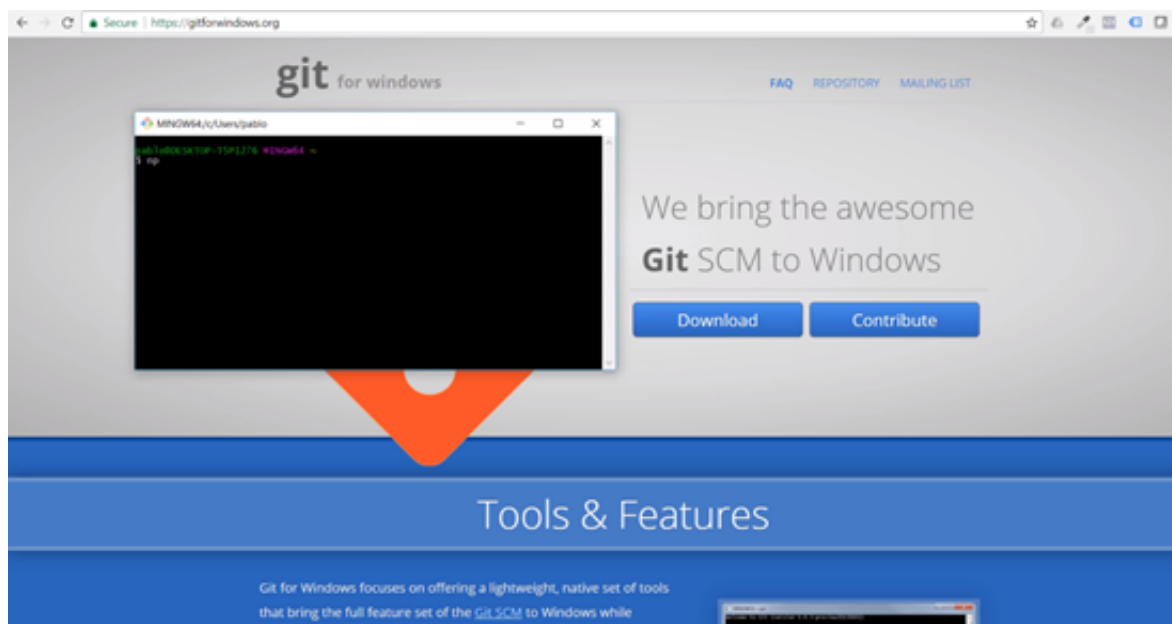
**Node JS is an application that allows you to run JavaScript code on a server, in this case your computer.**

For Window Users Only: Once you have downloaded Node JS you will need to download Git Bash, and this will give you access to a terminal on your system.

Mac users will already have access to the Mac terminal.

You can download **Git Bash** for windows here: <https://gitforwindows.org/>

The direct link to **download** Git Bash for windows is here: [Git Bash](#)



The first step is type **“npm”** in the terminal

If you type this and it is installed you will see this:



```
MINGW64/c/Users/pablo

pablo@DESKTOP-T5P1276 MINGW64 ~
$ npm
Usage: npm <command>

where <command> is one of:
  access, adduser, bin, bugs, c, cache, completion, config,
  ddp, dedupe, deprecate, dist-tag, docs, doctor, edit,
  explore, get, help, help-search, i, init, install,
  install-test, it, link, list, ln, login, logout, ls,
  outdated, owner, pack, ping, prefix, profile, prune,
  publish, rb, rebuild, repo, restart, root, run, run-script,
  s, se, search, set, shrinkwrap, star, stars, start, stop, t,
  team, test, token, tst, un, uninstall, unpublish, unstar,
  up, update, v, version, view, whoami

npm <command> -h      quick help on <command>
npm -l               display full usage info
npm help <term>      search for help on <term>
npm help npm         involved overview

Specify configs in the ini-formatted file:
  C:\Users\pablo\.npmrc
```

Once you verify that the npm is installed you can then type **"npm install http-server -g"** this is the name of the package that we want installed for us.

We now need to navigate and find our folder where the game is located.

In my case the game is located in the D drive, the **www folder>Phaser3>crossy-rpg** this will be different in your case.

If you have issues with using the command line options here you can just use Brackets, Brackets is much easier to use and doesn't require this extra setup for the http server setup.

So in the terminal you type **"cd d:"** this brings you to the D drive.

Then type **"cd www/phaser3/crossy-rpg/"**

Now you should be inside the correct folder in the terminal.

```
MINGW64/d/www/phaser3/crossy-rpg

npm <command> -h      quick help on <command>
npm -l               display full usage info
npm help <term>      search for help on <term>
npm help npm         involved overview

Specify configs in the ini-formatted file:
  C:\Users\pablo\.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@5.6.0 D:\Program Files\nodejs\node_modules\npm
pablo@DESKTOP-T5P1276 MINGW64 ~
$ cd d:

pablo@DESKTOP-T5P1276 MINGW64 /d
$ cd www/phaser
phaser/ phaser3/

pablo@DESKTOP-T5P1276 MINGW64 /d
$ cd www/phaser3/crossy-rpg/

pablo@DESKTOP-T5P1276 MINGW64 /d/www/phaser3/crossy-rpg (master)
$ |
```

Now that you are inside the correct folder you then type **"http-server"** and that should launch the web server for that particular folder.

You should now see that the server is available in different URLs.

```
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@5.6.0 D:\Program Files\nodejs\node_modules\npm

pablo@DESKTOP-T5P1276 MINGW64 ~
$ cd d:

pablo@DESKTOP-T5P1276 MINGW64 /d
$ cd www/phaser
phaser/ phaser3/

pablo@DESKTOP-T5P1276 MINGW64 /d
$ cd www/phaser3/crossy-rpg/

pablo@DESKTOP-T5P1276 MINGW64 /d/www/phaser3/crossy-rpg (master)
$ http-server
Starting up http-server, serving ./
Available on:
  http://192.168.99.1:8080
  http://192.168.1.103:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

You then can copy one of the URLs and go back into Google Chrome or whatever web browser you are using and paste the copied code into the navigation bar.

```
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

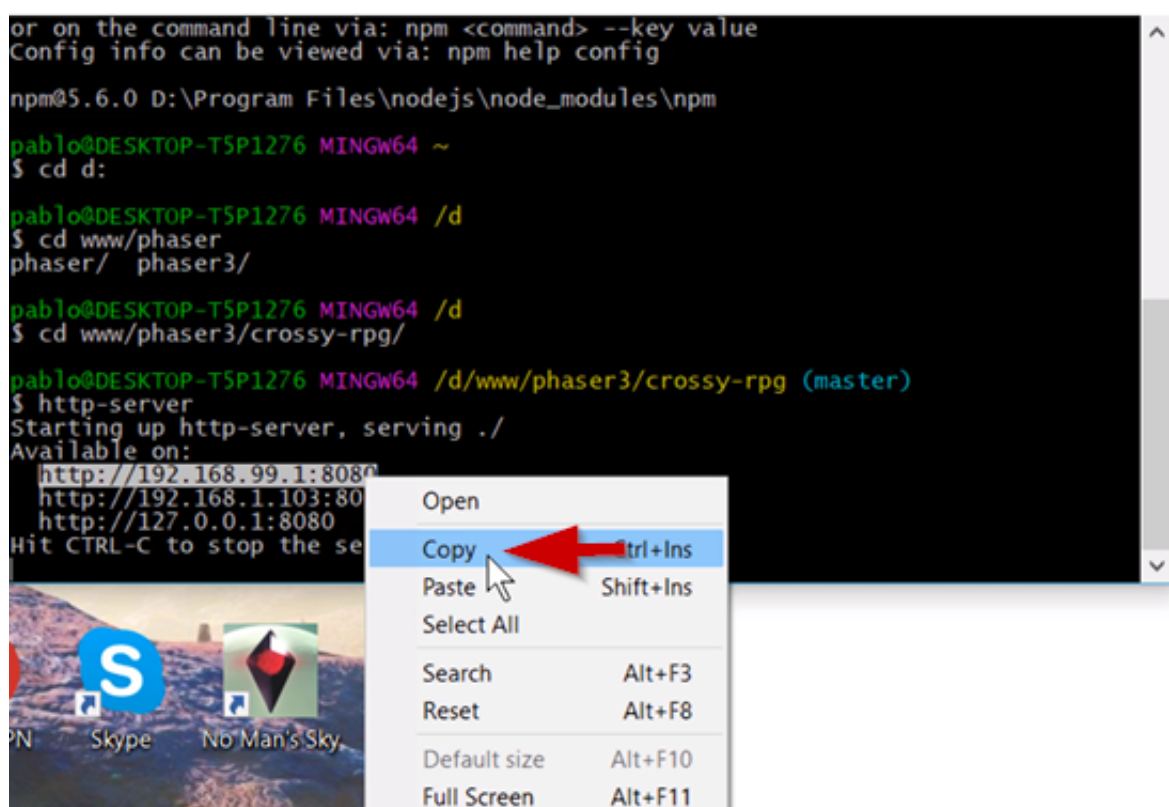
npm@5.6.0 D:\Program Files\nodejs\node_modules\npm

pablo@DESKTOP-T5P1276 MINGW64 ~
$ cd d:

pablo@DESKTOP-T5P1276 MINGW64 /d
$ cd www/phaser
phaser/ phaser3/

pablo@DESKTOP-T5P1276 MINGW64 /d
$ cd www/phaser3/crossy-rpg/

pablo@DESKTOP-T5P1276 MINGW64 /d/www/phaser3/crossy-rpg (master)
$ http-server
Starting up http-server, serving ./
Available on:
  http://192.168.99.1:8080
  http://192.168.1.103:80
  http://127.0.0.1:8080
Hit CTRL-C to stop the se
```



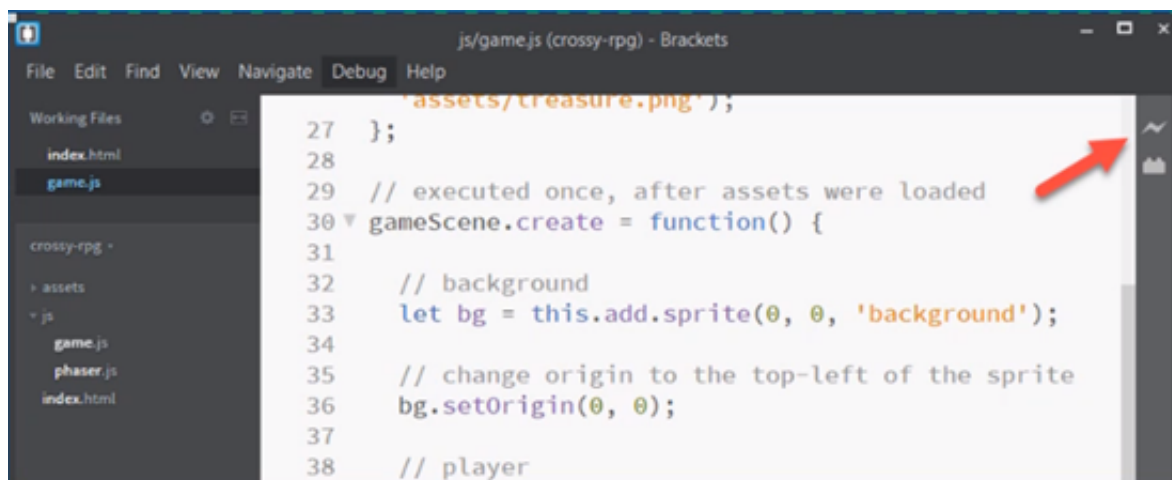




And now the game will be working just like it should.

## Learning Summary

- There are three basic requirements for Phaser development
  - Web browser
  - Code editor
  - Web Server
- A list of code editors that are free
  - Brackets(<https://brackets.io>)
  - Atom(<https://atom.io>)
  - Sublime text(<https://www.sublimetext.com/>)
  - Visual Studio Code(<https://code.visualstudio.com/>)
- Web Server-Brackets
  - Download and install Brackets (<https://brackets.io>)
  - File-Open Folder
  - Live preview



- Web Server-http-server
  - Download and install Node.js (<https://nodejs.org>)
  - (Windows Only) install Git Bash (<https://gitforwindows.org/>)



- Open terminal, install http-server: `npm install http-server -g`
- Navigate to the project folder, run the server: `http-server`



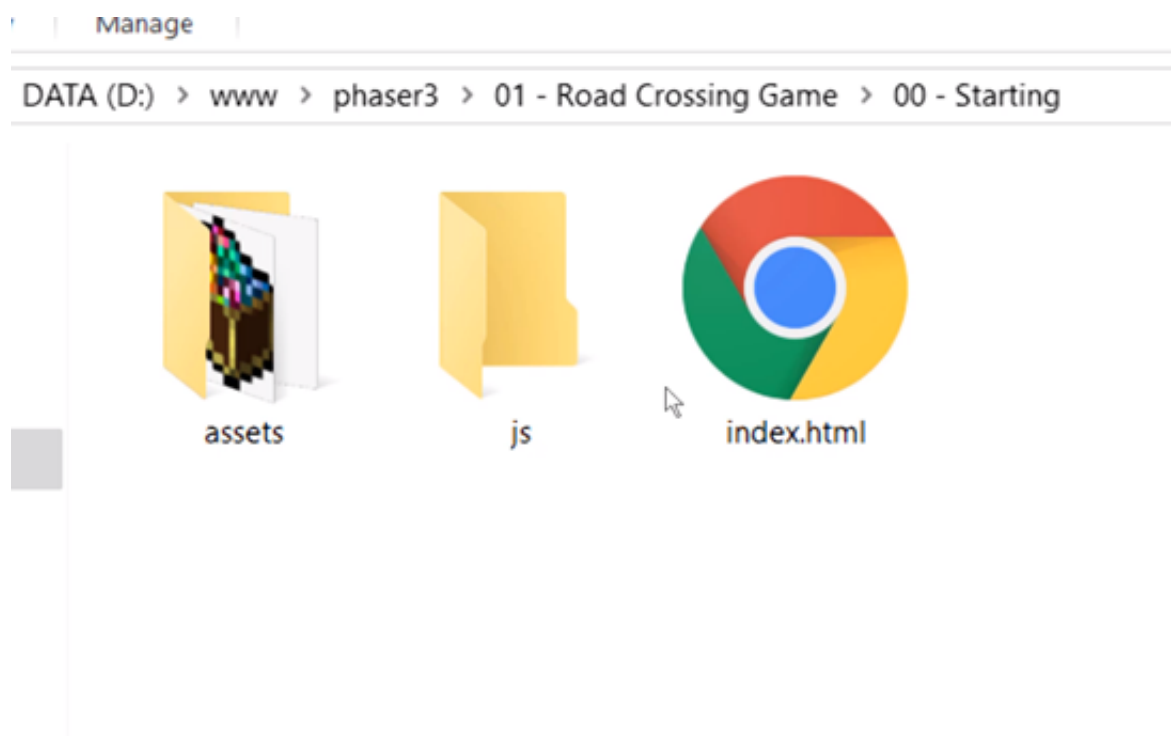
## Learning Goals

- Including Phaser in a project.
- Passing configuration to a new game.
- Creating a Phaser game with a scene.

In this lesson we will begin creating the new Phaser game.

It will be an empty game on an empty scene, and you will learn what a scene is.

**Download** the files for the course, and make sure you find the folder named “**oo-Starting**”



You will be building the game along as you watch the lessons in the course.

The **js** folder is empty and this is where all the **JavaScript files** will go.

There is also a folder called **assets** and this contains all the images for the project.

You can **open the index.html file in your code editor**. I am using Atom in this course for my code editor.

This is a simple file that doesn't really have anything.

We will need to **start by including the Phaser library**.

You can find the Phaser library at [phaser.io](https://phaser.io).

The direct link is here: [Phaser Download](https://github.com/phaserjs/phaser)



If you want to put your game on the internet you will always go with the **minified version**. This version has all the comments stripped out of it.



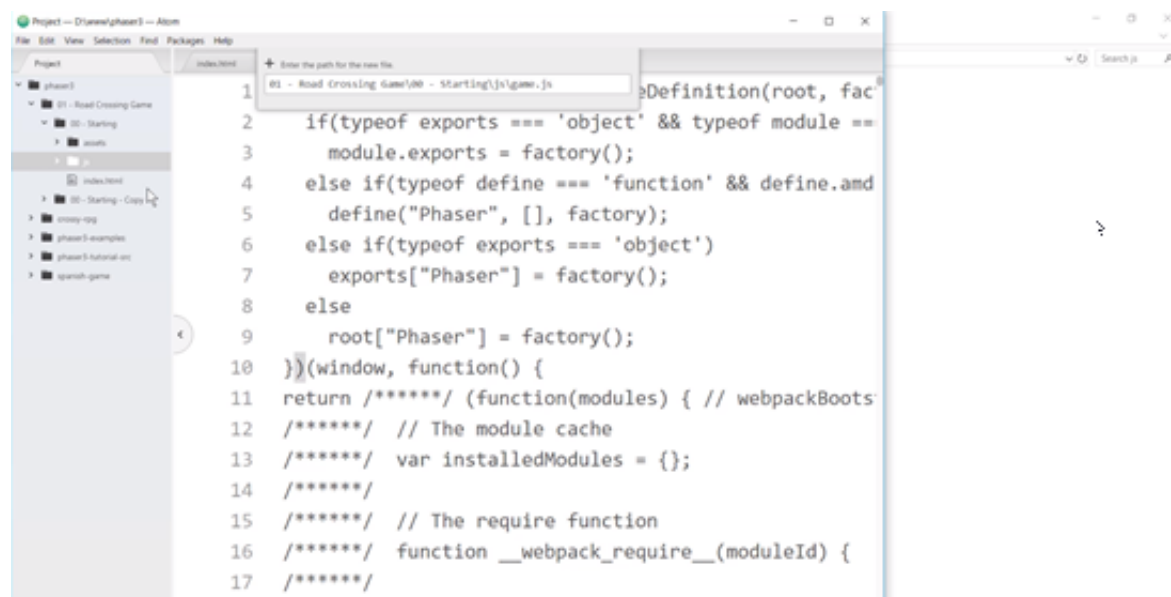
You need to **save the min.js file** in the JavaScript folder.

Then you need include the file in the index.html file.

See the code below and follow along:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Learn Game Development at Zenva.com</title>
</head>
<body>
  <script src="js/phaser.js"></script>
</body>
</html>
```

Create a new file called **"game.js"** and this is where the game code will go.



Make sure to **include that file in the index.html file**, but after the Phaser file. See the code below and follow along:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Learn Game Development at Zenva.com</title>
</head>
<body>
  <script src="js/phaser.js"></script>
  <script src="js/game.js"></script>
</body>
</html>
```

Inside the **game.js** file there are three things that need to happen. We need to **create a new scene**, set the **configuration of the game**, and **create the new game** and pass the **configuration** to it.

A **scene** is where all the action takes place. This is where all the game characters, enemies, and everything in the game will go in the scene. In Phaser you can actually have multiple scenes. You can use scenes to represent different screens, or in some cases different levels.

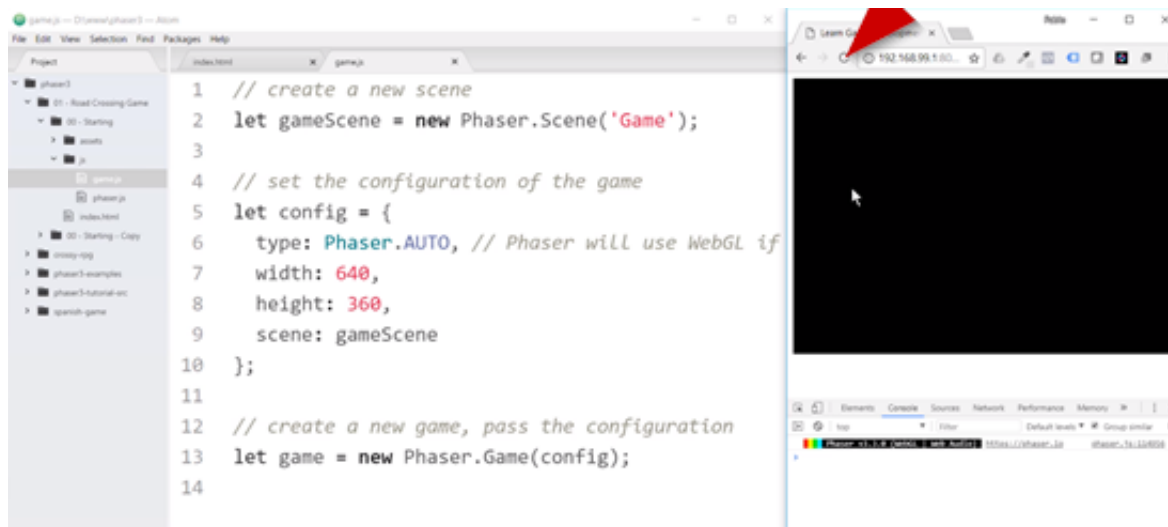
See the code below and follow along:

```
// create a new scene
let gameScene = new Phaser.Scene('Game');

// set the configuration of the game
let config = {
  type: Phaser.AUTO, // Phaser will use WebGL if available, if not it will use Canvas
  width: 640,
```

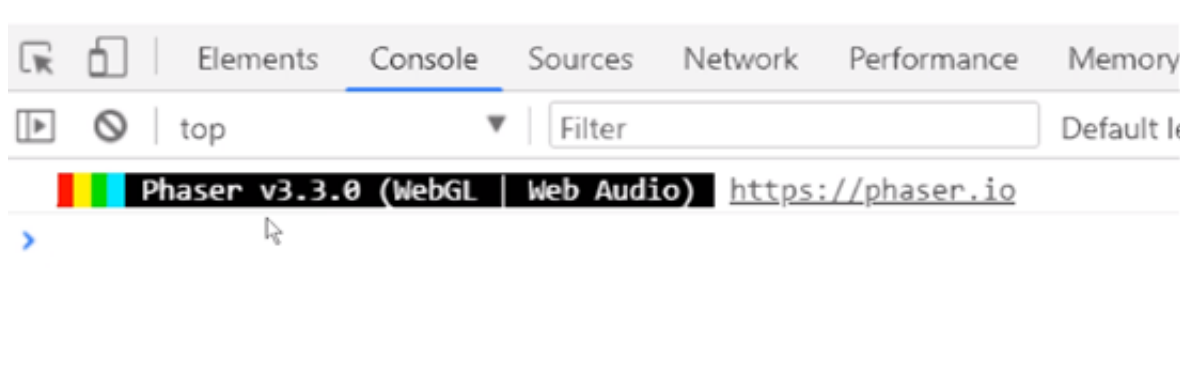
```
height: 360,  
scene: gameScene  
};  
  
// create a new game, pass the configuration  
let game = new Phaser.Game(config);
```

Then test out the changes by **refreshing** the web page:



There isn't much happening currently, there is a black screen of the size that we defined.

You will see that there is an indication in the **Console window** that Phaser is indeed working though.



In the next lesson we will be adding images to the screen.

## Learning Summary

- Obtaining Phaser
  - Phaser homepage(<http://phaser.io>)
  - Version: 3.x
  - Options:
    - Direct Download



- CDN
- Github
- npm
- Creating a new game
  - Create a scene
  - Configuration
  - Create game object

```
// create a new scene
let gameScene = new Phaser.Scene('Game');

// set the configuration of the game
let config = {
  type: Phaser.AUTO, // Phaser will use WebGL if available
  width: 640,
  height: 360,
  scene: gameScene
};

// create a new game, pass the configuration
let game = new Phaser.Game(config);
```

## Corrections

At 11'00", I say "340" when I should have said "640" (this was coded correctly on screen).

At 15'14" on the Slide "Coordinate System", the text should instead read:

X coordinate: positive to the LEFT, negative to the RIGHT

Y coordinate: positive DOWNWARDS, negative UPWARDS

## Learning Goals

- Phaser's preload and create methods.
- Rendering a sprite.
- Coordinate system.
- Sprite origin.

In this lesson you will learn how to display images on the screen using Phaser. The images that will be used during this lesson can be downloaded from the course. They are in the assets folder. There are a total of 4 images in the folder: background, dragon, player, and treasure.

We will begin by loading the background, and the loading will occur inside of our scene.

The diagram for the **life cycle of a scene**:



The **init() method**- called once this is used to initiate certain parameters of your scene. This method is not always used.

**preload() method**-all asset preloading takes place. This means that Phaser will load all of the images, all the audio files, and other external files you may want to use in your game. All of the files will be loaded to memory so that they can be used without any delay. When you want to show an enemy or character, you don't want the player to have to wait until that image is loaded.

**create() method**-called once and this is actually where you create the sprites and display them on the screen.

**update() method**-we will look at this method later on in the course, but it is called on each frame during game play. This method is used when things need to be checked all the time.

We will now create the preload method for the scene. See the code below and follow along:

```
// create a new scene
let gameScene = new Phaser.Scene('Game');

// load assets
gameScene.preload = function(){
  // load images
  this.load.image('background', 'assets/background.png');
  this.load.image('player', 'assets/player.png');
};
```

```
// called once after the preload ends
gameScene.create = function() {
  // create bg sprite
  let bg = this.add.sprite(0, 0, 'background');

  // change the origin to the top-left corner
  //bg.setOrigin(0,0);

  // place sprite in the center
  bg.setPosition(640/2, 360/2);

  let gameW = this.sys.game.config.width;
  let gameH = this.sys.game.config.height;

  console.log(gameW, gameH);

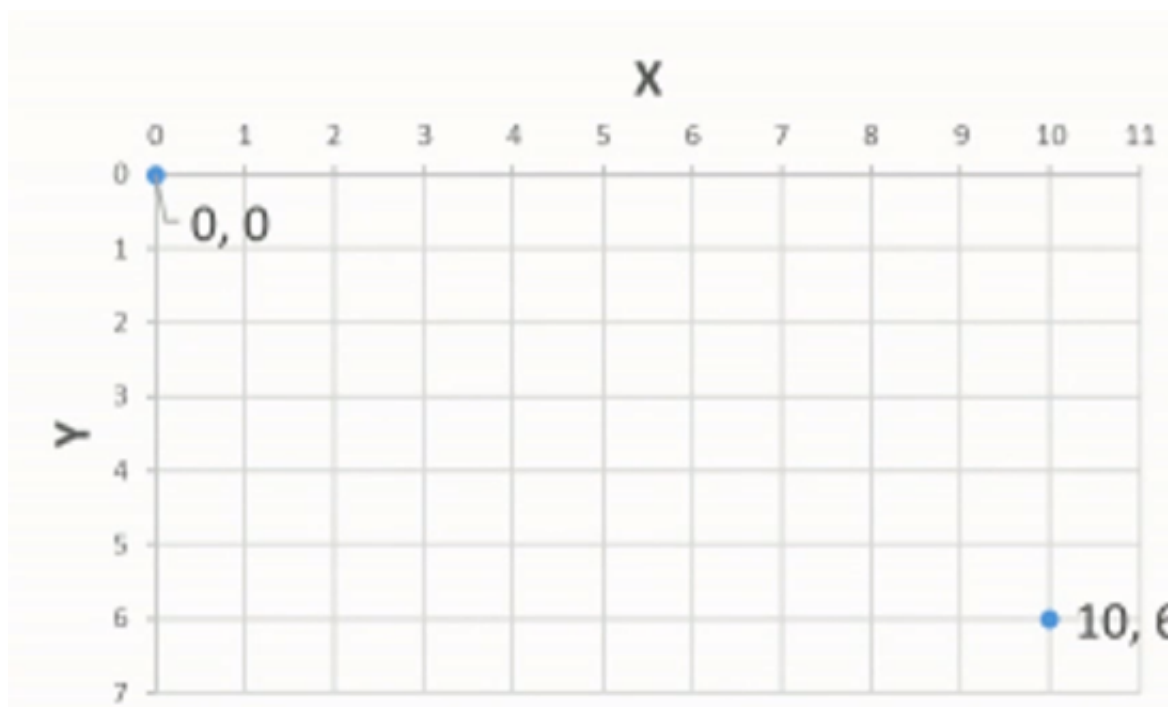
  console.log(bg);
  console.log(this);
};

// set the configuration of the game
let config = {
  type: Phaser.AUTO, // Phaser will use WebGL if available, if not it will use Canvas
  width: 640,
  height: 360,
  scene: gameScene
};

// create a new game, pass the configuration
let game = new Phaser.Game(config);
```

In Phaser games a **two dimensional coordinate system** is used. There is an X axis that goes from left to right. There is a Y axis that goes from top to bottom. The **origin(0,0)** is always at the top left corner of the game.





If we said we wanted the player to begin in position (5,4) What we are saying is that the center of our sprite will go in that position.

- **X:** Positive to the right, negative to the left.
- **Y:** Positive upwards, negative downwards.

By default the origin is the middle point on X and the middle point on Y.

You can use the **bg.setOrigin(0,0)** method to change the origin of sprites.

## Learning Summary

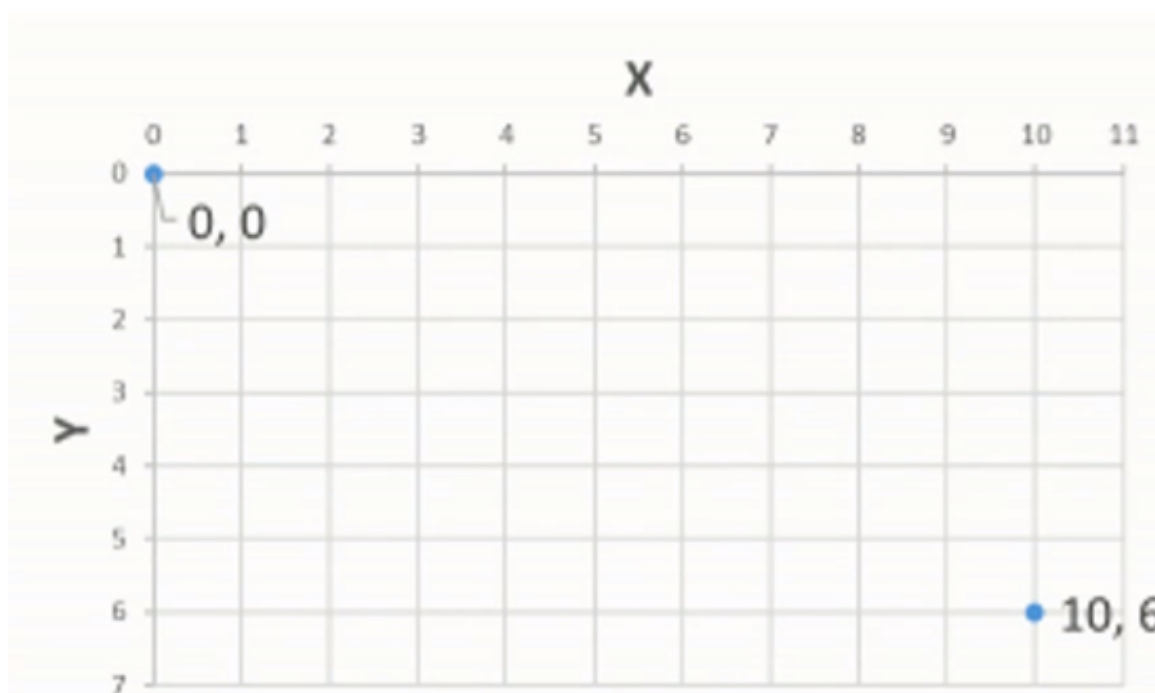




# Preloading assets

```
// Load assets
gameScene.preload = function(){
  // Load images
  this.load.image('background', 'assets/background.png');
  this.load.image('player', 'assets/player.png');
};
```

- Sprites
  - Game characters and elements.
  - Origin: by default, the middle.
  - Position on x, y can be accessed and changed: set position.
- Coordinate system
  - X: Positive to the right, negative to the left.
  - Y: Positive upwards, negative downwards.



- Challenge
  - Place the sprite in the middle of the screen by changing the position.
  - Don't change the origin.
  - Use the setPosition method.



### Some important corrections to the content of this video:

- 1) Scaling a sprite by 2 does NOT double its size, it doubles its width and its height. If you double both the width and the height, the area after scaling is 4x the initial area!
- 2) Similarly, if you scale a sprite by 0.5 you are NOT reducing its area by half — you are reducing both the width and the height by half. This means, the total area is multiplied by 0.25, so a quarter of the original area!

## Learning Goals

- Sprite depth
- Changing the size of a sprite.
- Flipping a sprite on X or Y.

We are now familiar with how to position sprites on the screen, as we learned that in the previous lesson.

In this lesson you will learn how to increase or reduce the size of our sprites. This is called **scaling**. You will also learn how to flip sprites, and also what the depth of the sprite does when you render multiple sprites.

We will begin coding by adding a second sprite, which will be the player. See the code below and follow along:

```
// create a new scene
let gameScene = new Phaser.Scene('Game');

// load assets
gameScene.preload = function(){
  // load images
  this.load.image('background', 'assets/background.png');
  this.load.image('player', 'assets/player.png');
  this.load.image('enemy', 'assets/dragon.png');
};

// called once after the preload ends
gameScene.create = function() {
  // create bg sprite
  let bg = this.add.sprite(0, 0, 'background');

  // change the origin to the top-left corner
  bg.setOrigin(0,0);

  // create the player
  let player = this.add.sprite(70, 180, 'player');

  // we are reducing the width by 50%, and we are doubling the height
  player.setScale(0.5);

  // create an enemy
  let enemy1 = this.add.sprite(250, 180, 'enemy');
```



```
enemy1.scaleX = 2;
enemy1.scaleY = 2;

// create a second enemy
let enemy2 = this.add.sprite(450, 180, 'enemy');
enemy2.displayWidth = 300;

// flip
enemy1.flipX = true;
enemy1.flipY = true;
};

// set the configuration of the game
let config = {
  type: Phaser.AUTO, // Phaser will use WebGL if available, if not it will use Canvas
  width: 640,
  height: 360,
  scene: gameScene
};

// create a new game, pass the configuration
let game = new Phaser.Game(config);
```

By default sprites are shown in order, this order is in which they are added. Since we added the background first and then added the player next. The player is showing on top of the background. You can change this by changing the depth of the sprites. The higher the number, the more on top the sprite will be.

When you have sprites and you want to have control over what goes on top you can use the **depth property**.

In order to scale sprites you can use the **set scale method** which works in a similar way to set origin. This means we will have a value for both X and Y. Increase (>1) or reduce (<1) the size of sprites, on X and/or Y. displayWidth and displayHeight: rendered size of the sprite.



```
// create the player
let player = this.add.sprite(70, 180, 'player');

// we are reducing the width by 50%, and we are doubling the height
player.setScale(0.5, 2);

// create an enemy
let enemy1 = this.add.sprite(250, 180, 'enemy');

enemy1.scaleX = 2;
enemy1.scaleY = 2;

// create a second enemy
let enemy2 = this.add.sprite(450, 180, 'enemy');
enemy2.displayWidth = 300;
```

You can **flip a sprite** for both X and Y axes.

```
// flip
enemy1.flipX = true;
enemy1.flipY = true;
```

## Learning Summary

- Sprite depth
  - by default, sprites are stacked in the order that are created.
  - The depth property can be used to customize which ones go on top.
  - Sprites with a higher depth value are shown above those with a lower value.
- Scaling
  - Increase (>1) or reduce (<1) the size of sprites, on X and/or Y.
  - displayWidth and displayHeight: rendered size of the sprite.

```
// create the player
let player = this.add.sprite(70, 180, 'player');

// we are reducing the width by 50%, and we are doubling the height
player.setScale(0.5, 2);

// create an enemy
let enemy1 = this.add.sprite(250, 180, 'enemy');

enemy1.scaleX = 2;
enemy1.scaleY = 2;

// create a second enemy
let enemy2 = this.add.sprite(450, 180, 'enemy');
enemy2.displayWidth = 300;
```

- Flipping
  - Can be done for both X and Y.

```
// flip
enemy1.flipX = true;
enemy1.flipY = true;
```



## Learning Goals

- Rotating a sprite in angles and radians.
- Update method.
- Changing a property on each frame.

In this lesson you will learn about the **rotation of sprites and the update method**. The update method is part of the scenes life cycle and it is a method that we can use to change the property of something over time.

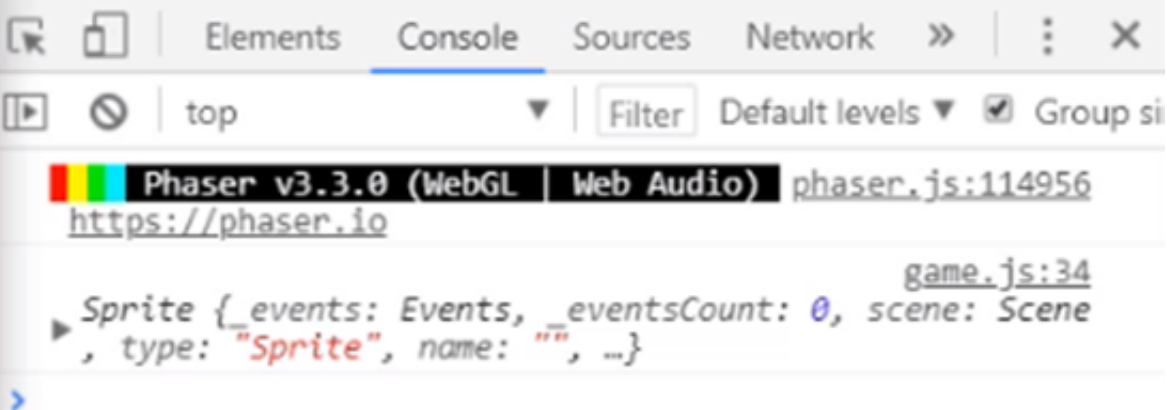
There are two ways of rotating a sprite. The rotation always takes place about its origins, so the position of the origin doesn't change.

We will begin by increasing the size of the enemy a little bit, see the code below and follow along:

```
// create an enemy
this.enemy1 = this.add.sprite(250, 180, 'enemy');
this.enemy1.setScale(3);

this.enemy1.angle = 45;
this.enemy1.setAngle(45);
```



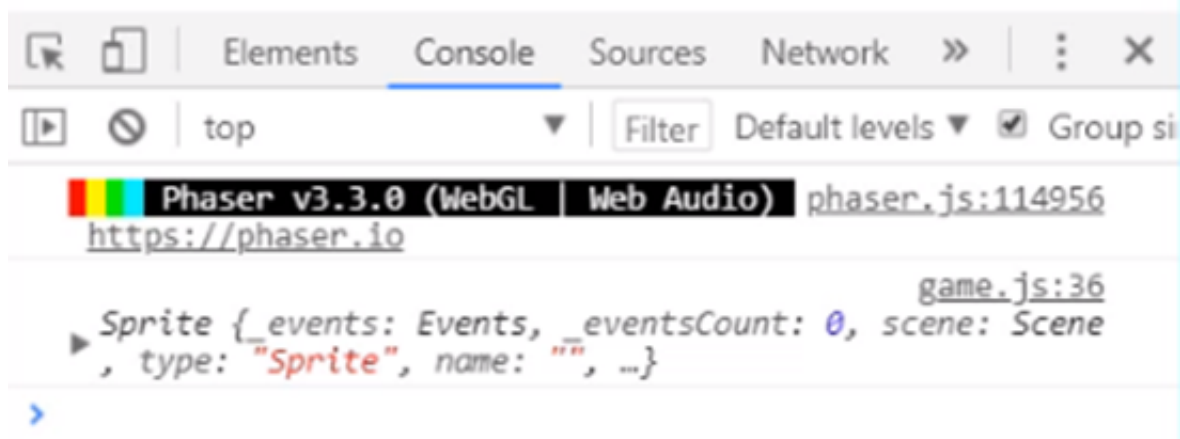


This is how you rotate a sprite by **picking the angles option**.

See the code below and follow along with **how to use the radians option to rotate the sprite**:

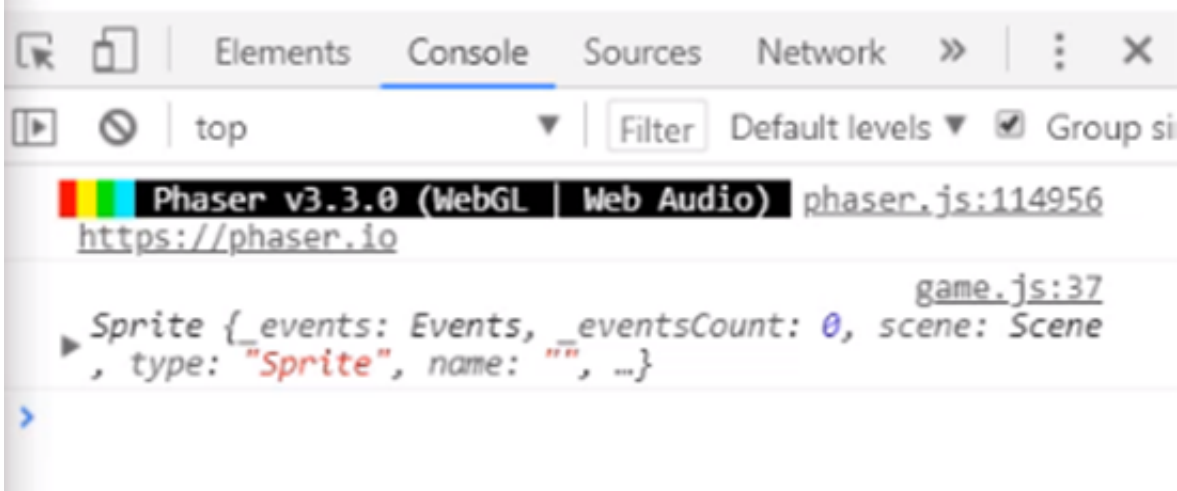
```
this.enemy1.rotation = Math.PI / 4;  
this.enemy1.setRotation(Math.PI / 4);
```

You can see that we are rotating about the origin, which is in the middle.



Now if we didn't want to rotate about the middle anymore, and rotate about a specific point we could do that, see the code below and follow along:

```
this.enemy1.setOrigin(0, 0);  
this.enemy1.rotation = Math.PI / 4;  
this.enemy1.setRotation(Math.PI / 4);
```



This is how you can rotate sprites and now you know how to change a sprites position, how to scale them up and down, flip sprites, and you now know how to rotate them.

The next thing to cover in this lesson is the **update method**.

In a previous lesson we went over how scenes had special methods.



The **update() method** is called 60 times per second if that is of course supported by the device.



60 frames per second(fps) is the frame rate that the framework aims to have.

We will now create the update method, see the code below and follow along:

```
// this is called up to 60 times per second
gameScene.update = function(){
    //this.enemy1.x += 1;

    this.enemy1.angle += 1;
```

Now there is a challenge for you, and it has to do with playing with the update method.

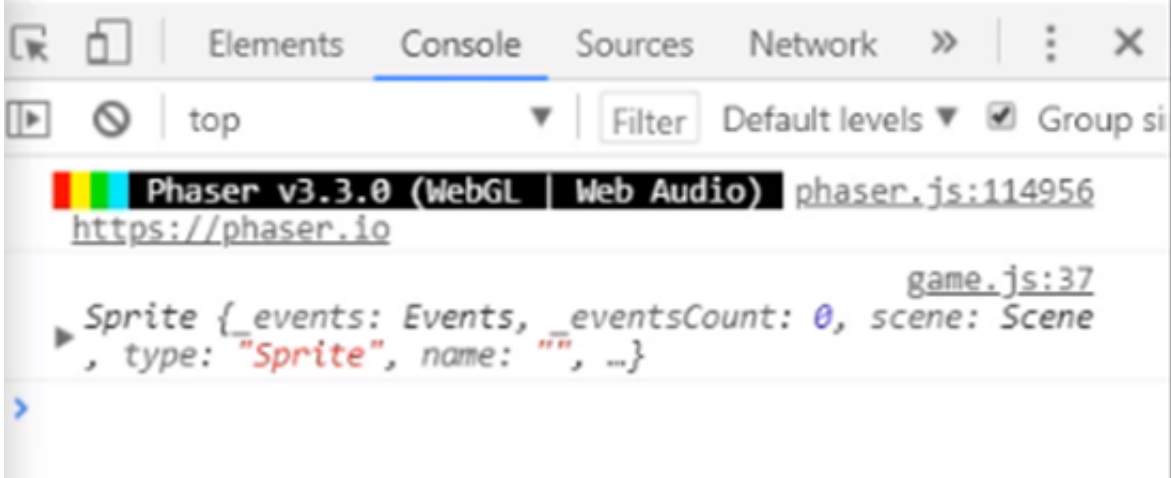
**Challenge-** Make a sprite grow over time, only until it reaches twice it's original dimensions(width and height)

Try to solve the challenge on your own without looking at the solution, but if you need help feel free to watch the solution and follow along.

Solving the challenge:

```
// check if we've reached scale of 2
if(this.player.scaleX < 2) {
    // make the player grow
    this.player.scaleX += 0.01;
    this.player.scaleY += 0.01;
}
```





## Learning Summary

- Rotation
  - Rotation using your left hand is positive.
  - In angles: `mySprite.angle = 45;`
  - In radians: `mySprite.rotation = Math.PI/4;`





- `update()`
  - Called once up to 60 times per second.
  - Used for things that need to be done or checked “at all times”
  - Executed for the first time after `create()`
- Challenge
  - Make a sprite grow over time, only until it reaches twice it’s original dimensions(width and height)