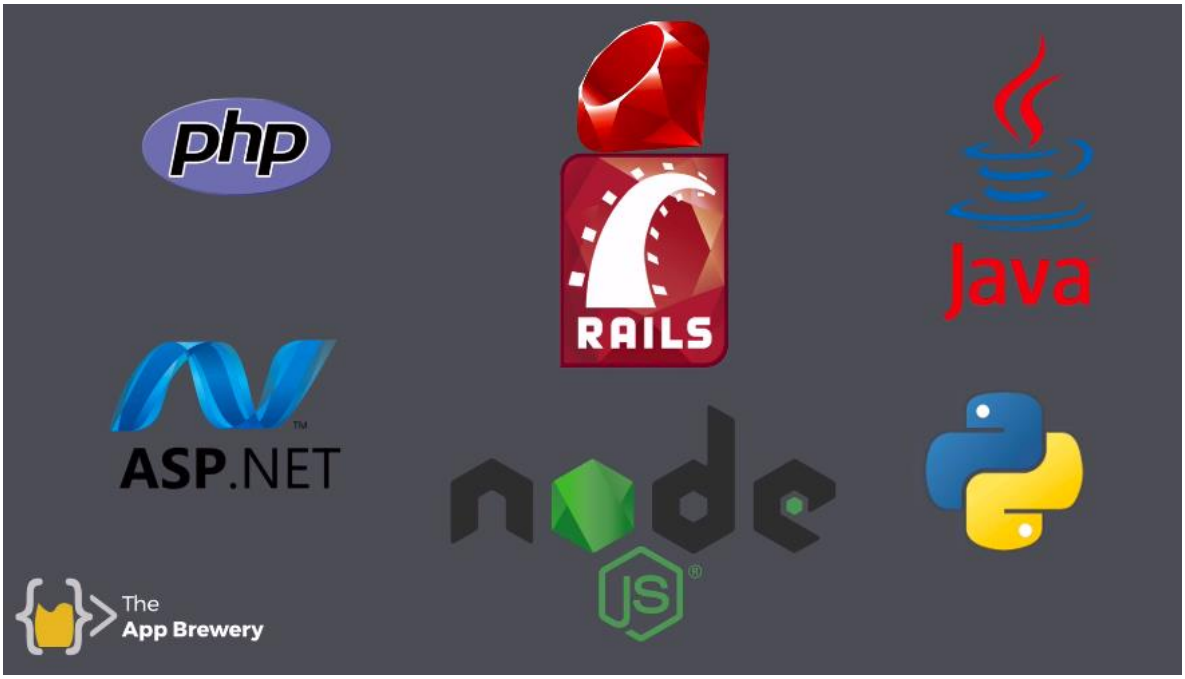
SECTION 18: NODE JS:

WHAT IS NODE.JS?



Python is used to create web applications too.
NODE JS → Allow us to create backend code using JavaScript.

COMPANIES THAT USE NODE JS:

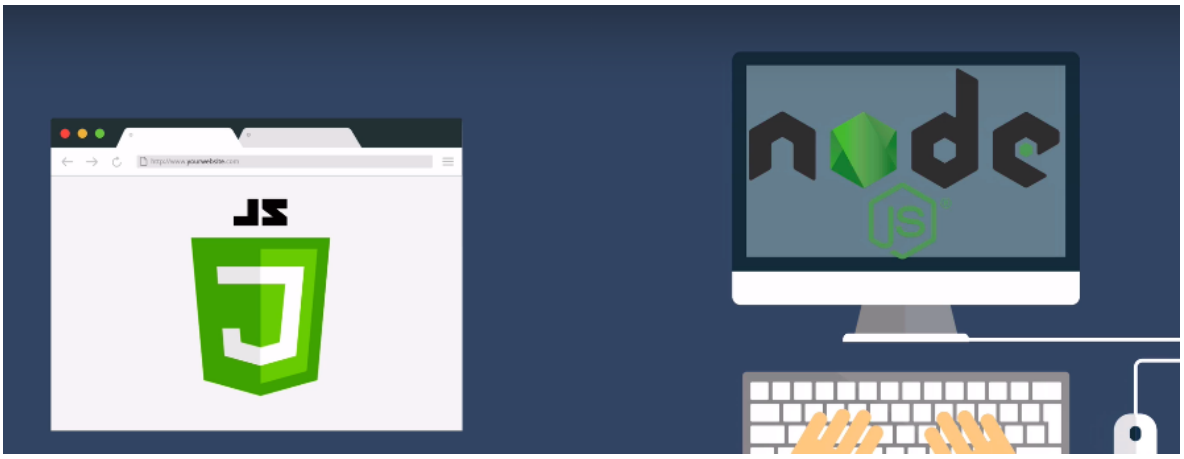
Until now we've learnt to build code that runs on the browser with JS.

But we cannot access the filesystem of the users:

For example Desktop applications do that.

IN OTHER WORDS, WE NEED TO WRITE CODE THAT INTERACTS DIERECTLY WITH THE COMPUTER, INDEPENDLY OF THE BROWSER.

NODE JS ALLOW US TO TAKE JS OUT OF THE BROSER AND LIBERATE IT ALLOWING IT TO INTERACT DIRECTLY WITH THE HARDWARE OF A COMPUTER.



Atom is a desktop application based on node.js

IN OTHER WORDS WE USE NODE JS TO WRITE JS CODE IN SOMEBODY ELSE'S COMPUTER (A SERVER).

FOR EXAMPLE A REQUEST FROM GOOGLE:

ALL THE LOGIC HAPPENS ON THE SERVER MAKING THE APPLICATION VERY FAST.

USES OF NODEJS:

- Access local files.
- Listen to Network traffic.
- Manipulate databases.

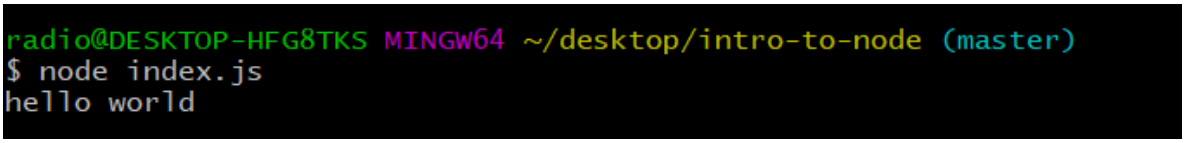
THE POWER OF THE COMMAND LINE AND HOW TO USE NODE

We start creating a new folder inside desktop, called intro-to-node.
We created a file called index.js
We wrote some code inside index.js

Until now we were only able to run JS code if we incorporated it inside an html file but now we are going to run it using a computer.

node index.js → It says, use node to run this file.

And you can see the ouput in the computer console (not the browser):



THE NODE REPL (READ EVALUATION PRINT LOOPS)

Allows to execute code in byte size chunks.

To access repl only type:

node and hit enter. You will see a prompt like this > → Here you can write code directly.

Inside you can get shortcut help like for exampe typping ctrl + tab, you will get a lot of possible commands.

Press tab twice and you'll see more possibilities.

.exit or control+c (twice) → To exit the repl

HOW TO USE NATIVE NODE MODULES

Remember from JS:

const a = 2; → Used to create constants. (This value cannot be changed);

For example to use the file system module:

You need (require) → That is another method from node.

With using file system you can:

- Specify paths.
- Open files.
- Change files.
- Access files.
- Read files.

Example of just one of the multiple things that we can do with the module:

const fs = require("fs"); → First we need to require the module.

Then, (the syntax is in the node documentation):

fs.copyFileSync("file1.txt", "file2.txt"); → It'll look into the current directory, it'll look for a file called file1.txt and copy it into a file called file2.txt

At the moment file2.txt doesn’t exist → This is going to be the name of the copy.

- **The example showed in the tutorial didn’t work straight so I made some researching and it was successful:**

```
//jshint esversion: 6

const fs = require("fs");
fs.copyFileSync("C:/Users/radio/Desktop/intro-to-
node/file1.txt",                              "NEWFILE.txt",
fs.constants.COPYFILE_EXCL);
```

In the case that NEWFILE.txt already existed, the command is going to replace the existing content. **It’s necessary to do some more research to solve this point.**

```
//jshint esversion: 6

const fs = require("fs");
fs.copyFileSync("C:/Users/radio/Desktop/intro-to-
node/file1.txt",                            "NEWFILE1.txt",
fs.constants.COPYFILE_EXCL);
```

In the following paragraph we are going to see how to use external modules.

THE NPM PACKAGE MANAGER AND INSTALLING EXTERNAL NODE MODULES

NPM → Package manager for external modules

The packages are bits of reusable code that somebody else wrote, and you can incorporate those packages into your project, saving the time and effort.

npm init → To initialize npm and create a package.json

Everytime that we start to install external packages, they are going to show up in the package.json too.

Example:

```
{
  "name": "intro-to-node",
  "version": "1.0.0",
  "description": "This is an intro to node project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "ANDRES R. BUCHELI",
  "license": "ISC"
}
```

- **We are going to install a external package called superheroes:**

```
npm install superheroes;
```

And now the package.json is going to change:

```
{
  "name": "intro-to-node",
  "version": "1.0.0",
  "description": "This is an intro to node project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "ANDRES R. BUCHELI",
  "license": "ISC",
  "dependencies": {
    "superheroes": "^3.0.0"
  }
}
```

Now you can see that the project has dependencies, it means it depends of the superheroes package.

- **Example of how to use the intalled package: (In the index.js file):**

```
var superheroes = require('superheroes');

var mySuperheroName = superheroes.random();

console.log(mySuperheroName);
```

Finally just run with node like always: → node index.js

SUMMARY:

npm init → initializes a package file.

The project depend on the installed package because we require it inside our project.