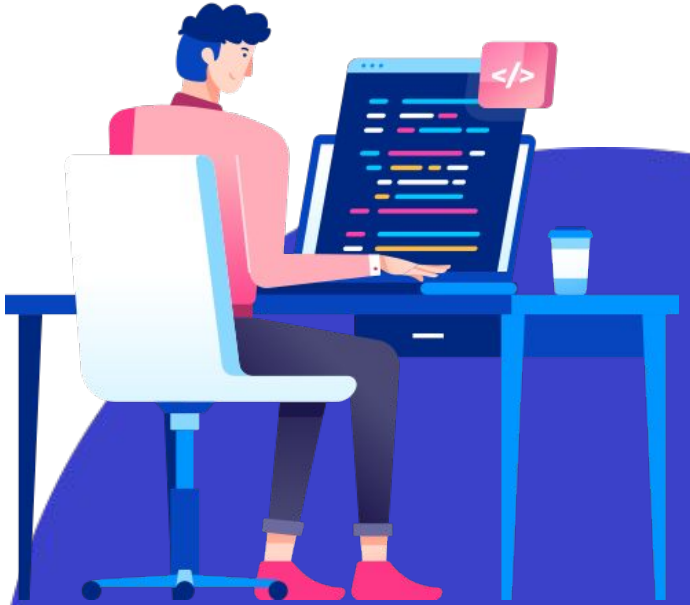
PROGRAMMING FUNDAMENTALS

JS Overview

DIPLOMA IN FULL-STACK DEVELOPMENT

Certificate in **Computing Fundamentals**



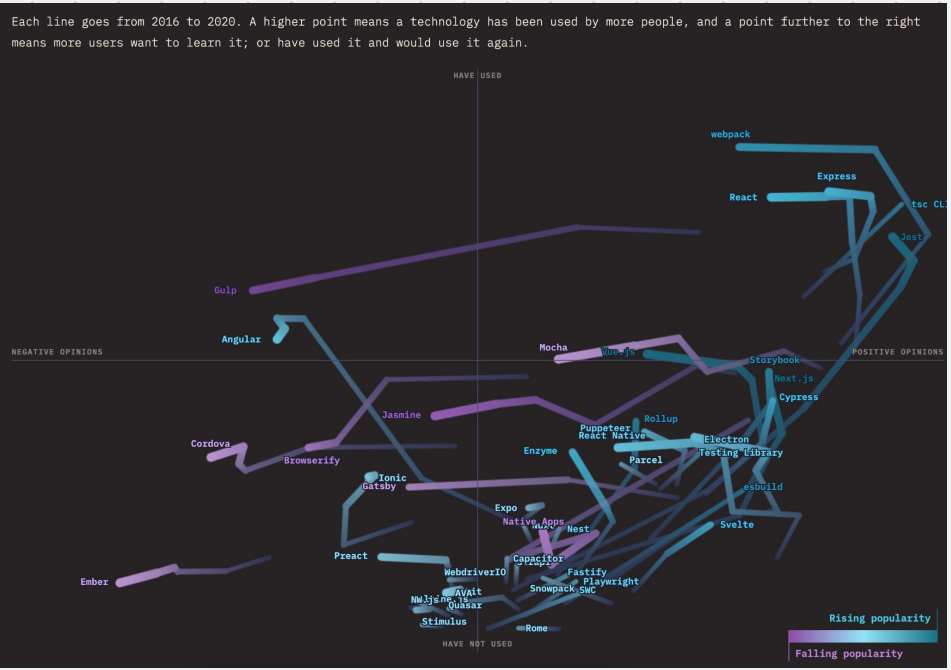
JS Fundamentals

Let’s Start

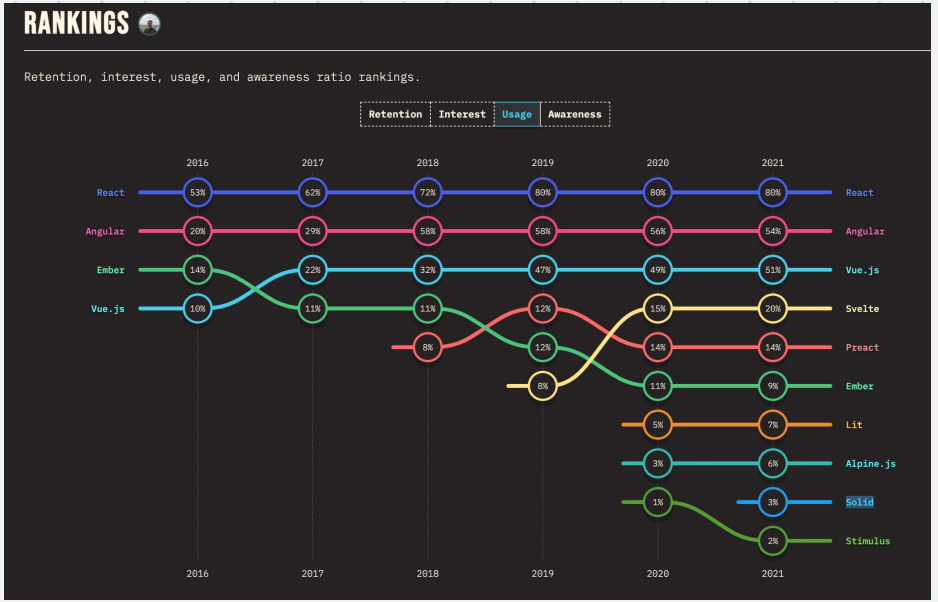
History of JS

- *Always* being confused with Java (two different things)
- Invented by Netscape
- Used to have different versions thank to Microsoft (i.e, the Browser War)
- Chrome standardizes most aspects of JavaScript

State of JS



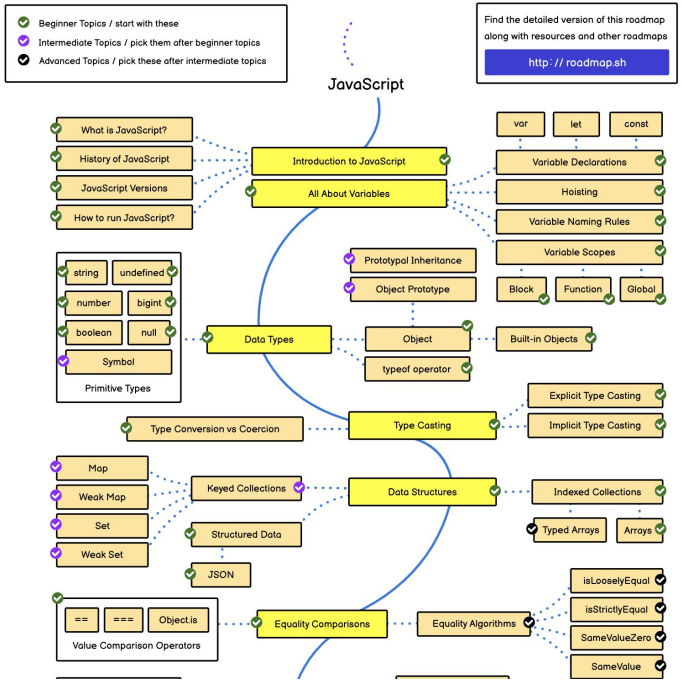
https://2021.stateofjs.com/en-US/libraries

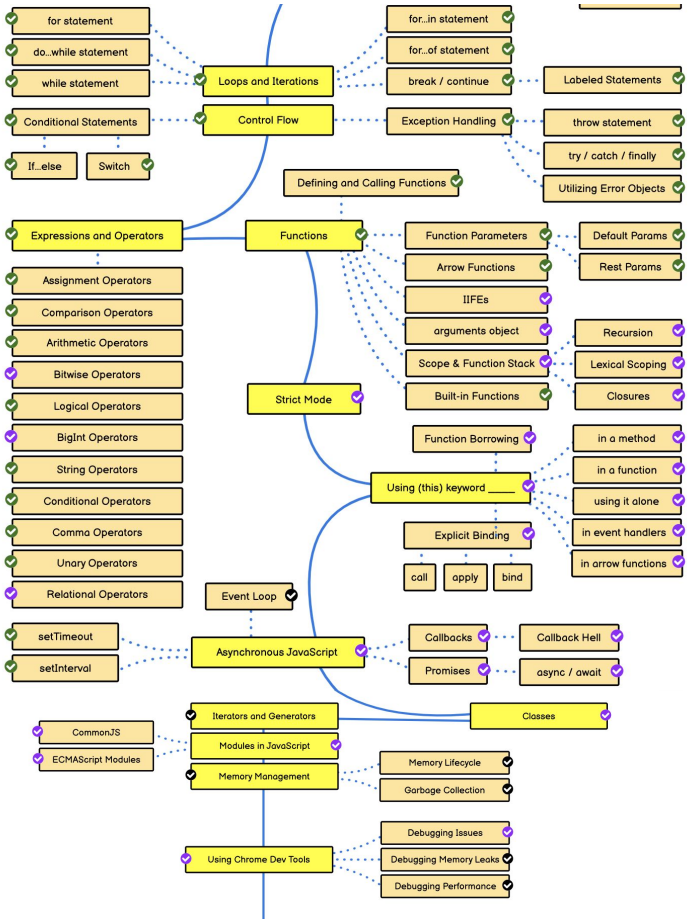


https://2021.stateofjs.com/en-US/libraries/front-end-frameworks

JS Roadmap

https://roadmap.sh/javascript





Two Kinds of JavaScript

BROWSER JAVASCRIPT

- Works on the browser
- Doesn't have all the cutting edge features
- Not all browsers support the same features
- Used **only** for front-end

NODEJS

- Runs on your computer
- It's work like Python
- Cutting edge
- Can be used for both **frontend** and **backend**

Recent Developments

- NodeJS is also used in *mobile app development* thanks to React Native and NativeScript
- You can now develop desktop applications that work on any platform using *Electron*
- The only areas where NodeJS has yet to break in are AI and data science (still the speciality of Python).

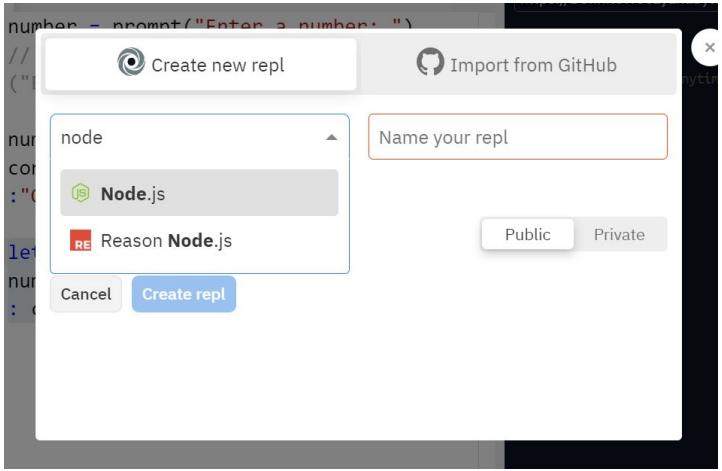
All the things you can build with JavaScript

FLATIRON SCHOOL

Ready for Examples

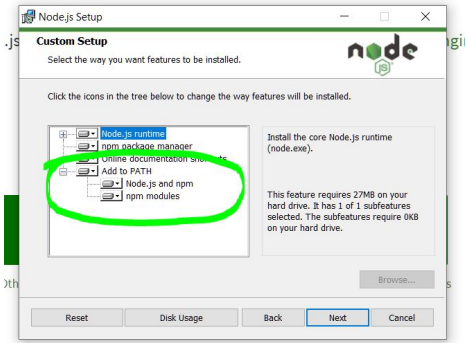
REPL.IT

1. Setup an account for http://www/repl.it
2. When starting a new repl, choose *NodeJs*



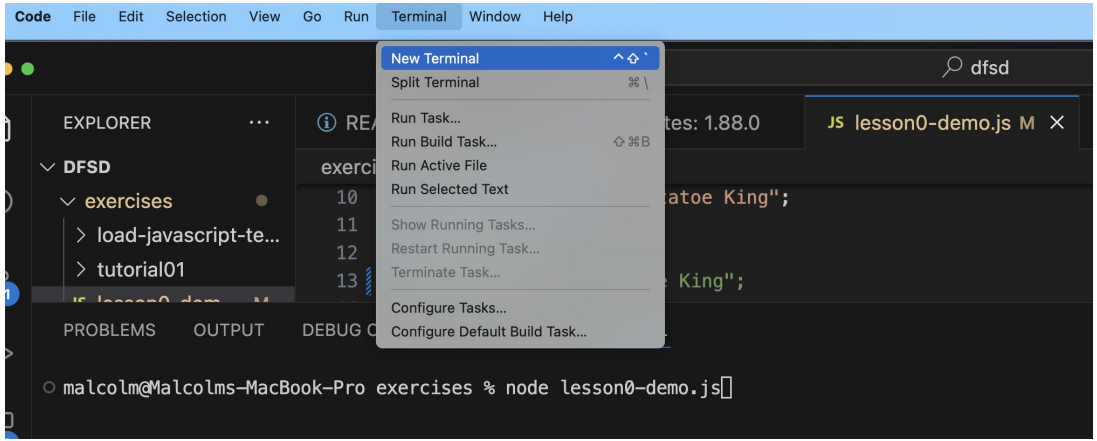
VISUAL STUDIO CODE

1. Download *NodeJS* from https://nodejs.org/en/
2. When installing, make sure to there are no red Xs beside the *Add to Path* (it should look like below)



VS Code Instructions

1. Place all your JS code in a folder
2. Open VS Code
3. Drag the folder in your Explorer or Finder to the code edit area
4. To run the Node code, go to the terminal and type in *node <filename>* and press [ENTER]



Your First JS Program

Start a file named *hello.js* and type in:

```
console.log("Hello World!")
```

Save and run the program.

Browser & Editor

What you need to get started

Choose your **favorite browser** to work in
They all have a great developer **console**

https://www.channelnewsasia.com/world/so-long-internet-explorer-browser-finally-retiring-2747386

Get a Great **Code Editor**

Visual Studio Code

WS

repl.it

JSFIDDLE

Sublime Text

CODEPEN

Code Editors
https://code.visualstudio.com/ ❤️
https://www.jetbrains.com/webstorm/
https://www.sublimetext.com/

Online Editors
https://repl.it ❤️
https://jsfiddle.net/
https://codepen.io/

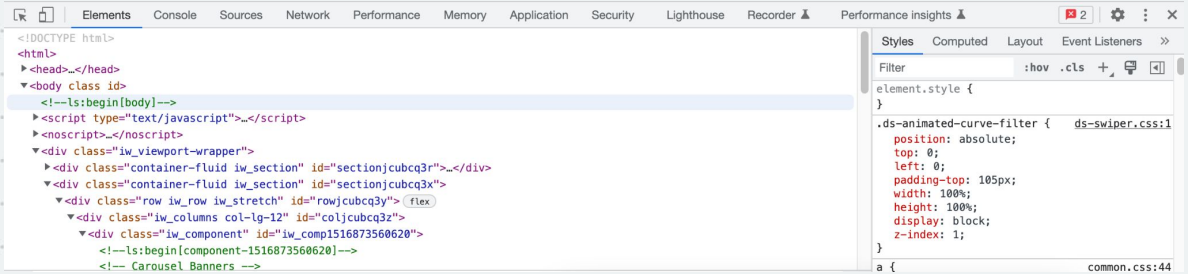
Sign up for repl.it

Join Team:
https://replit.com/teams/join/pvcnsrhwscvibppdyteakxrdgtzwzmcl-modern-frontend

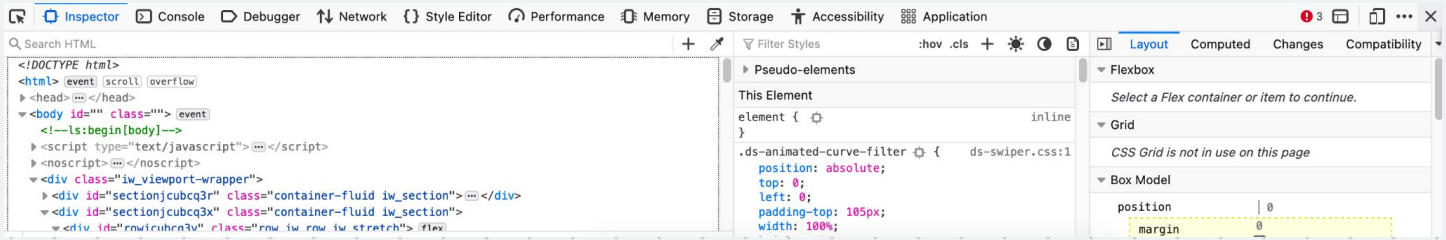
The Console

Developer tools in a browser

Modern day browsers come feature packed. They are equipped with a developer console which helps to facilitate debugging, analysis and enhance the web development experience.



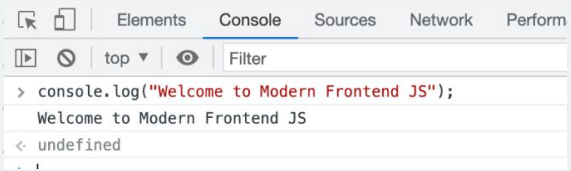
The Chrome Dev Tools
Win: CTRL+SHIFT+J
Mac: Option + Command + J



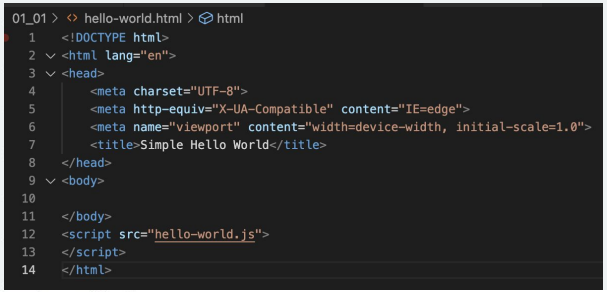
The Firefox Dev Tools
Win: Shift + F2
Mac: Fn + Shift + F2

Running & Load JS

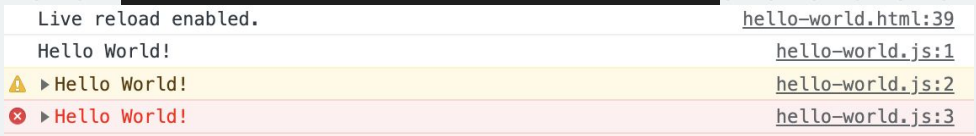
Different ways of setup



In a brower's dev console



In a Repl.it on Shell Terminal (running NodeJS)



Using VSCode with LiveServer extension

We can install NodeJS and setup our terminal to run JavaScript (not covered). We can use repl.it NodeJS support to run in the shell terminal

Programming 101

Never store sensitive information like passwords.. Obviously you won’t want your data to be stolen easily.

```
console.log("Frontend");
console.log("Development");
function display(){
   console.log("Let's build!");
}
display();
```

Three Processes of Data

Input, Process, Output

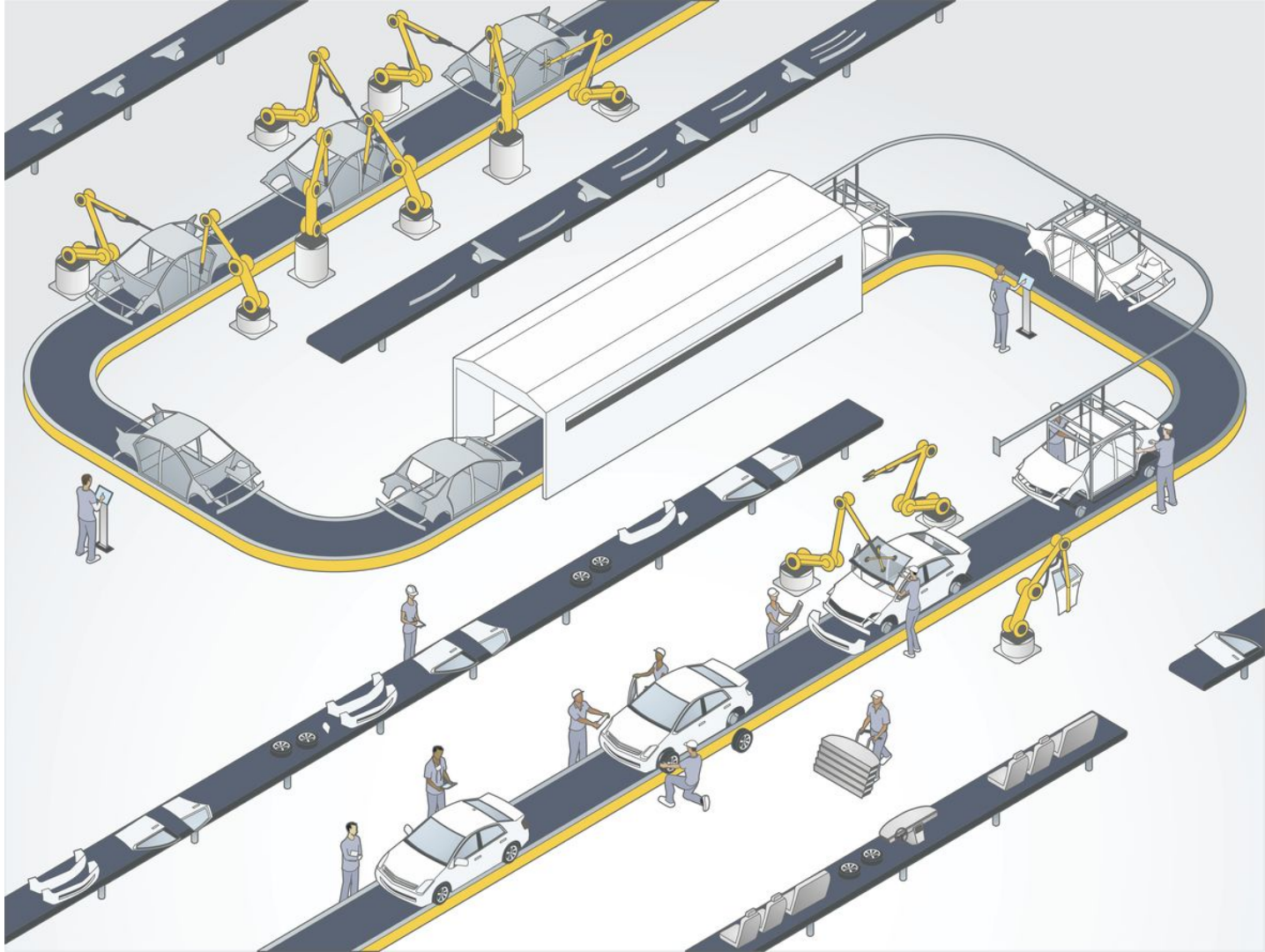
Program acts on Data

Most of the programs (or software) that we use manipulates data:

- Amazon - what do we want to buy and where do we live
- Netflix - what had we match and what we might want to watch next
- Grab - where are we and where do we want to go

Program

Assembly Line of Data

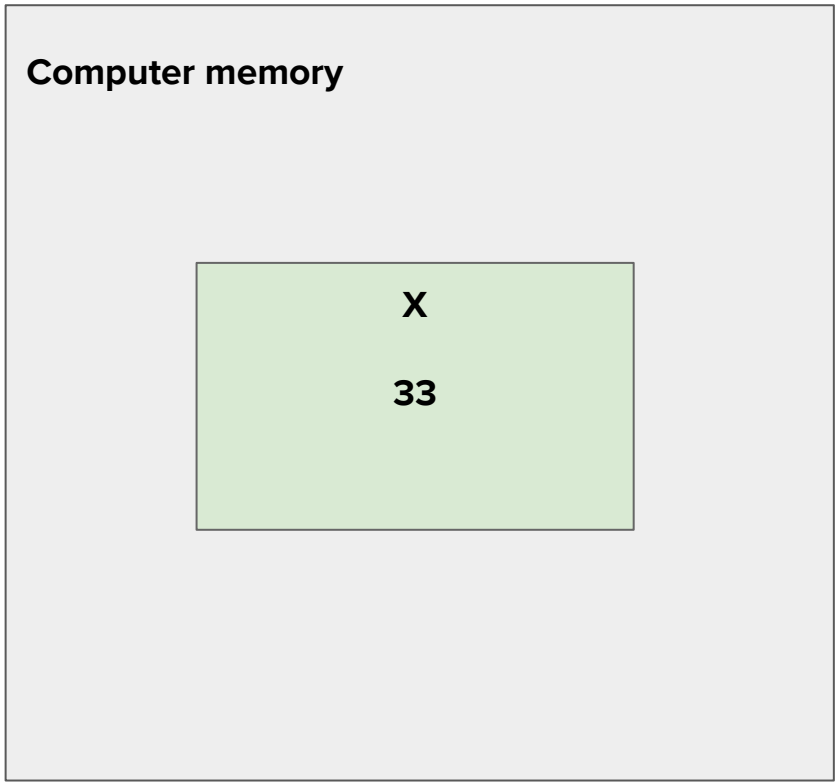


Variables - Store Data

Variables allow us to store data

```
let x = 33;
console.log(x);
```

Assign 33 to the variable x



let defines a variable

We use *let* to define a variable.

A variable can only be defined once*

Of course there are exceptions

```
let x; // just define a variable, but it stores nothing
x = 3; // assign the value 3 to x

let y = 4; // declare variable y and assign 4 to it
```


EQUALS SIGN
ASSIGNMENT OPERATOR
imgflip.com

Suppose we are creating a program where username is required.
To work with username, we'll create a variable and store the **username** in it.

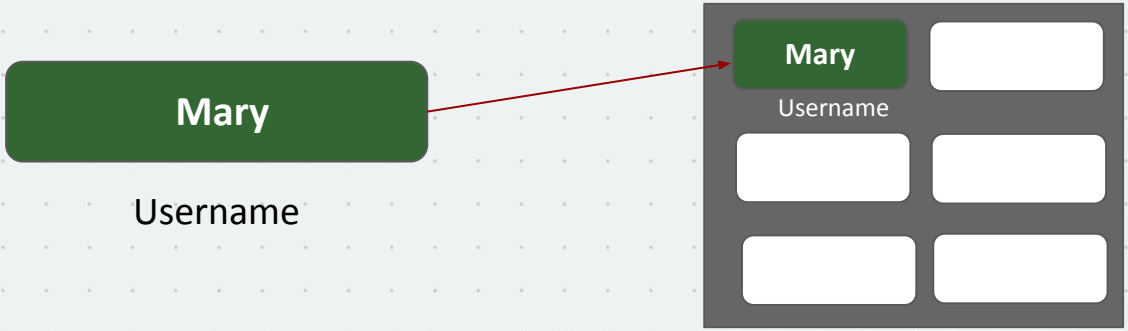
Username

Mary is the data that we'll be using in the program. But we cannot use it directly like "Mary", this way the program will not be maintained properly and code will be not readable.

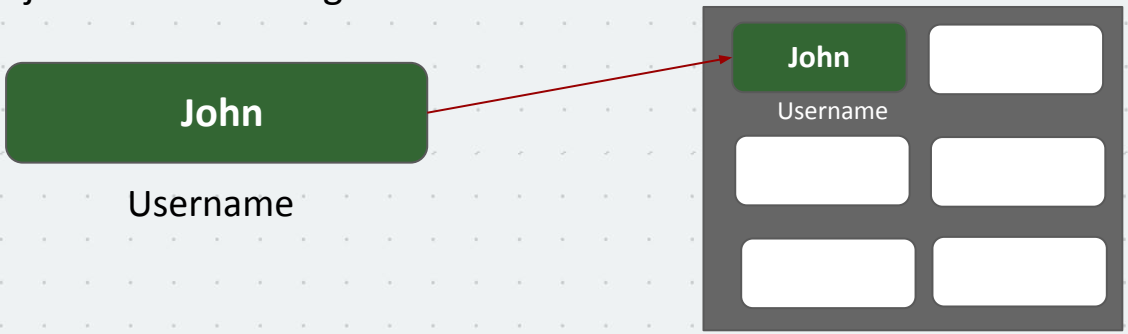
Also, we won't be able to change the user's name easily when needed

You can initialize variable at the time of declaration as well as later in the program

So it is advisable to create a variable and store our data in it.



This way if you want to change the name of the username.
You just need to change the value of username variable.



DEFINITION

A variable is a named container for a ***value***

Used to store, manipulate & retrieve data

The **name** of the variable can be anything so long as it is meaningful.

(However, it must follow the **camelcase** naming convention – to be covered in the lesson on Best Practices.)

Each variable must have a **unique** name.

NGEE ANN POLYTECHNIC

In JavaScript, there are **3 ways** to declare variables

Var

Variables declared with var keyword are either local or global to the function.

If they are declared inside a function, they're local to the function and if declared outside the function then they are global variable.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

Let

Variables declared with let keyword are block-scoped

I.e., they are local to the block they're declared in.
These variables unlike the one with var keyword cannot be redeclared.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

Const

Variables declared with const keyword are the one's that never change their value throughout the program.

If you know that some value to the variable won't change throughout the program, declare them with const.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const

Python

```
quantity = 3
```

C#

```
type variableName = value;
int quantity = 3;
```


```
console.log(quantity);
```

*Instead of var, we are using let (ES6)

Reading Materials:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

https://hackernoon.com/why-you-shouldnt-use-var-anymore-f109a58b9b70

https://wesbos.com/is-var-dead/

Variable Declaration in JS

var

```
function greetA(){
   //variable greeting can be used here
   var greetingA = 'namaste';
   console.log(greetingA);
}
//variable a is local to the function so cannot be used here
console.log(greetingA);//Reference Error

greetA();
```

let

```
//let is a block scoped variable declaration
let greetingB = "Bonjour";
function wish(){
   let greetingWish = "namaste";
   console.log(greetingWish); //will this print out Bonjour?
}

function greetB(){
   let greetingC = "Ni hao";
   console.log(greetingB + " " + greetingC);//namaste Ni hao
   console.log(gretingWish);
//ReferenceError

}
```

const

```
//const: variables can only be declared once
const greetingC = "Bye!";
function greetC(){
   greetingC = "Namaste";//TypeError: you cannot reassign value to const variable
   console.log(greetingC);
}

console.log(greetingC);//Bye!
greetC();
```

Where to Put JS

Where to Put JS

Using the script tag in a HTML doc

JavaScript in body or head: Scripts can be placed inside the body or the head section of an HTML page or inside both head and body.

JavaScript can be placed both inline into an HTML document and as a separate .js file.

How is code rendered & interpreted?

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Hello World</title>
    <script src="hello-world.js"></script>
    <script>
        console.log("Hello World");
    </script>
</head>
<body>
    <script src="hello-world.js"></script>
</body>
<script src="hello-world.js">
</script>
</html>
```

Inline JS

Ideal!

JS is normally used as a frontend scripting platform
For our module, at this point we will use it in a NodeJS environment.
So the HTML part is ignored.

Demo

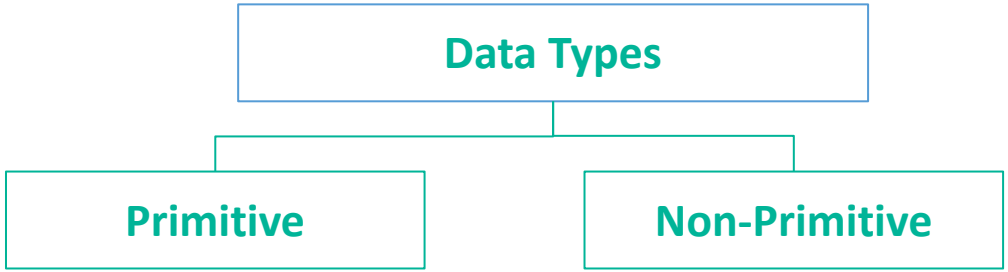
Data Types

Data Types

Data types tells your computer about the type of data you're working with.

Whether it's a number, a string, a decimal or complex types as objects, arrays, etc.

JS is a **dynamic programming language** hence you don't need to specify data type of any variable



1. String
2. Number
3. BigInt
4. Boolean
5. Undefined
6. Null
7. Symbol

1. Object

Example
https://replit.com/@immalcolm/JSDataTypes-Node

The Object data type(non-primitive type) can store collections of data, whereas primitive data type can only store a single value

https://developer.mozilla.org/en-US/docs/Glossary/Primitive

Primitive Data Types’

Primitive data types are the type
of data that contain single value
Eg. Numbers, strings, boolean, etc

Non-Primitive Data Types

Primitive data types are the type
of data that contain single value
Eg. Numbers, strings, boolean, etc

```
/*
[Strings]
String is used to store text. In JS, strings are surrounded by quotes:
- Single Quotes: 'Hello'
- Double Quotes: "Hello"
- Template Literals (Backticks): `Hello`
*/
const name1 = 'ram';
const name2 = "laxman";
const result = `The names are ${name1} and ${name2}`;
```

Using Quotes for Strings

Note that for **String** variable, the value must be enclosed using either the double quotes or the single quotes. (It's more common to use the double quotes).

Example:

```
let item = "durian cake";
let fruit = 'durian';
```

** Notice how the string is placed inside quote marks

*if start with double quotes, end with double quotes
If start with single quotes, end with single quotes

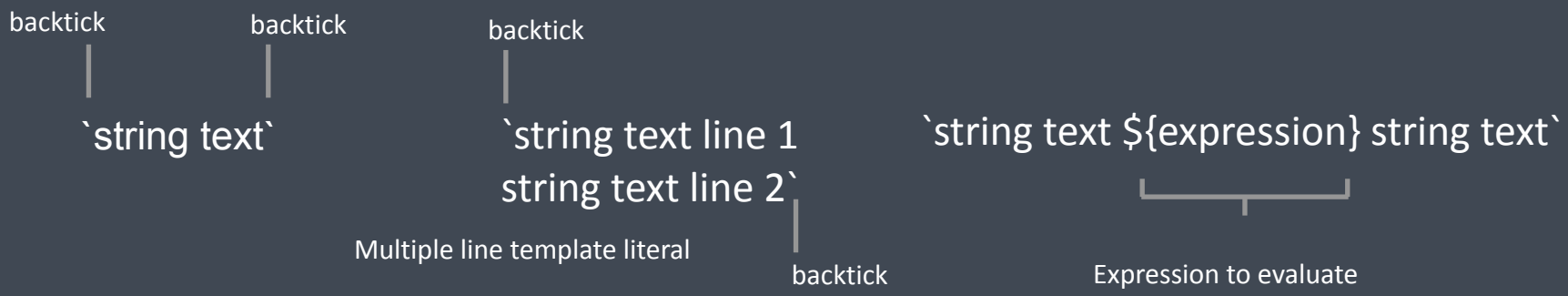
✓ "hello"
✓ 'hello'

X "hello'
X 'hello"

✓ " "
✓ ' '

X “ ”
X ‘ ’

Template literals are enclosed by the back-tick (` `)



*Reference Article
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

Usual method of writing strings

```
var age = 21;
console.log('My age is ' + age);
```

Substitution using Template String

```
var age = 21;
console.log(`My age is ${age}`);
```

Template String

- ✓ String Substitution
- ✓ Cleaner
- ✓ More Readable

```
/*
[Number]
Number represent integer and floating numbers (decimals and exponentials).
*/
const number1 = 3;
const number2 = 3.433;
const number3 = 3e5 // 3 * 10^5

/*
[JS BigInt]
A BigInt number is creating by appending 'n' to the end of an integer
*/
//Big Int value
const value1 = 482849238938293829389 23n;

//adding two big integers
const result1 = value1 + 1n;
console.log(result1);
```

```
/*
[JS Boolean]
This data type represents logical entities.
Boolean represents one of two values: true or false.
It is easier to think of it as a yes/no
*/
const dataChecked = true;
const valueCounted = false;

/*
[Undefined]
The undefined data type represents value that is not assigned.
If a variable is declared but the value is not assigned,
then the value of that variable will be undefined
*/
let userName;
console.log(userName);

/*
[null]
In JS, null is a special value that represents empty or unknown value.
*/
const num = null;
//the above code suggest that the number variable is empty
```

Type Conversion

Type Conversion

Suppose you’re finding sum of two variables - one is of **integer data type** and other is of **float data type.** To get the correct solution, what would you do?

Type Conversion

You could convert either type of variable into another to find the solution

Type conversion is all about converting variable of one type into variable of another type.

We do this to make variables compatible to each other.

Types of type conversion

Implicit conversion
Automatically converting data types.

Explicit conversion
Manually converting data types

Demo Type Conversion

https://replit.com/@immalcolm/TypeConversion

Types of type conversion

Implicit Conversion

```
//[implicit type conversion]
let result;
result = '21' + 2;
console.log(result); //"212"

result = '5' - 2;
console.log(result); //3

result = '4' - true;
console.log(result); //3
```

Explicit Conversion

```
//[explicit type conversion]
//converting numbers
result = Number('324-1');
console.log(result); //32.4

result = Number("45");//45
console.log(result);

result = Number(true);//1
console.log(result);

result = Number(false);
console.log(result); //0

result = parseInt('20.01');
console.log(result);//20

result = parseFloat('20.01');
console.log(result);

//converting to strings
result = String(45);//"45"
console.log(result);

result = String(true);//"true"
console.log(result);
```

Demo Type Conversion

https://replit.com/@immalcolm/TypeConversion

How above converting to false booleans?

Types of type conversion

Converting to Boolean

```
//converting to boolean
result = Boolean(45);//True
console.log(result);

result = Boolean(0);//False
console.log(result);

result = Boolean(" ");//False
console.log(result);
```

Demo Type Conversion

https://replit.com/@immalcolm/TypeConversion

Weakly Typed

```
let superNum = 42; // foo is now a number
superNum = "bar"; // foo is now a string
superNum = true; // foo is now a boolean

const sillyNum = 42; // foo is a number

// JavaScript coerces foo to a string, so it can be concatenated with the other operand
const nowANum = sillyNum + "1";

console.log(nowANum); // 421
//is it a string or number now?
console.log(typeof(nowANum)); // 421
```

The Object data type(non-primitive type) can store collections of data, whereas primitive data type can only store a single value

https://developer.mozilla.org/en-US/docs/Glossary/Primitive

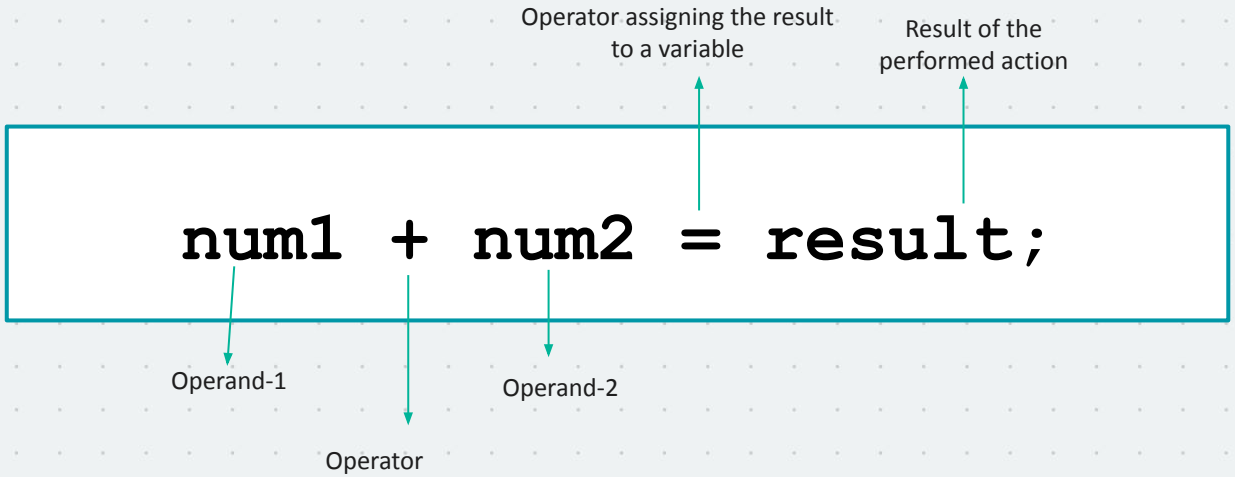
Comparison & Logical Operators

Operators

Operators helps you to build that logic easily. They help us manipulate the data and perform the required tasks

Operators are symbols to perform specific operation operands. The entities on which you perform operations are **operands**

Operators



Here, the plus symbol is performing addition operation on the values in the variable **num1** and **num2**. It stores the result in variable `**result**`

Operators

Operators and their description

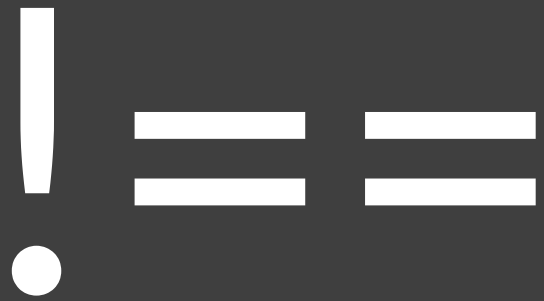
Operator	Symbol	Purpose
Arithmetic	+, -, *,/,%,etc	Perform mathematical operations on operands
Logical	&&, || , !	Perform logical operation & returns boolean
Assignment	=, +=, -=, *= etc	Assigns values to variable
Comparison	>, <, >=, <=, ==, ===, etc	Compares two/more values & returns boolean value
Ternary	?	Assigns value to variable based on condition & is an alternative to if-else
Bitwise	&, |, ^, ~ etc	Perform operation on one-bit or one operand & binary representation

NEW!

STRICT EQUAL TO

This operator compares two values to check that both the **data type and value** are the same

'8' === 8 returns **false**
Because they are not the same data type or value.
'8' === '8' returns **true**
Because they are the same data type and value.



NEW!

STRICT NOT EQUAL TO

This operator compares two values to check that both the **data type and value** are ***not*** the same

'8' !== 8 returns **true**
Because they are ***not*** the same data type or value.
'8' !== '8' returns **false**
Because they are the same data type and value.

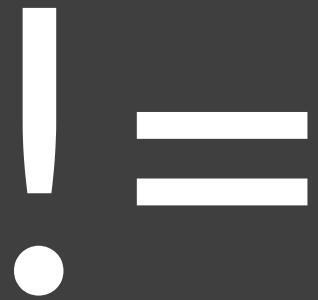
IS EQUAL TO

This operator compares two values (numbers, strings, or Booleans) to see if they are the same.

'Hello' == 'Goodbye' returns **false** because they are not the same value

5 == '5' returns **true**

Because they have the **same value** although their **data types are different**



IS NOT EQUAL TO

This operator compares two values (numbers, strings, or Booleans) to see if they are ***not*** the same.

'Hello' != 'Goodbye' returns **true** because they are not the same string
'Hello' != 'Hello' returns **false** Because they are the same string

GREATER THAN

This operator checks if the number on the left is ***greater than*** the number on the right.

8 > 7 returns **true**

7 > 8 returns **false**

LESSER THAN

This operator checks if the number on the left is ***greater than*** the number on the right.

7 < 8 returns **true**

8 < 7 returns **false**

GREATER THAN OR EQUAL TO

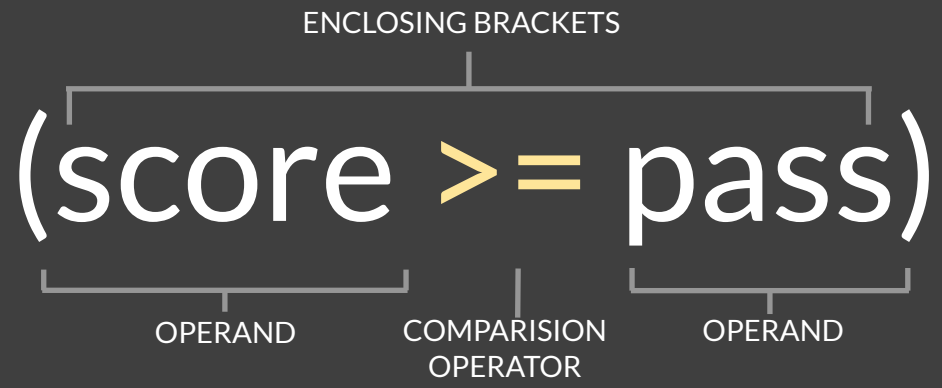
This operator checks if the number on the left is ***greater than or equal*** to the number on the right.

8 >= 7 returns **true**
7 >= 8 returns **false**
8 >= 8 returns **true**

LESSER THAN OR EQUAL TO

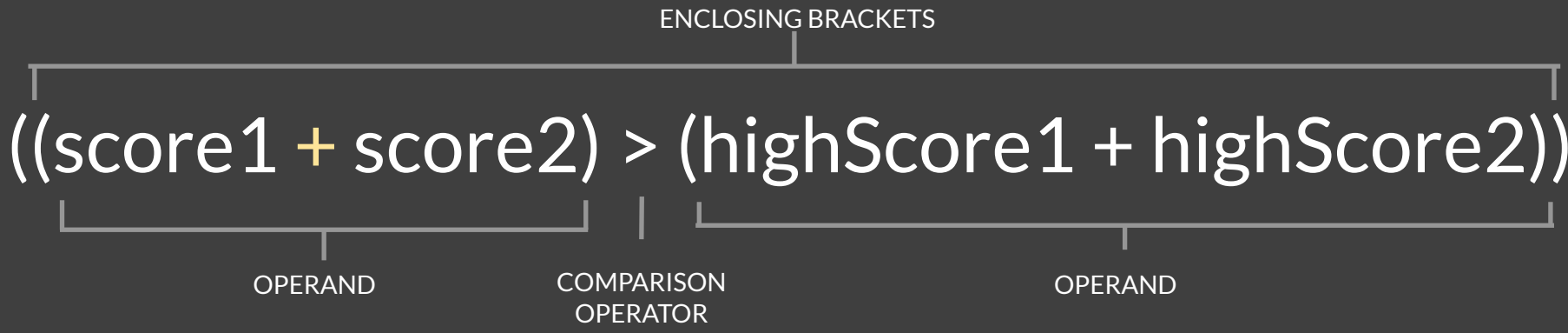
This operator checks if the number on the left is ***lesser than or equal*** to the number on the right.

8 <= 7 returns **false**
7 <= 8 returns **true**
8 >= 8 returns **true**



Example: Checking if student passed

```
let pass = 50; //passing mark
let score = 85; //actual score

//check if student passed
if(score >= pass){
  console.log("You passed!");
}
```



The operand does not have to be a single value or variable name.

An operand can be an *expression (because each expression evaluates into a single value)*

To perform calculations on numbers, we can use the **arithmetic operators**

Example: Using the modulo operator

```
let num1 = 11;
let num2 = 3;
let result;

// Using Modulo operator
result = num1 % num2;
console.log("Result: " + result);
```

Output:

Result: 2

Check out JS Math Library

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/round

To round off numbers, we can use **Math.round(x);**

X is a parameter (required)

** rounds number to the nearest integer

Example: Using the Math.round() function

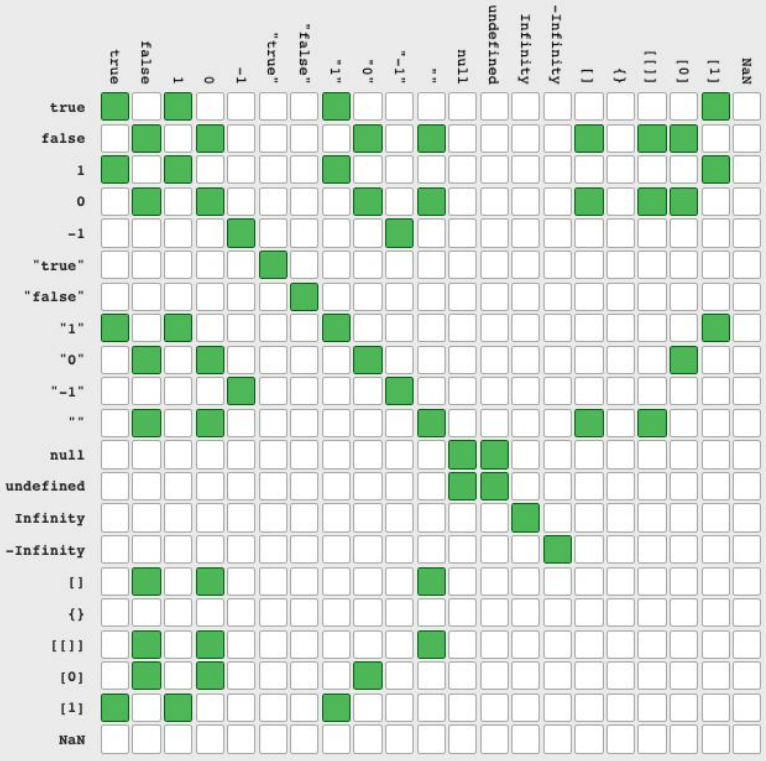
```
let num1 = 6.5;

console.log ("Rounded Result:" +
Math.round(num1)");
```

Output:
Rounded Result: **7**

Truthy

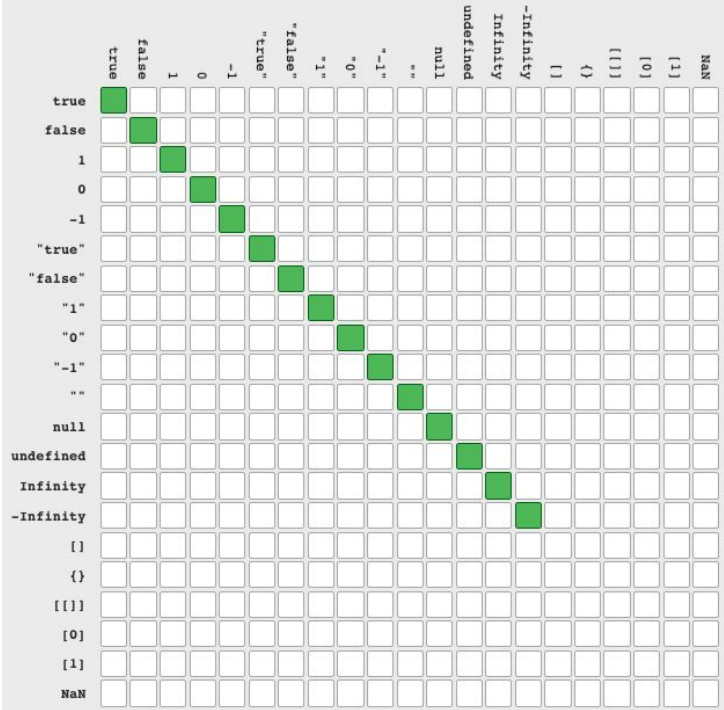
== (negated: !=)

When using two equals signs for JavaScript equality testing, some funky conversions take place.



=== (negated: !==)

When using three equals signs for JavaScript equality testing, everything is **as is**. Nothing gets converted before being evaluated.



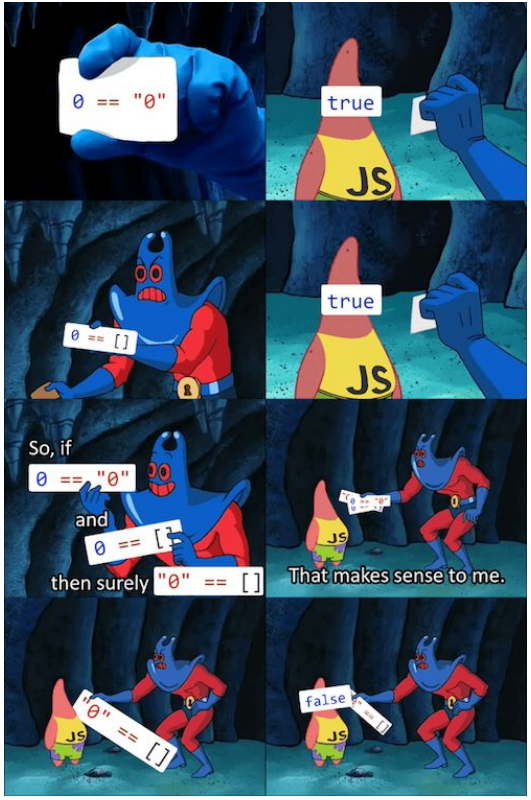
https://dorey.github.io/JavaScript-Equality-Table/

Evaluating ifs

A standard IF statement. If(*value*) {/*- green -*/} else { /*- white -*/ }

Note: This row does not match up with any of the rows in the other table.

`true`	■	if (true) { /* executes */ }
`false`		if (false) { /* does not execute */ }
`1`	■	if (1) { /* executes */ }
`0`		if (0) { /* does not execute */ }
`-1`	■	if (-1) { /* executes */ }
`"true"`	■	if ("true") { /* executes */ }
`"false"`	■	if ("false") { /* executes */ }
`"1"`	■	if ("1") { /* executes */ }
`"0"`	■	if ("0") { /* executes */ }
`"-1"`	■	if ("-1") { /* executes */ }
`""`		if ("") { /* does not execute */ }
`null`		if (null) { /* does not execute */ }
`undefined`		if (undefined) { /* does not execute */ }
`Infinity`	■	if (Infinity) { /* executes */ }
`-Infinity`	■	if (-Infinity) { /* executes */ }
`[]`	■	if ([]) { /* executes */ }
`{}`	■	if ({}) { /* executes */ }
`[[]]`	■	if ([[]]) { /* executes */ }
`[0]`	■	if ([0]) { /* executes */ }
`[1]`	■	if ([1]) { /* executes */ }
`NaN`		if (NaN) { /* does not execute */ }



https://dorey.github.io/JavaScript-Equality-Table/

Logical Operators

DEFINITION

Logical Operators allow you to compare the results of more than one comparison operator.

&& ||

I will give you an Apple **AND** Orange

What do you have?

I will give you an Apple **OR** Orange

What do you have?

If you give me an Apple **AND** an Orange, I will give you 5 dollars.

What do you have to give me for $5?

If you give me an Apple **OR** an Orange, I will give you 5 dollars.

What do you have to give me for $5?

I will give you an Apple **AND** Orange

What do you have?

If you give me an Apple **AND** an Orange, I will give you 5 dollars.

What do you have to give me for $5?

In other words, for **AND** logical operator, both conditions must be true before you get your $5.

I will give you an Apple **OR** Orange

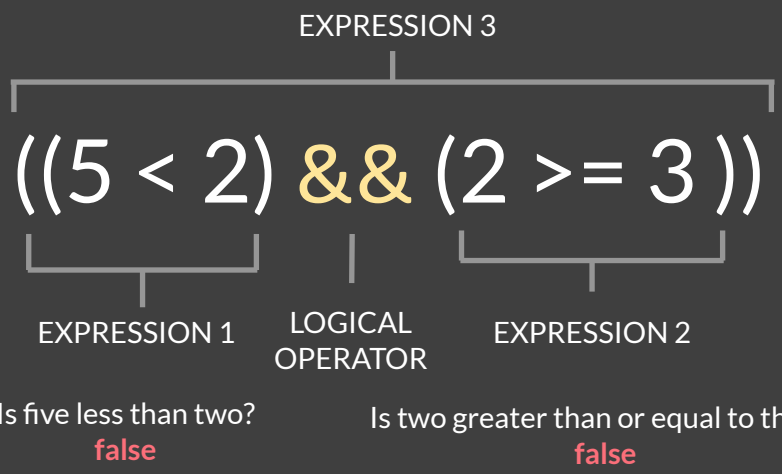
What do you have?

If you give me an Apple **OR** an Orange, I will give you 5 dollars.

What do you have to give me for $5?

For **OR** logical operator, so long as one of the conditions is true, you will get your $5.

(Note: For **OR** logical operator, you will still get your $5 even when both conditions are true. In other words, I will be most glad to receive both Apple and Orange, even though one of them is sufficient.)



The expressions on the left and the right both use comparison operators, and both return false.

The third expression uses a logical operator (rather than a comparison operator)
The logical **AND '&&'** operator checks to see whether both expressions on either side of it return **true** (In this case they do not, so it evaluates to false)

&&

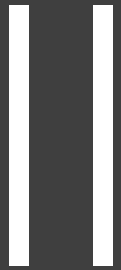
LOGICAL AND

This operator test **more than one** condition.

((2 < 5) && (3 >= 2))
returns **true**

If both expressions evaluate to true then the expression returns true. If just one of these returns false, then the expression returns false.

true && true returns **true**
true && false returns **false**
false && true returns **false**
false && false returns **false**



LOGICAL OR

This operator test **at least one** condition.

((2 < 5) || (3 >= 2))
returns **true**

If both expressions evaluate to true then the expression returns true. If just one of these returns false, then the expression returns false.

true ||true returns **true**
true || false returns **true**
false || true returns **true**
false || false returns **false**

NOT

This operator inverts true or false values

(!true)
returns **false**

Operator simply inverts the value contained within

Casting

Some Good practises to adopt

Use == if you want to test whether the two things being compared are equal

```
console.log(99 == "99"); // true
console.log(0 == false); // true
```

Use === if you want to test whether the two things being compared are of the **same type and value**

```
console.log(99 === "99"); // false
console.log(0 === false); // false
```

***Type Coercion** Happening
Converts Data type to another

*** Better & Safer**

Basically use === if you are not sure of your data types. It is also preferred in the long run.
Use == for quick testing and when you are sure of your data types.

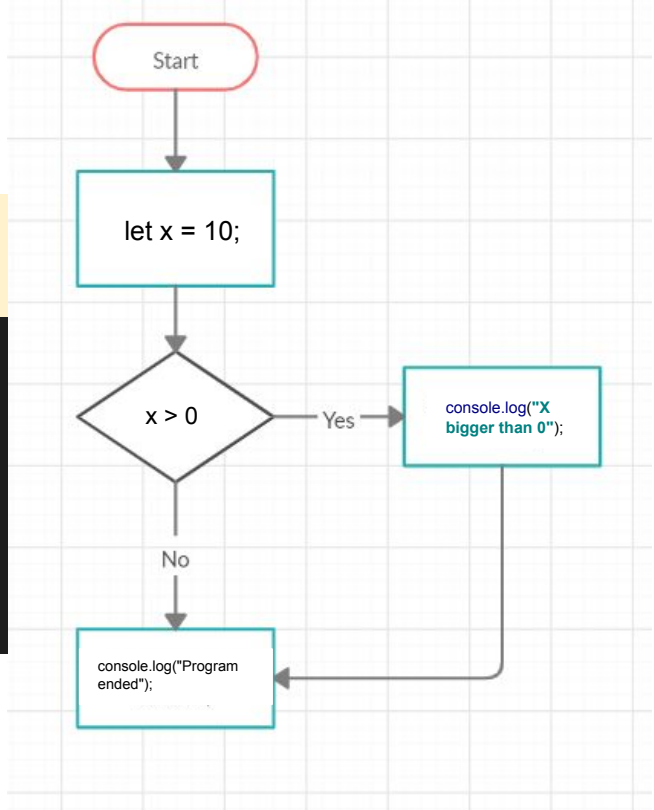
Branching

Visualising Ifs

We can use JavaScript to run certain code only if a condition is met:

If this conditional operator result is true, then do everything indent under the **if** statement

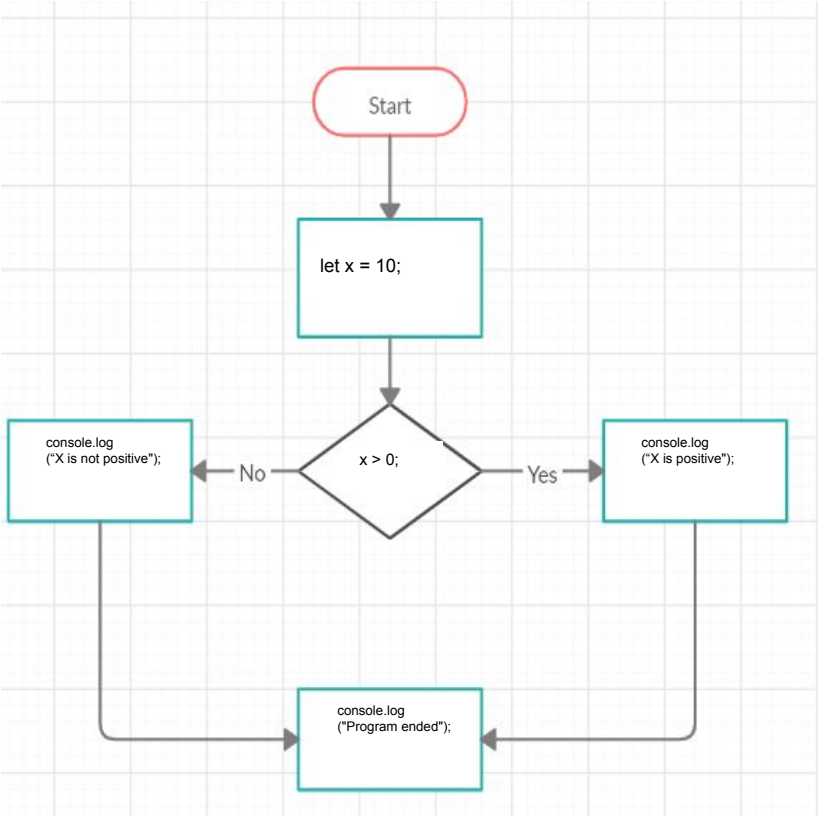
```
let x = 10;
if (x > 0) {
    console.log("x is bigger than 0");
}
console.log("Program ended");
```



Visualising If..else

We can have another branch of code to execute if the condition is **not met.**

```
let x =10;
if (x > 0) {
    console.log("X is positive");
} else {
    console.log("X is not positive");
}
console.log("Program Ended");
```

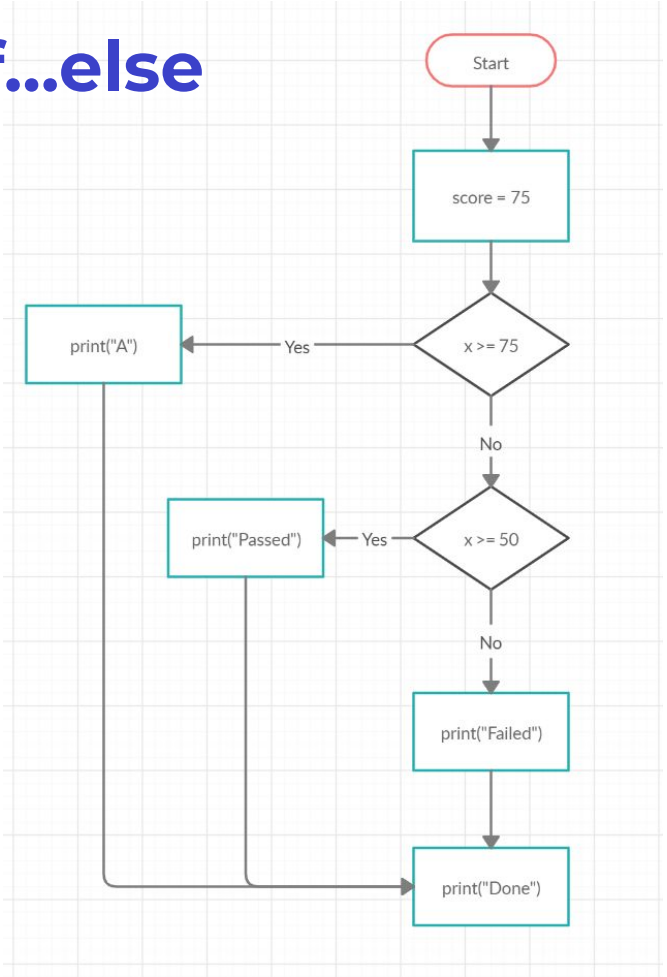


Visualising If..else if...else

We can *chain* a if with an **else if**

An else if will only be considered if the preceding if or else if is **false**

```
let score = 75;
if (score >= 75) {
    console.log("Your grade is A");
} else if (score >=50) {
    console.log("You passed");
} else {
    console.log("You failed");
}
console.log("Done");
```



Summary

- Basics of writing JavaScript code
- concept of variables was introduced, the use of var, let, and const
- Different data types in JavaScript (such as string, number, and boolean)
- Operators and Basic Arithmetic
- Control structures, including if-else statements and switch cases
- Learning how to debug!