Javascript Basics

Created By Varishtha Anand Patni (@thwarted_pentester)

Let's Start:

> ***Recommended (VS CODE editor)

- To link JS file to HTML

```
<body>

  <script src="script.js"></script>

</body>
```

#Topic-1 **Variable and Operators**

> **Variables**

- There are 3 types of variables in JavaScript: **var, let,** & **const**.
- The var (variable) keyword was originally the only variable available, but thanks to the upgrade to ECMAScript 6 back in 2015, which is the specification that JavaScript conforms too, we now have multiple ways of declaring a variable or data types.

Quick Overview:

- **let**: If a variable is going to be reassigned later within the application, this is the ideal variable type to use.
- **var**: It's better to use either let or const for variables, but this variable type will still work and is still used in applications to this day. This variable can be updated and re-declared.
- **const**: If the variable will never change or won't be reassigned anywhere else in the application, this keyword is the best option.

Good things to remember:

```
The **var** variable is globally scoped and can be updated and re-declared.
The **let** variable is block-scoped and can be updated but not re-declared.
The **const** variable is block-scoped and cannot be updated or re-declared.
```

Global Scope: A variable declared outside a function. This means all scripts and functions on a web application or webpage can access this variable.

Block Scope: A variable declared inside a block. This means we can use these variables inside of loops, if statements, or other declarations within curly brackets and have them be only used for that declaration instead of the entire application having access to it.

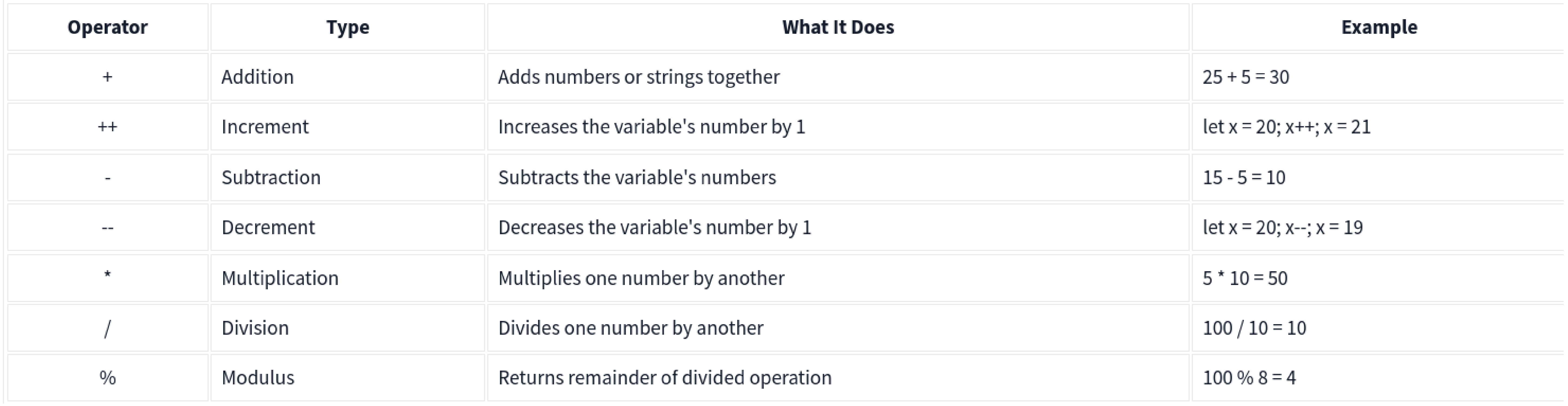
```
Examples:

var variableOne = 'Linus Torvalds';

let variableTwo = 50;
```

```
const variableThree = 'Creator of the Linux Kernel';
```

- Main data types that you can store within a variable:

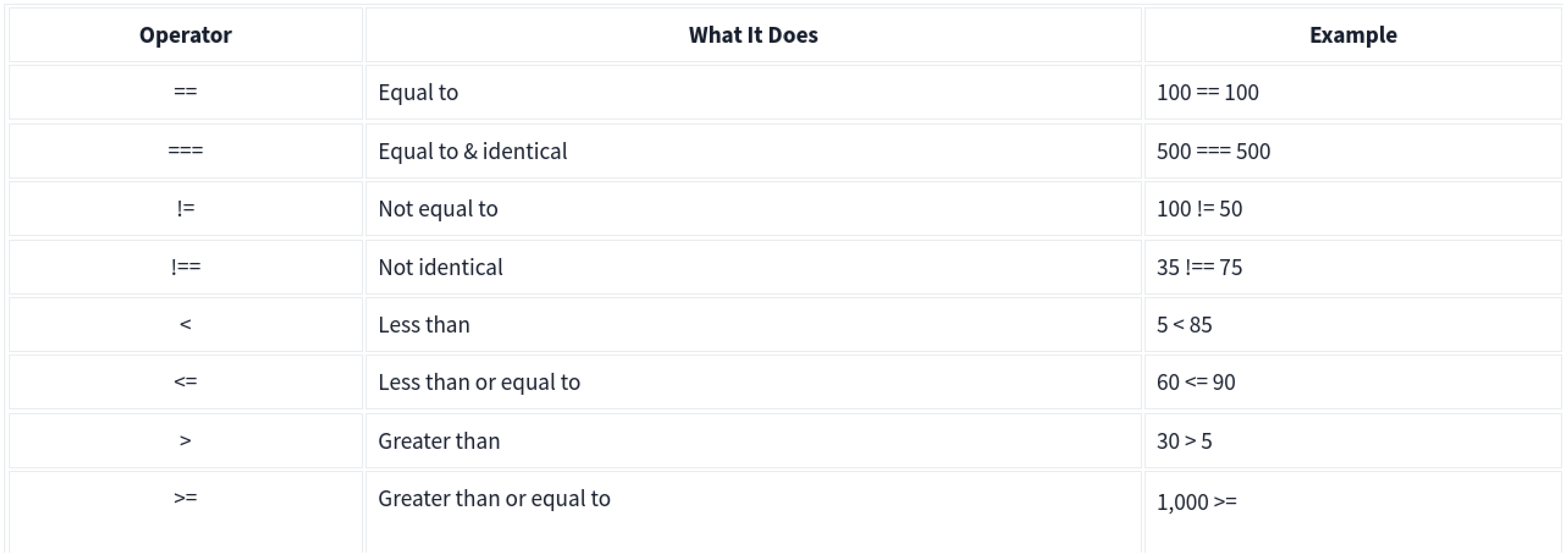
Data Type	Example
Number	200
String	'Neo'
Boolean	True or False

- **Arithmetic Operator:**



Operator	Type	What It Does	Example
+	Addition	Adds numbers or strings together	25 + 5 = 30
++	Increment	Increases the variable's number by 1	let x = 20; x++; x = 21
-	Subtraction	Subtracts the variable's numbers	15 - 5 = 10
--	Decrement	Decreases the variable's number by 1	let x = 20; x--; x = 19
*	Multiplication	Multiplies one number by another	5 * 10 = 50
/	Division	Divides one number by another	100 / 10 = 10
%	Modulus	Returns remainder of divided operation	100 % 8 = 4

- **Comparsion Operator:**



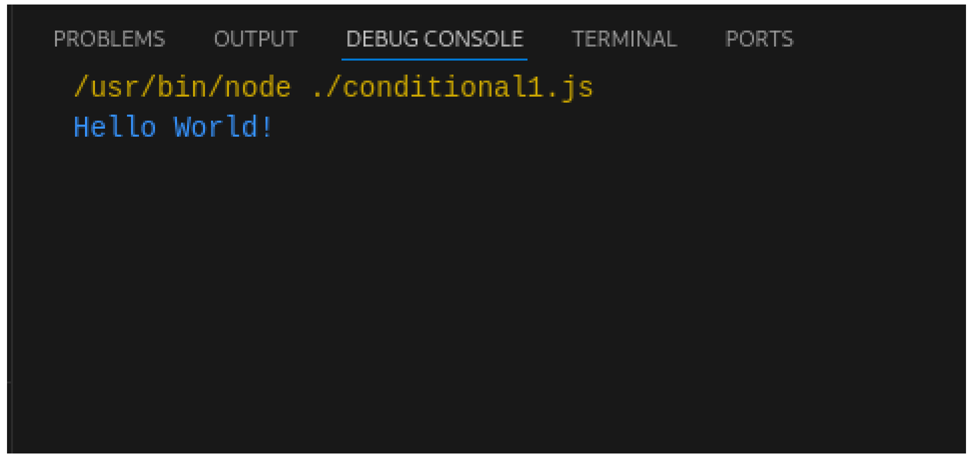
Operator	What It Does	Example
==	Equal to	100 == 100
===	Equal to & identical	500 === 500
!=	Not equal to	100 != 50
!==	Not identical	35 !== 75
<	Less than	5 < 85
<=	Less than or equal to	60 <= 90
>	Greater than	30 > 5
>=	Greater than or equal to	1,000 >=

#Topic-2 Conditional Statements

- Code

```
if (5 === 5) {

console.log('Hello World!'); // Prints Hello World! to the console

};
```

- Output

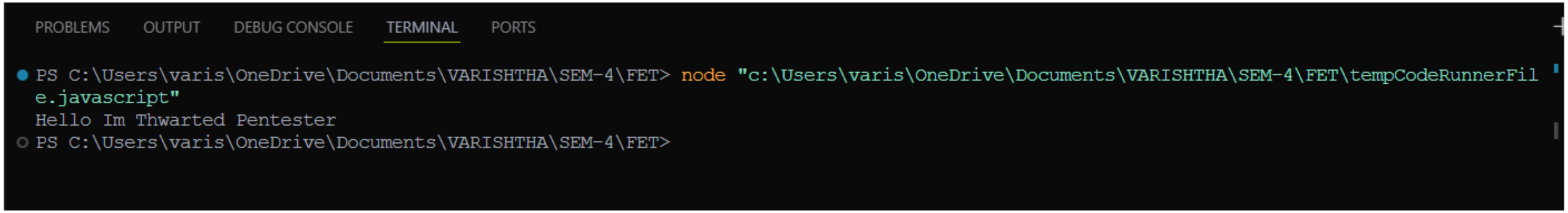


```
/usr/bin/node ./conditional1.js
Hello World!
```

- Code

```
if (5 === 10) {

console.log('Hello World!'); // Skips this code

} else if (10 === 10) {

console.log('Hello World!'); // Prints Hello World! to the console



} else {

console.log('ERROR ERROR ERROR');

};
```

- Output



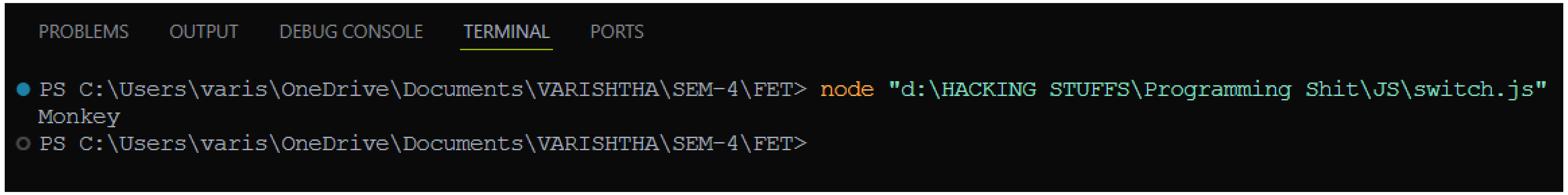
```
PS C:\Users\varis\OneDrive\Documents\VARISHTHA\SEM-4\FET> node "c:\Users\varis\OneDrive\Documents\VARISHTHA\SEM-4\FET\tempCodeRunnerFil
e.javascript"
Hello Im Thwarted Pentester
PS C:\Users\varis\OneDrive\Documents\VARISHTHA\SEM-4\FET>
```

> ***Switch Cases***: If you need to test multiple conditions, then most of the time switch cases are best for optimization and readability within your code. If, else if, else statements and switch cases can both do similar tasks, but switch cases are better for performing multiple different conditions.

- Code

```
const animal = 3;

switch (animal) {
  case 1:
    console.log('Cow');
    break;
  case 2:
    console.log('Chicken');
    break;
  case 3:
    console.log('Monkey');
    break;
  default:
    console.log('Animal?');
}
```

- Output



```
● PS C:\Users\varis\OneDrive\Documents\VARISHTHA\SEM-4\FET> node "d:\HACKING STUFFS\Programming Shit\JS\switch.js"
  Monkey
○ PS C:\Users\varis\OneDrive\Documents\VARISHTHA\SEM-4\FET>
```

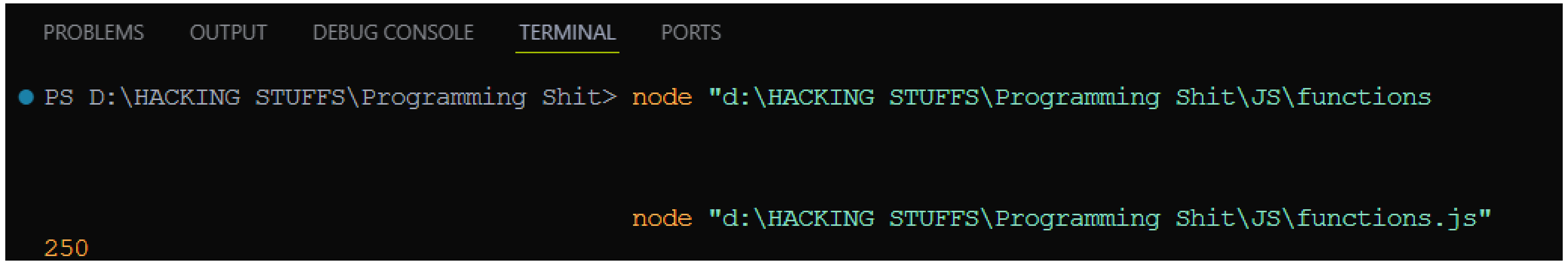
#Topic-3 **Functions**

> Functions are one of the most vital parts of programming.

- Code

```
const func = (a, b) ⇒ {

    let nums = a * b;

    console.log(nums); // Outputs 250
}
func(25, 10);
```

- Output



```
● PS D:\HACKING STUFFS\Programming Shit> node "d:\HACKING STUFFS\Programming Shit\JS\functions


                                        node "d:\HACKING STUFFS\Programming Shit\JS\functions.js"
  250
```

#Topic-4 **Objects & Array**

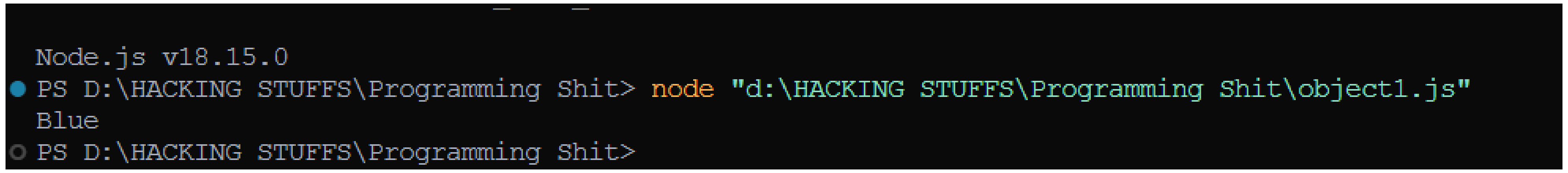
Object

> The most important thing about objects is to remember that they're just another variation of variables.

- Code

```
var choosePill = {

    pillOne: 'Red',

    pillTwo: 'Blue',

    numberofpills: 2

}

    var choice = choosePill.pillTwo; //This will access the Objects property

    console.log(choice)
```

- Output



```
Node.js v18.15.0
PS D:\HACKING STUFFS\Programming Shit> node "d:\HACKING STUFFS\Programming Shit\object1.js"
Blue
PS D:\HACKING STUFFS\Programming Shit>
```

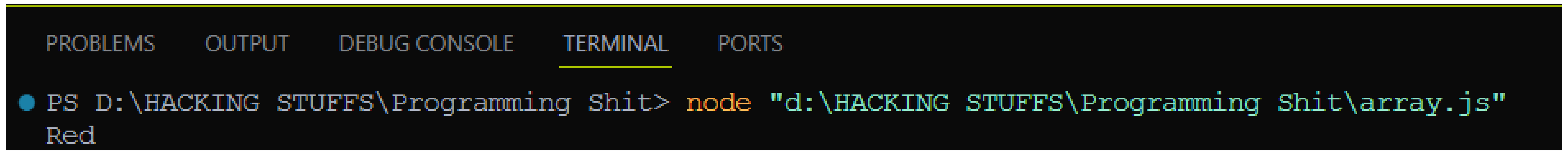
Array

> Arrays are fairly similar to objects, they have different stored values and syntax, but can be used for almost anything.

- Code

```
var choosePill = ['Red', 'Blue', 2];
var choice = choosePill[0];
console.log(choice);
```

- Output

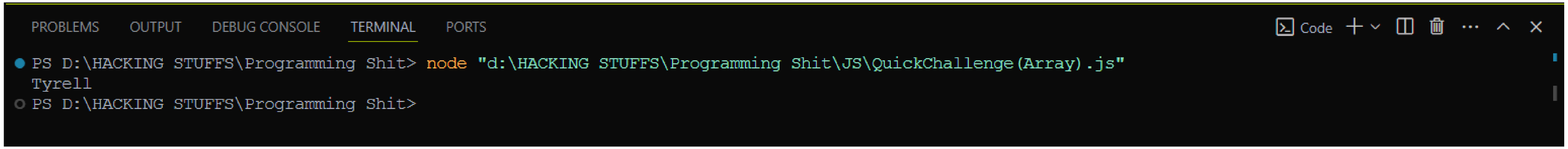


```
PS D:\HACKING STUFFS\Programming Shit> node "d:\HACKING STUFFS\Programming Shit\array.js"
Red
```

- Code

```
var mrRobot = ['Elliot', 'Angela', 'Tyrell', 'Darlene'];

let character = mrRobot[2];

console.log(character); // What is the output?
```

- Output



```
PS D:\HACKING STUFFS\Programming Shit> node "d:\HACKING STUFFS\Programming Shit\JS\QuickChallenge(Array).js"
Tyrell
PS D:\HACKING STUFFS\Programming Shit>
```

#Topic-5 Loops

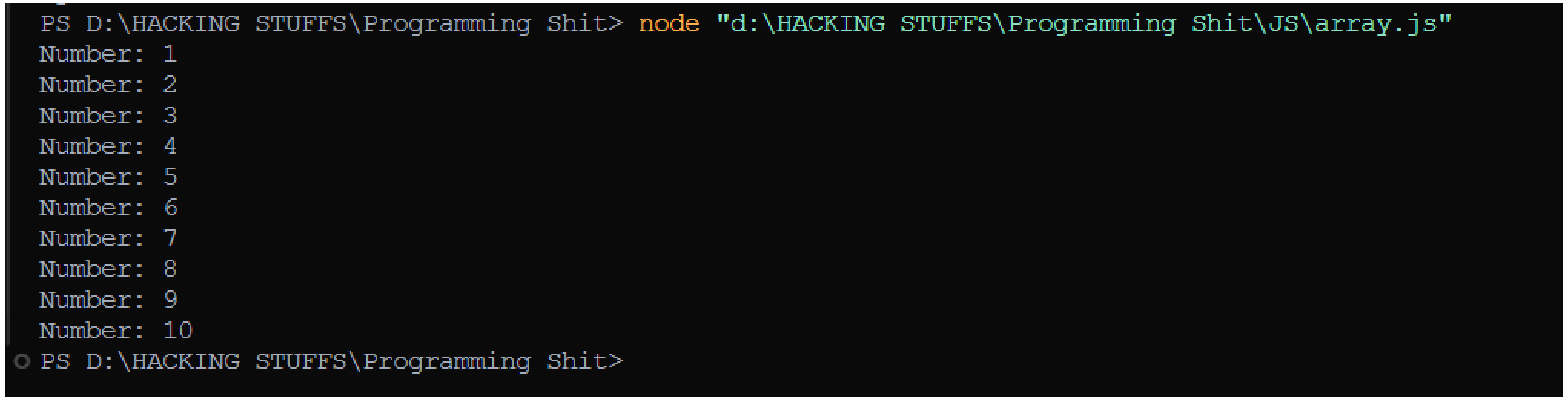
> Loops can be complicated, there are **for loops**, **while loops**, and **do...while loops**.

For Loop

- Code

```
for (a = 1; a ≤ 10; a++) {

    console.log(`Number: ${a}`); // Outputs 1-10 in our console

}
```

- Output



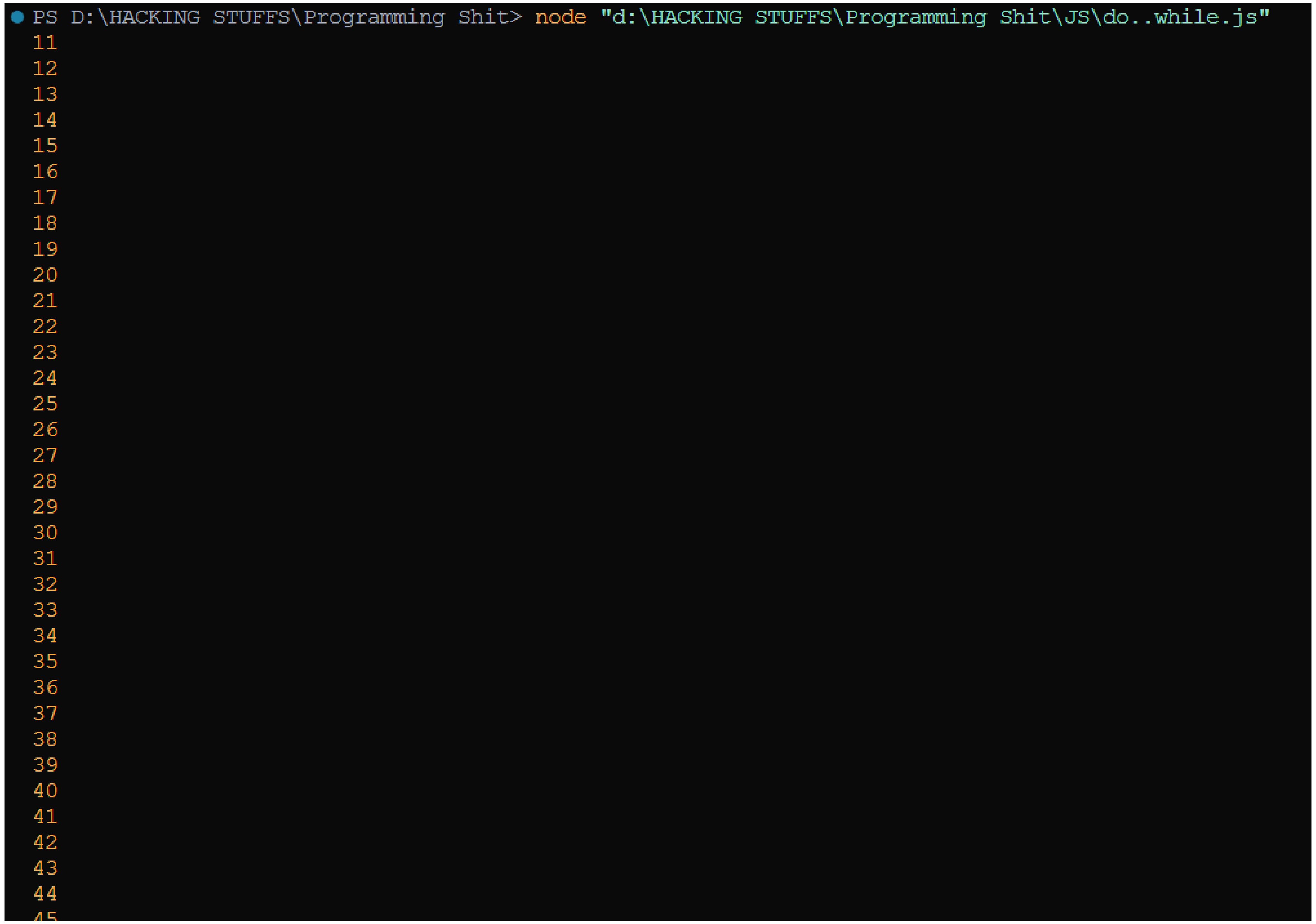
```
PS D:\HACKING STUFFS\Programming Shit> node "d:\HACKING STUFFS\Programming Shit\JS\array.js"
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
PS D:\HACKING STUFFS\Programming Shit>
```

While Loop

- Code

```
let c = 11;

do {

console.log(c++); // Outputs 10-50

} while (c ≤ 50);
```

- Output



```
● PS D:\HACKING STUFFS\Programming Shit> node "d:\HACKING STUFFS\Programming Shit\JS\do..while.js"
  11
  12
  13
  14
  15
  16
  17
  18
  19
  20
  21
  22
  23
  24
  25
  26
  27
  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38
  39
  40
  41
  42
  43
  44
```

#Topic-6 **DOM - Document Object Model***

> The Document Object Model (DOM) has a ton of resources, and combining that with CSS will help your webpage **POP**. Eventually, if you take on web development or front-end programming, then you'll not only gain knowledge around the DOM, but you'll be able to manipulate a virtual DOM using React, cut your code in half using jQuery, and even combine your skills with PHP or Nodejs files for server manipulation.

Here is what we will be covering in the DOM section (keep in mind that these are just a few lines of code, DOM manipulation is a vast subject):

```
`document.getElementByID('Name_of_ID'); // Grabs the element with the ID name from the connected HTML file`

`document.getElementByClassName('Name_of_Class'); // Grabs the element with the class name from the connected HTML file`

`document.getElementByTagName('Name_of_Tag'); // Grabs a specific tag name from the connected HTML file`
```

There are also methods we can use to access different things within our HTML files such as **addEventListener**, **removeEventListener**, and many more. Most of what the DOM does is change, replace, edit, or in some form, manipulate the HTML file or webpage that you're working on. For us to successfully manipulate the DOM, we use events. These events are added to HTML tags to work with our JavaScript file. Some of the more important events that are used a lot, you can find here:

- **onclick**: Activates when a user clicks on the specific element
- **onmouseover**: Activates when a user hovers over a specific element
- **onload**: Activates when the element has loaded
- and many more that are used. You can find a complete list here: https://www.w3schools.com/js/js_htmldom_events.asp

---

THE END!!!!