1. JavaScript Beginner

1. Introduction

- Just In Time Compiled - Conforms to ECMA (**European Computer Manufacturers Association**) Script.
- Client-Side and Server-Side Applications.
- Mobile App Development - iOS and Android.
- Desktop Applications.

---

2. Execution

- JavaScript Code Execution - Web Browser or NodeJS Runtime.
- <script> tag before the end of <body>.
- We can also specify the source of the JavaScript file as an attribute to the script tag for execution.
- To execute in the terminal -

```
> node filename.js
or
> node filename

// Single Line Comment
```

```
/*
  Multi-Line Comment
*/
```

---

3. Variables

- Let - Does not need to be initialized and new values CAN be assigned.
- **Const -** Needs to be initialized and n**e**w values CANNOT be assigned.

	Initialization	Mutable
Let	❌	☑️
Const	☑️	❌

---

4. Data Types

- **Primitive - Written as Unique Values**
  - *String*
    - Single Code, Double Quotes, Back Ticks
  - *Number*
    - Integer, Float
  - *Null*
    - Empty or Unknown
  - *Undefined*
    - When a variable is declared but not defined.
  - *Boolean*
    - true
    - false
  - *Big Int*
    - Value larger than what data Number can hold.
  - Symbol
    - Unique and Unchangeable

- **Non-Primitive - Collection of Values**
  - *Objects*
    - Key - Always String
    - Values - Any Data Type

```
//Objects

const person = {
'firstName': 'Bruce',
'age': 20
}

//Access
Object Literal -> person.firstName
```

- JavaScript - Dynamically Type Language
  - No need to define data types
  - Converted during execution

```
let a = 10
a = 'Nikhil'
a = true

// Variables are typed dynamically - the last value will be assigned!
```

---

5. Type Conversions

- **Implicit Conversion - *Type Coercion***
  - JavaScript - Automatically Convert the Type

```
console.log(2 + "3");
//23

console.log(true + "3");
//true3

console.log("4" - "2");
//2

console.log("4" * "2");
//8
```

```
console.log("4" / "2");
//2

console.log("Bruce" - "Wayne");
//Nan

//False - 0 and True - 1

console.log("5" - false);
//5

console.log("5" - true);
//4

console.log("5" - null);
//5

console.log(5 + undefined);
//NaN
```

- **Explicit Conversion**
  - Manually Converted

```
console.log(Number("5"));
//5

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

console.log(Number(""));
//0

console.log(parseInt("3.14"));
//3

console.log(parseFloat("3.14"));
//3.14

console.log(String(500));
//500

console.log((500).toString());
//500
// Does Not Work on Null and Undefined

console.log(Boolean(10));
//true

//null - undefined - NaN - '' - 0: All give False when converted to a Boolean type.
//Everything Else - Returns True
```

---

6. Equality

```
const var1 = false; or const var1 = 0
const var2 = "";

console.log(var1 == var2);
//true
console.log(var1 === var2);
//false

const var1 = null
const var2 = undefined

//true on ==
```

---

7. Conditional Statements

- if

```
const num = 5;
if (num > 0){
    console.log('Positive Number')
}
```

- if-else

```
const num = 5;
if (num > 0) {
  console.log("Positive Number");
} else {
  console.log("Negative Number");
}
```

- if-else if-else

```
const num = 5;
if (num > 0) {
  console.log("Positive Number");
} else if ((num = 5)) {
  console.log("Number = 5");
} else {
  console.log("Negative Number");
}
```

- switch

```
const color = "red";

switch (color) {
  case "red":
    console.log("Color is Red!");
    break;
  case "blue":
    console.log("Color is Blue!");
    break;
  default:
    console.log("Unknown Color");
}
```

---

8. Looping Code

- for

```
for (let i = 0; i <= 10; i++) {
  console.log(`Value is ${i}`);
}
```

- while

```
let i = 1;
while (i < 10) {
  console.log(`Value is ${i}`);
  i++;
}
```

- do-while (runs at least once)

```
let i = 0;
do {
  console.log(`Value is ${i}`);
  i++;
} while (i < 10);
```

- for .. of
  - Best suited for a collection of data
  - No need to keep track of variable

```
const numArray = [1, 2, 3, 4, 5];
for (const number of numArray) {
  console.log(numArray[number - 1]);
}
```

---

9. Functions

- Block of Code - Designed to Perform a Task
- Reusable
- Divide Complex Tasks into Smaller Chunks

```
function sampleFuntion(parameter){
// ... do something
}

sampleFunction(arguement);

--

function greet(username) {
  console.log(`Hey ${username}`);
}

greet("Bruce");
greet("Diana");
greet("Clark");

--

//Arrow Functions

const greet = (usrnm) => {
 return (`Hey ${usrnm}`);
};

console.log(greet("Nikhil"));

or

const greet = (usrnm) => `Hey ${usrnm}`;
console.log(greet("Nik"));
```

---

10. Scope

- Determines visibility or accessibility of variables

- **Block**

```
if (true) {
  const myName = "Nik";
}

console.log(myName);

// ReferenceError: myName is not defined
```

- **Global**

```
const myName = "Nik";
if (true) {
 console.log(myName);
}
```

- **Function**

```
if (true) {
 const myName = "Nik";
 console.log(myName);
}

function testFn(){
const myName = 'Tan'
console.log(myName);
}

testFn()

//Variables defined within a Function will be scoped to the same.
```