Javascript

Comments:

- Comments are annotations in the code that are completely ignored by JavaScript engines.
- They are essential for making your code more readable and understandable for yourself and other developers.
  Two Types of comments
  1. //Single line Comment
  2. /* Multi Line
     Comment
     */
- Pros: improves code readability and code maintainability
- Cons: Maintenance Overhead

Variables:

- Variables are containers for storing data .
- They can be declared in 4 ways :
  1. Automatically
  2. Using var
  3. Using let
  4. Using const
- When to use var,let or const?
  1. Always declare variables
  2. Always use const if the values should not be changes
  3. Only use let if you can't use const
  4. Only use var if you must support old browsers
- Pros: Redeclaring, Reassigning, Hoisting , Supports old version browser
- Cons: Naming conflicts

Let:

- It is introduced by ES6 to solve redeclaring problem of variables.
- It is used to provide block scope in Javascript
- Pros: Reassignable , block scope
- Cons: No redeclaration , hoisted but not intialised

Const:

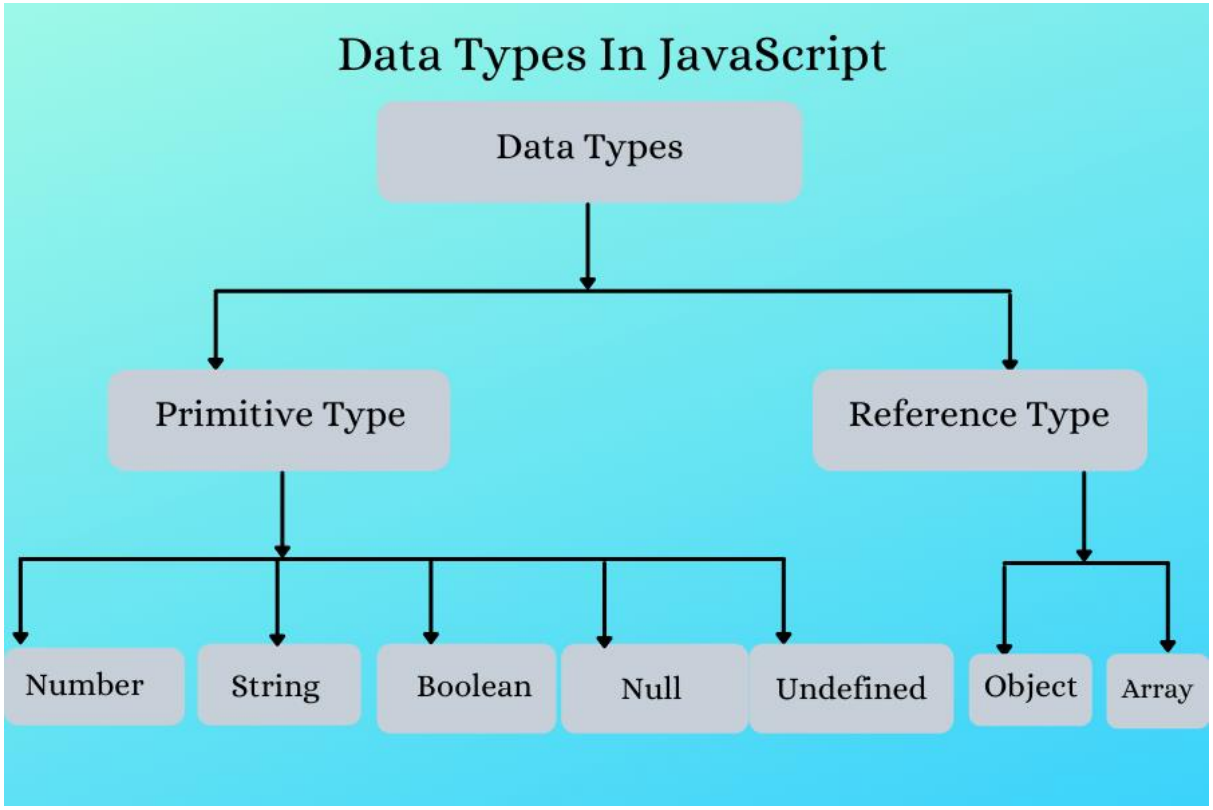
- It is Introduced by ES6
- It is used when the values should not be changed and also provide block scope in Javascript.
- Pros: Block scope
- Cons:No Redeclaration, No Reassignment and no hoisting

Operators:

- Javascript operators are used to perform different types of mathematical and logical computations
- Types of Javascript Operators:
    1. Arithematic opertors(+,-,*,**,/,%,++,--)
    2. Assignment Operators(=, +=,-=, *=, /=, %=, **=)
    3. Comparison Operators(==,===,!=, !==, >,<,>=,<=,?)
    4. String Operators(>,<,+=)
    5. Logical Operators(&&, ||, !)
    6. Bitwise Operators(&,|,~,^,<<,>>,>>>)
    7. Ternary Operators(?)
    8. Type Operators(typeof, instanceof)
- Pros: Enable Fundamental Operations, Foundation for control Flow
- Cons: Precedence and Associativity issues

Datatypes:

- It specifies the kind of value that a variable can hold
- Javascript has 2 types of datatypes



Arrays

- Collection of homogenous and heterogenous data
- An array can hold many values under a single name, and you can access the values by referring to an index number.
- **Creating an array:**
  const cars = [“saab”,”Volvo”,”BMW”];
  const cars = new Array(“Saab”,”Volvo”, “BMW”);
- **Array methods**
  Array.length() – returns the length of an array
  Array.toString() – Converts an array to a string of array values
  Array.at() – returns an index element from an array
  Array.join() – Joins all array elements into a string.(can specify separator)

- **Operations:**
  pop() – This method removes the last element from an array
  push() - This method adds a new element to an array
  shift() - This method work on removing first element instead of last
  unshift()- This method adds a new element to an array(at the beginning)

Objects:

- A data type that stores key-value pairs, allowing you to group related data and functions together, acting as a container for properties and method
- An object literal is a list of property **names:values** inside curly braces **{}**.

- **Object Creation:**

```
// Create an Object
const person = new Object();

// Add Properties
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

- **Object Constructor Function:**

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

- **Object Prototypes:**
  All JavaScript objects inherit properties and methods from a prototype.

Conditional Statements:

- Conditional statements control the flow of your program by executing different blocks of code depending on whether a condition is true or false.
- **If statement**

```
let age = 20;
if (age >= 18)
{
console.log("You are an adult.");
}
```

- **If-else Statement**

```
let isRaining = true;

if (isRaining)
{
  console.log("Take an umbrella.");
}
Else
 {
  console.log("Enjoy the sunshine!");
}
}
```

- **Switch statement**

```
let day = "Saturday";

switch(day) {
  case "Monday":
    console.log("Back to work!");
    break;
  case "Saturday":
  case "Sunday":
    console.log("It's the weekend!");
    break;
  default:
    console.log("Just another day.");
}
```

- **Ternary Operator**
  let age = 17;
  let message = age >= 18 ? "Adult" : "Minor";
  console.log(message); // "Minor"

Looping statements:

- Looping statements in JavaScript are fundamental control flow structures that allow you to execute a block of code repeatedly until a certain condition is met.
- They are essential for automating repetitive tasks and iterating over data structures.
  Syntaxes for different types of Loops:
  1. **For Loop:**
     for (initialization; condition; increment/decrement) {
     // Code to be executed repeatedly
     }

  2. **While Loop:**
     while (condition) {
     // Code to be executed repeatedly
     // Make sure to update the condition within the loop
     // to avoid infinite loops!
     }

  3. **do while Loop:**
     do {
     // Code to be executed repeatedly
     // Make sure to update the condition within the loop
     // to avoid infinite loops!
     } while (condition);

4. **for in Loop:**

```
const person = { name: "Alice", age: 30, city: "New York" };

for (let key in person) {
  console.log(key + ": " + person[key]);
}
// Output (order might vary):
// name: Alice
// age: 30
// city: New York
```

5. **for of loop:**

```
const numbers = [1, 2, 3, 4, 5];

for (let number of numbers) {
  console.log(number);
}
// Output:
// 1
// 2
// 3
// 4
// 5
```

- **Control Flow within Loops:**
  You can use the following statements to control the flow of execution within loops:
  1. **break**: Immediately terminates the current loop and transfers control to the statement following the loop.

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Exit the loop when i is 5
  }
  console.log(i);
}
// Output: 0, 1, 2, 3, 4
```

2. **continue**: Skips the rest of the statements in the current iteration of the loop and moves to the next iteration.

```
for (let i = 0; i < 5; i++) {
  if (i === 2) {
    continue; // Skip the iteration when i is 2
  }
  console.log(i);
}
// Output: 0, 1, 3, 4
```