Chapter 2: Variables & Datatypes

Q. What are variables? What is the difference between var, let, and const ?

Q. What are data types in JS?

Q. What is the difference between primitive and non-primitive data types?

Q. What is the difference between null and undefined in JS?

Q. What is the use of type of operator?

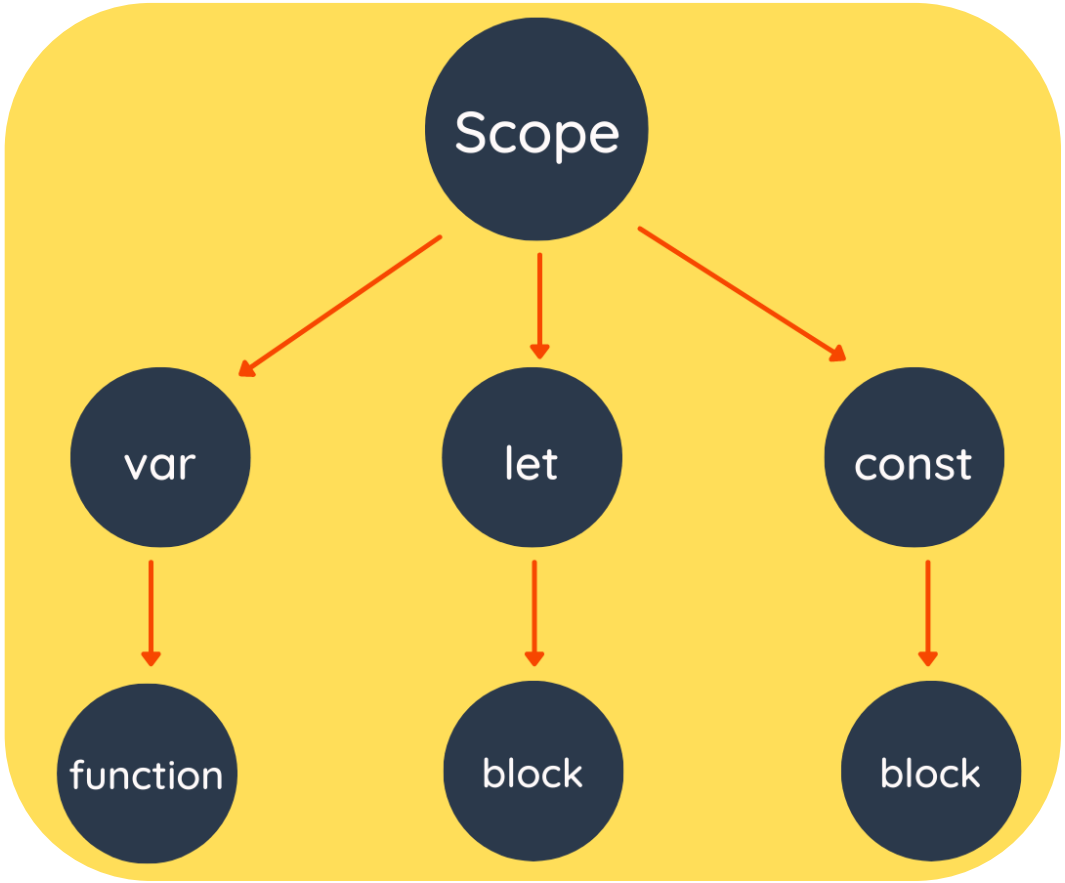
Q. What is type coercion in JS?

Back to main index

Q. What are variables? What is the difference between var, let, and const ? V. IMP.

- Variables are used to **store** data.

```
var count = 10;
```



back to chapter index

Q. What are variables? What is the difference between var, let, and const ? V. IMP.

- var creates a **function-scoped** variable.

```
//using var
function example() {

  if (true) {

    var count = 10;
    console.log(count);
    //output: 10
  }

  console.log(count);
  //Output: 10
}
```

- let creates a **block-scoped** variable

```
//using let
function example() {

  if (true) {

    let count = 10;
    console.log(count);
    //Output: 10
  }

  console.log(count);
  //Output: Uncaught
  //Reference Error:
  //count is not defined
}
```

- const can be assigned only once, and its value **cannot be changed** afterwards.

```
// Using constant
const z = 10;
z = 20;

// This will result
//in an error
console.log(z);
```

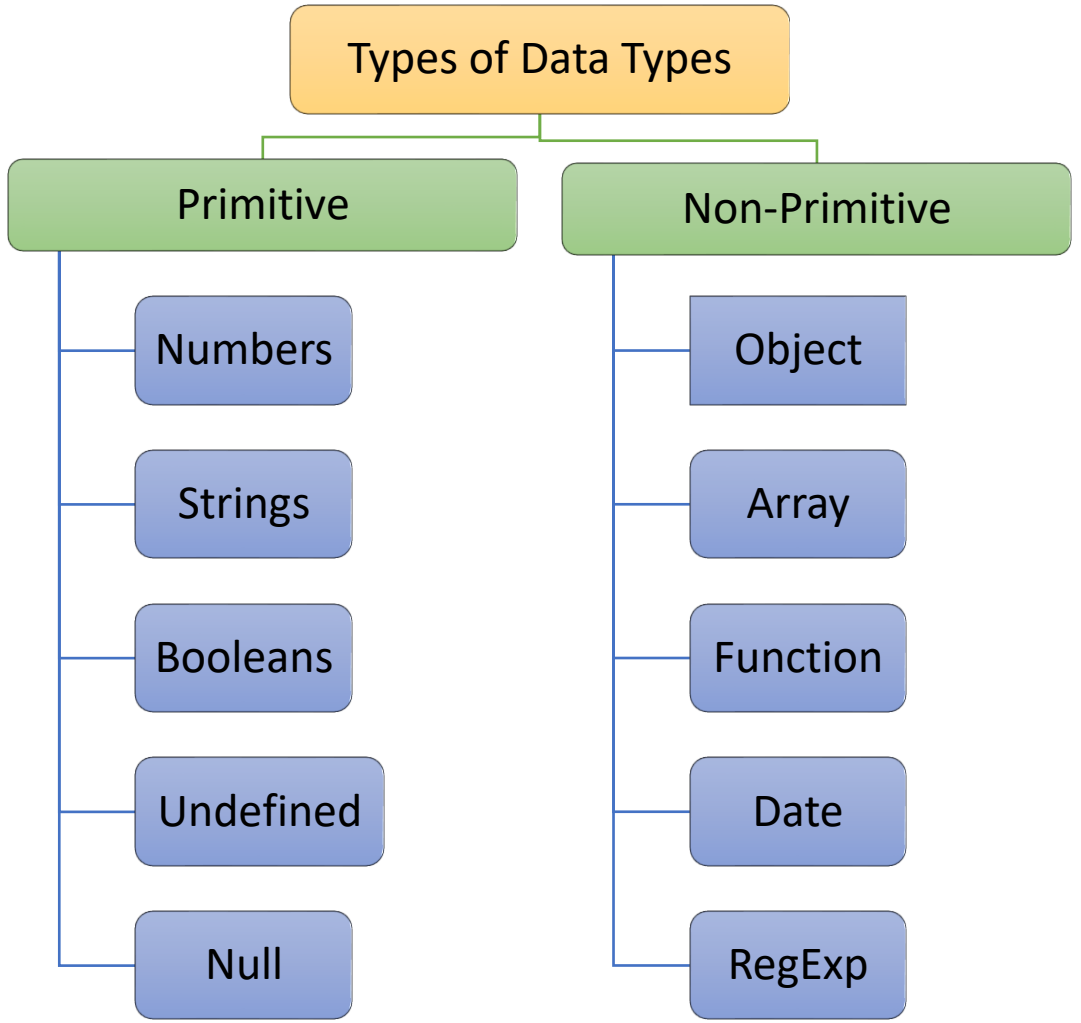
back to chapter index

Q. What are data types in JS?

- A data type determines the **type of variable.**

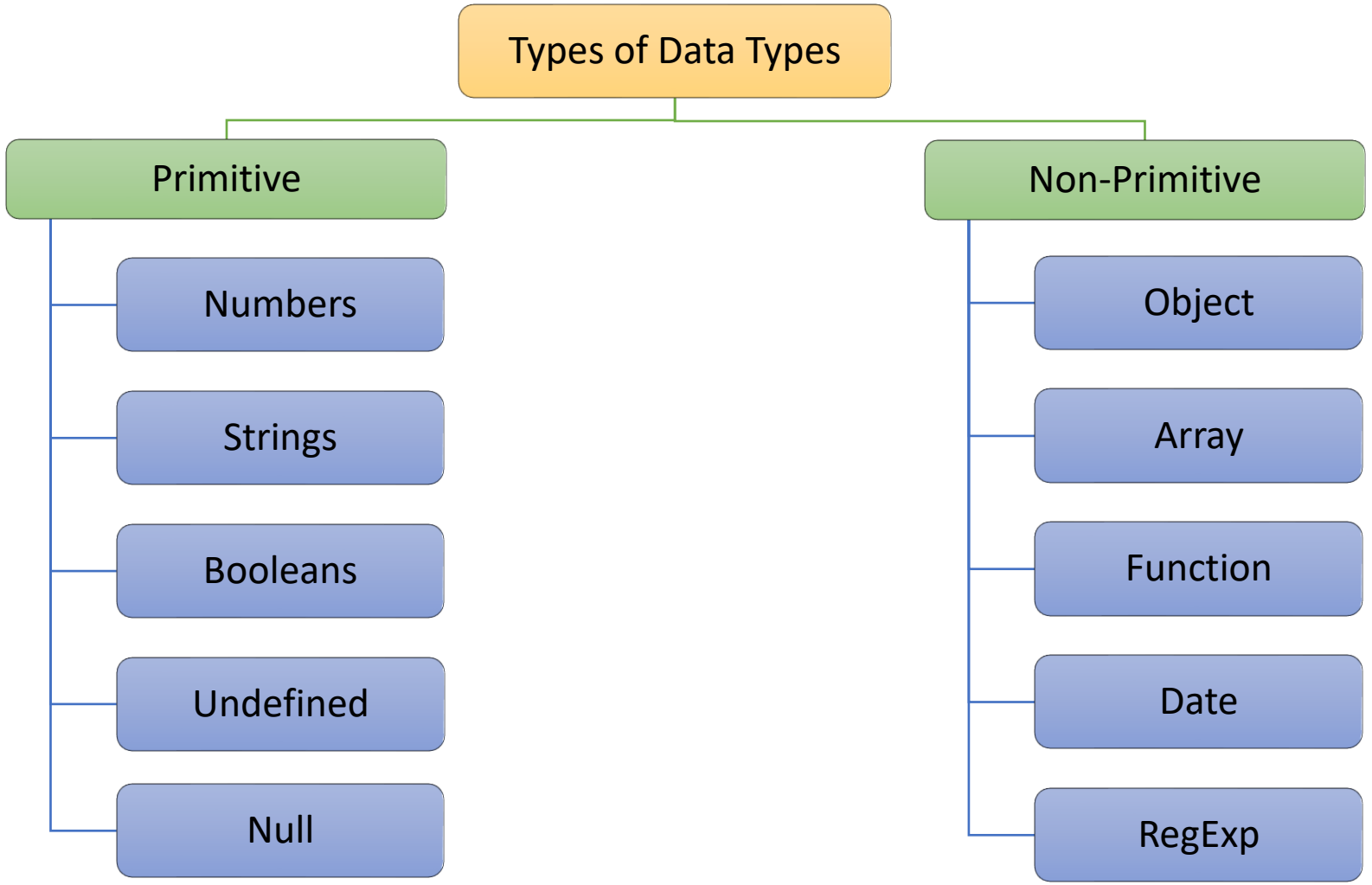
```
//Number
let age = 25;
```

```
//String
let message = 'Hello!';

//Boolean
let isTrue = true;

//Undefined
let x;
console.log(x);
// Output: undefined

//Null
let y = null;
console.log(y);
// Output: null
```



back to chapter index

Q. What is the difference between primitive and non-primitive data types? V. IMP.

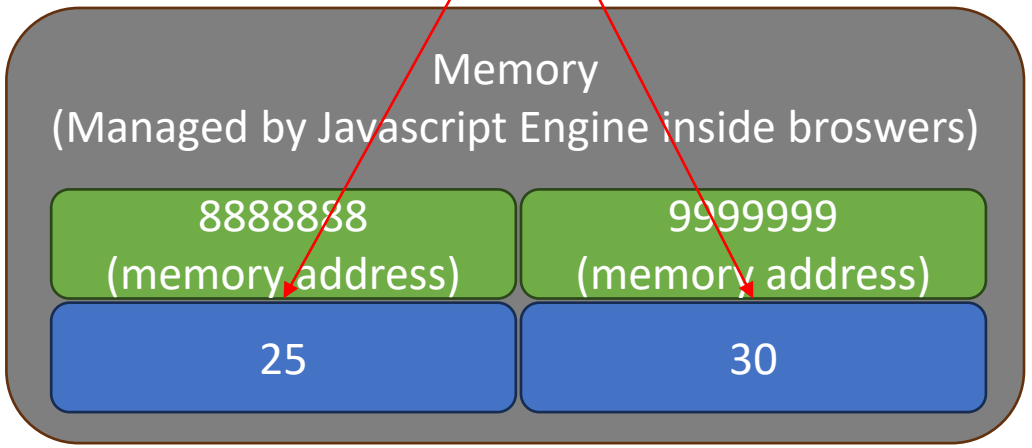


back to chapter index

Q. What is the difference between primitive and non-primitive data types? V. IMP.

- Primitive data types can hold only **single** value.
- Primitive data types are **immutable**, meaning their values, once assigned, cannot be changed.

```
//Number
let age = 25;
    age = 30;
```



- Non primitive data types can hold **multiple** value.
- They are mutable and their values can be changed.

```
//Non primitive data types

//Array
let oddNumbers = [1, 3, 5]

//Object
let person = {
  name: "John",
  age: 30,
  grades: ["A","B","C"],
  greet: function() {
  console.log(this.name);
  }
};
```

back to chapter index

Q. What is the difference between primitive and non-primitive data types? V. IMP.

Primitive Data Types	Non-primitive Data Types
1. Number, string, Boolean, undefined, null are primitive data types.	Object, array, function, date, RegExp are non-primitive data types.
2. Primitive data types can hold only **single** value.	Non-primitive data types can hold **multiple** values and methods.
3. Primitive data types are **immutable** and their values cannot be changed.	Non-primitive data types are **mutable** and their values can be changed.
4. Primitive data types are **simple** data types.	Non-primitive data types are **complex** data types.

back to chapter index

Q. What is the difference between null and undefined in JS?

```
let value1 = 0;

let value2 = '';
```

- (A stand on the wall with also a paper holder) Means there is a **valid variable** with also a value of **data type number**.

```
let value3 = null;
```

- (There is just a stand on the wall) Means there is a **valid variable** with a value of **no data type**.

```
let value4;
```

- (There is nothing on the wall) Means variable is **incomplete variable** and not assigned anything.

back to chapter index

Q. What is the difference between null and undefined in JS?

```
let undefinedVariable; //no value assigned
console.log(undefinedVariable);
// Output: undefined
```

- undefined: When a variable is declared but has **not been assigned a value**, it is automatically initialized with undefined.
- Undefined can be used when you don't have the value right now, but you will get it after some logic or operation.

```
let nullVariable = null; //null assigned
console.log(nullVariable);
// Output: null
```

- null: null variables are intentionally assigned the **null value**.
- Null can be used, when you are sure you do not have any value for the particular variable.

back to chapter index

Q. What is the use of typeof operator?

- typeof operator is used to determine the **type** of each variable.
- Real application use -> typeOf operator can be used to **validate the data** received from external sources(api).

```
let num = 42;
let str = "Hello, world!";
let bool = true;
let obj = { key: "value" };
let arr = [1, 2, 3];
let func = function() {};
```

```
//using typeof
console.log(typeof num); // Output: "number"
console.log(typeof str); // Output: "string"
console.log(typeof bool); // Output: "boolean"
console.log(typeof obj); // Output: "object"
console.log(typeof arr); // Output: "object"
console.log(typeof func); // Output: "function"
console.log(typeof undefinedVariable);
// Output: "undefined"
```

back to chapter index

Q. What is type coercion in JS?

- Type coercion is the automatic conversion of values from one data type to another during certain operations or comparisons.

- Uses of type coercion:

1. Type coercion can be used during **String and Number concatenation.**
2. Type coercion can be used while using **Comparison operators**.

```
let string = "42";
let number = 42;
let boolean = true;
let nullValue = null;
```

```
//Type coercion - automatic conversion
console.log(string + number); // Output: "4242"

console.log(number + boolean); // Output: 43

console.log(number == string); // Output: true
console.log(boolean == 1); // Output: true
console.log(boolean + nullValue); // Output: 1
```

back to chapter index