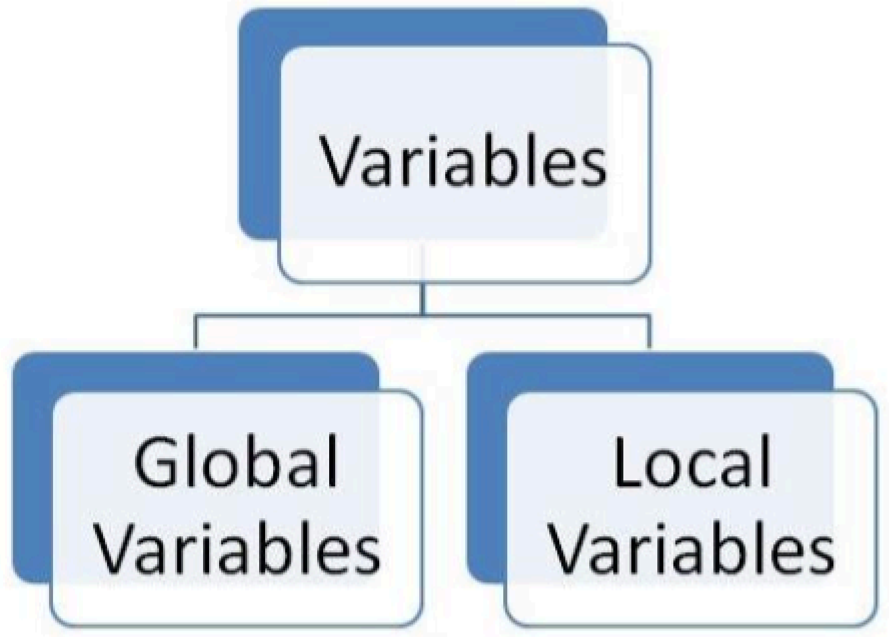
Scope of Variables

Scope of a variable refers to its **visibility** of a variable i.e., in which parts of the program the variable can be seen or used.

JavaScript variables have two scopes:

- **Global Variables**: A global variable has a global scope, which means it is not defined locally (i.e., within any function or block of code) and can be accessed anywhere in your JavaScript code.
- **Local Variables**: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
- **Lexical Scoping**: Visibility of a global variable in the local scope is called lexical scoping.



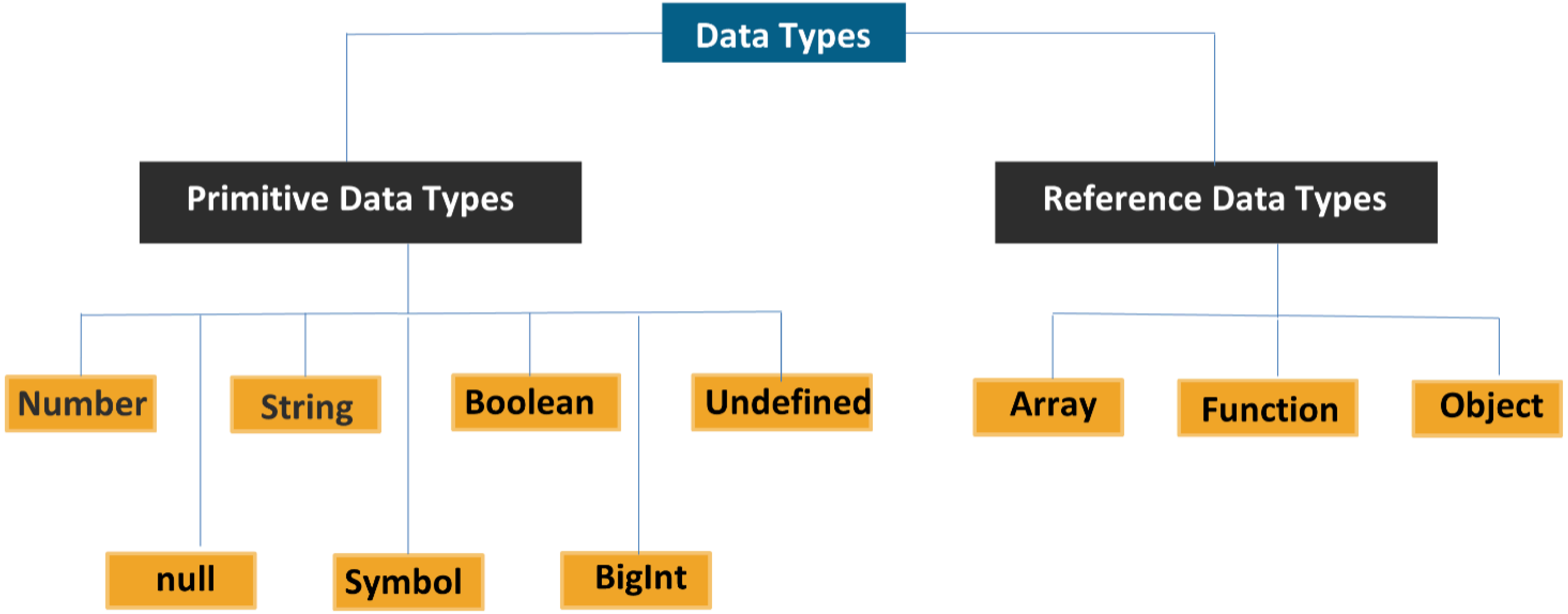
edureka!

Scope of Variables - Example

```
var globalVar = "I am a global variable";

function exampleFunction() {
    var localVar = "I am a local variable";

    if (true) {
        let blockVar = "I am a block-scoped variable";
        console.log(blockVar);  // Accessible here
    }

    console.log(localVar); // Accessible here
    // console.log(blockVar); // Un-commenting this line will cause an error because blockVar is
not accessible here
}

exampleFunction();

console.log(globalVar); // Accessible anywhere in the code
// console.log(localVar); // Un-commenting this line will cause an error because localVar is not
accessible here
```

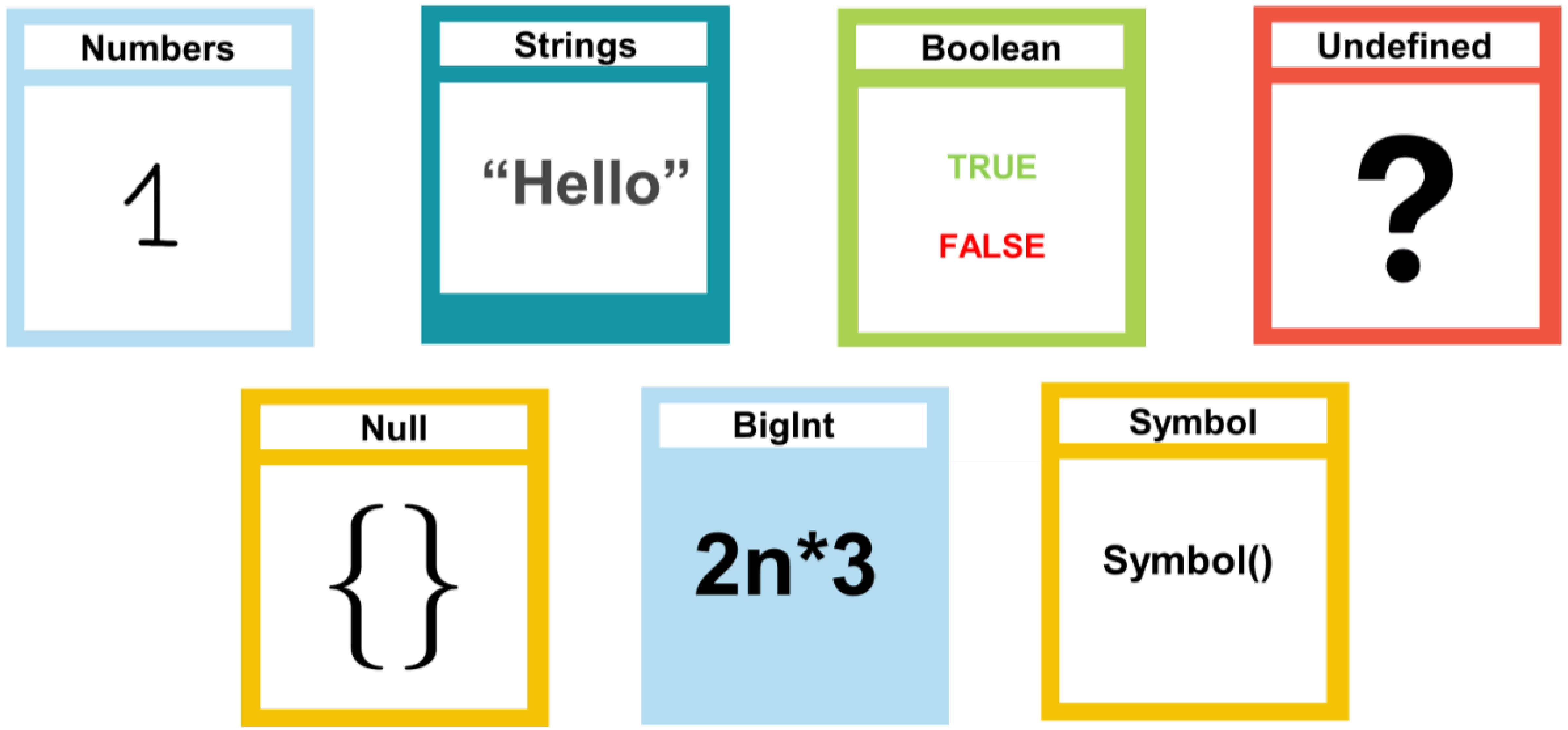
Data Types in JavaScript

Data types in JavaScript categorize data into types like numbers, text (strings), and true/false (booleans), guiding how the computer processes them, such as adding numbers but not text.



Primitive - Data Types in JavaScript

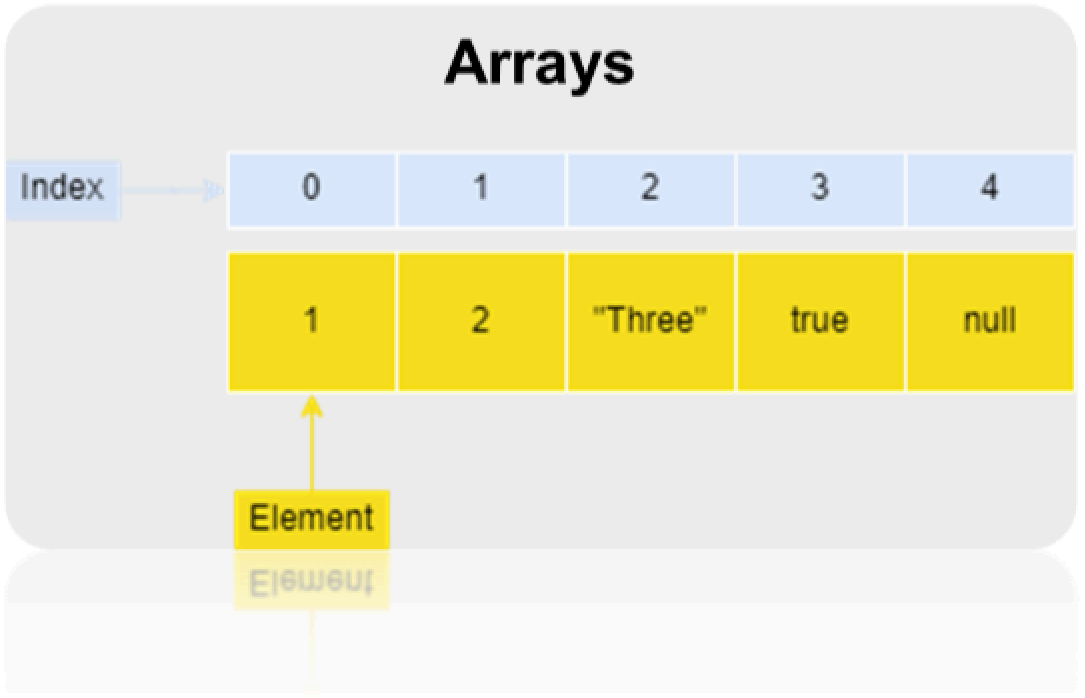
They are basic building blocks that represent simple, singular values like a number, a piece of text, or a true/false condition. They're the most straightforward types of data you can work with, not composed of other values.

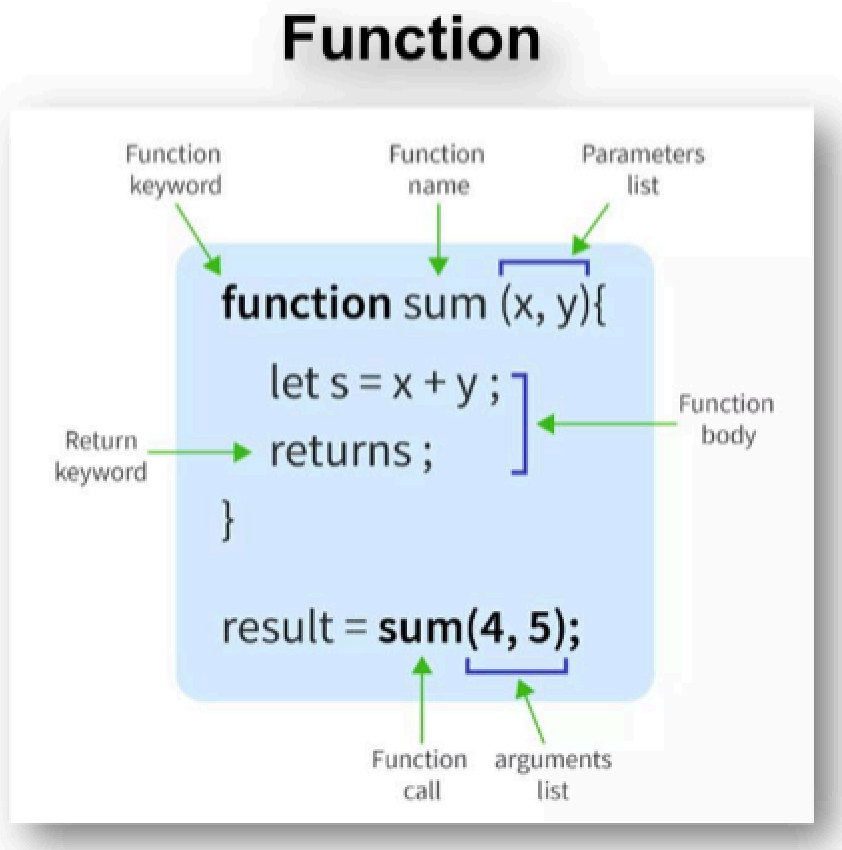


edureka!

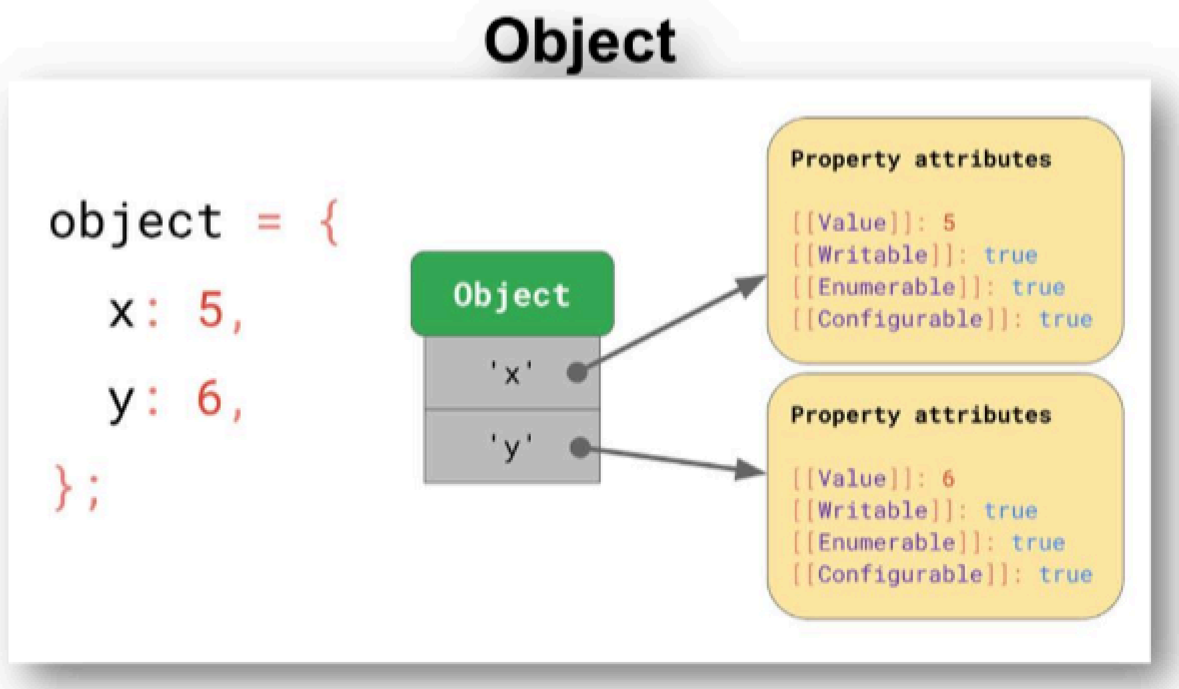
Reference - Data Types in JavaScript

They are variables that store references or links to the actual data, rather than the data itself. They include objects, arrays, and functions, allowing for more complex structures and behaviors in code.





```
function sum (x, y){
    let s = x + y ;
    returns ;
}

result = sum(4, 5);
```



```
object = {
  x: 5,
  y: 6,
};
```

```
Property attributes

[[Value]]: 5
[[Writable]]: true
[[Enumerable]]: true
[[Configurable]]: true
```

```
Property attributes

[[Value]]: 6
[[Writable]]: true
[[Enumerable]]: true
[[Configurable]]: true
```

edureka!

Hoisting in JavaScript

- Hoisting is a mechanism of JavaScript which allows variables and functions to be used before they are even declared.
- The declaration of variables and functions are moved to the top of the scope, before the execution.
- Whether the scope of the declarations is global or local, they will always be moved to the top of their scope.
- Initializations will not be hoisted to the top.

Example 1

```
console.log(hoist)
```

Output 1

```
Uncaught ReferenceError: hoist is not defined
    at <anonymous>:1:13
```

Example 2

```
var hoist = "This variable has been hoisted"
console.log(hoist)
```

Output 2

```
This variable has been hoisted
```

Example 3

```
console.log(hoist)
var hoist = “This variable has
been hoisted”
```

Output 3

```
undefined
```

Variables get moved to the top of their scope when JavaScript compiles at runtime

edureka!