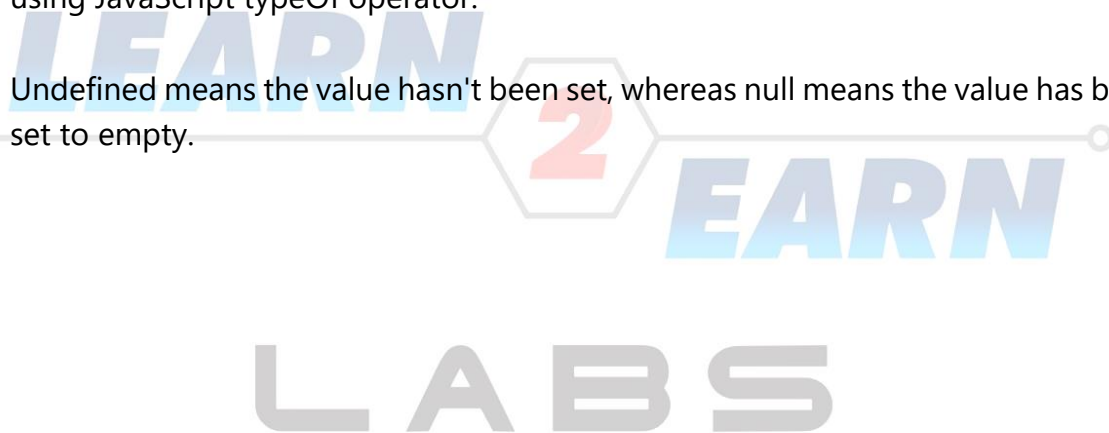


Data Types

- JavaScript data types are an important concept in programming. Lots of programming languages use classification for a better coding experience and bug-free code.
- Data types specifies the type of data stored in a variable.
- Since JavaScript is a dynamic language, we don't need to specify the type of the variable. It is dynamically used by the JavaScript Engine. The var keyword specifies the type of data in the variable like a number, string, etc.
- Data Types tells the interpreter whether the data value is numeric, alphabetic etc., so that it can perform the appropriate operation.
- JavaScript is a dynamic type language; means you don't need to specify type of the variable because it is dynamically detected by JavaScript engine.
- Data in JavaScript is classified into different categories, which can be identified using JavaScript typeof operator.
- Undefined means the value hasn't been set, whereas null means the value has been set to empty.



Types of data types in JavaScript

There are two types of data types in JavaScript :-

1) Primitive Datatypes

- The primitive data types are the basic data types that are available in most of the programming languages.
- All datatypes except Objects define immutable values i.e. these values are incapable of changing. These values are "primitive values".
- In other words we can say that all primitive data types in JavaScript are immutable(not changeable).
- The primitive data types are used to represent single values.

JavaScript has six types of primitive data types, based on the type of data we are storing in the variable :-

- a) Number
- b) Boolean
- c) String
- d) Null
- e) Undefined
- f) Symbol (new in ECMAScript 6)

2) Non-Primitive / Composite Datatypes / Complex Datatypes / Reference Datatypes

- Those data types that are derived from primary data types are known as non-primitive data types.
- These data types are used to store group of homogenous (same type) or heterogenous (different type) data items / values.
- The non-primitive datatypes in JavaScript include objects, arrays, and functions.

There are three non-primitive data types in JavaScript :-

- a) Objects
- b) Arrays
- c) Functions



Primitive Datatypes

a) Numbers

ECMAScript has two built-in numeric types: Number and BigInt. The Number type represents both integer and floating-point numbers.

a.1) Number

- JavaScript Numbers is a reference for numeric values.
- Number type represents integer, float, hexadecimal, octal or exponential value.
- All numbers in JavaScript are 64-bit floating-point numbers.
- Infinity represents a number too big for JavaScript to handle. JavaScript has special keyword 'Infinity' and '-Infinity' to represent positive and negative infinity respectively.

- Example -- All examples referred to number datatype

```
var num1 = 123;  
var num2 = 12.3456;  
var num3 = -1000;  
var num4 = 0xff;  
var num5 = 4.25e+6;  
var num6 = 4.25e-6;  
var num7 = Infinity;  
var num8 = -Infinity;  
var num9 = NaN;
```

a.2) BigInt

- This is a new numeric datatype in JavaScript that can represent arbitrary precision known as BigInt.
- It can safely store and operate on large integers that are beyond the safe limit for Number.
- We create 'bigint' by appending 'n' to the end of the integer or by calling the constructor.

Example -- All examples referred to BigInt

```
var a = 23n;    // bigint
var b = 2n ** 53n; // bigint
var abc = 23n * 20 // error, cannot operate bigint with number & other types
```

b) Boolean

A boolean is a data type that can have only one of two possible values: true or false. Booleans in JavaScript are used for testing conditions.

Example

```
var a = true // a boolean value
var b = false // a boolean value
```

c) Strings

- Strings are JavaScript data types that hold information in text & characters.
- They are encapsulated with quotes (unlike numbers, which do not need them). Both double and single quotes are accepted with strings.
- The String type represents textual data in JavaScript and is a set of "elements" of 16-bit unsigned integer values.
- Each element in the String occupies a position in the String, starting with index 0.
- JavaScript strings are immutable, i.e., once it is created, it is not possible to modify it.

Example

```
var abc = "Hello Strings" // double quotted string ;  
var bac = 'Hello Strings' // single quotted string ;
```

d) Null

- Null is empty value .
- The null keyword cannot be used as name of the variable.
- In JavaScript null is not same as 0. It is absence of any value.
- If we try to reference a value for variable which is not defined and has no value then it returns a null value.
- In JavaScript the null is consider as Object.
- We can clear an object by setting it to JavaScript 'null'.

Example

```
var abc = null; // Value is null, but type is still an object
```

e) Undefined

- If a variable does not have a value, JavaScript automatically sets it as undefined.
- Undefined is also a primitive value in JavaScript.
- A variable or an object has an undefined value when no value is assigned before using it.
- We can also clear out an object by setting it to undefined and due to this an object loses both its value and type.

Example

```
var human = undefined; // Value is undefined, type is undefined
```

f) Symbol

- The Symbol type is a unique and immutable primitive value and may be used as the key of an Object property.
- We use them to create unique identifiers for objects.

Example

```
var symbol1 = Symbol(); // a symbol  
var symbol2 = Symbol(5); // a symbol
```



2) Non-Primitive / Reference Datatypes

a) Objects

- JavaScript Object is a collections of properties and methods.
- In JavaScript, anything that is not a primitive is an Object.
- The object type must have a key and a value: they are written in key:value pairs.
- A key value is a String or a Symbol value.
- In JavaScript, objects are used when working with 'complex' data structures.
- Property values can be values of any type, including other objects, which enables building complex data structures.
- In JavaScript, anything that is not a primitive type (undefined, null,boolean, number, or string) is an object.

Example

```
var lang = { // Object with key:value pair
  markup : "HTML",
  styling : "CSS",
  scripting : "JavaScript"
}
```

LABS


b) Arrays

- JavaScript arrays are defined with square brackets.
- Arrays are one of those JavaScript data types that are commonly used with variables.
- Array items are normally separated with commas.
- Values in an array are accessed by the array name and location/index of the value.
- The first element of the array is always indexed as 0.

Example

```
var languages = ["HTML", "CSS", "JAVASCRIPT"];
```

Difference Between Arrays & Objects

- Arrays are a special type of object in JavaScript.
- The main difference between arrays and objects is that array contents have a natural order and the keys are numeric and sequential. On the other hand, object keys can be numeric as well as strings.
- Also "arrays" are declared within square brackets '[]', while that of objects are declared within curly brackets '{}'.


c) Functions

- Functions are 'function objects'.
- A function is a subprogram designed to perform a particular task.
- Functions are executed when they are called. This is known as invoking a function.
- Functions belong to the object type. But typeof treats them differently, returning "function".

Example

```
function hello(){  
    console.log("Hello Function")  
}  
hello()
```



JavaScript engine with primitive & reference types

When we assign a value to a variable, the JavaScript engine firstly determine whether the value is a primitive or reference value.

1) In Case of Primitive Types

If the value is primitive, when you access the variable, you are manipulating the actual value stored in the variable. In other words, the variable that stores a primitive value is accessed by value.

When we assign a variable that stores a primitive value to another, the value stored in the variable is created and copied into the new variable.

Example

```
var a = 20; // Primitive Value
var b = a; // Here, JavaScript engine copies the value stored in 'a' into the location of 'b'.
b = 30; // Value of 'b' changes as there is no relationship between 'a' and 'b'
```

```
console.log(a); // 20
console.log(b); // 30
```

Here, a and b have no relationship, therefore when we change the value stored in the b variable, the value of the a variable remains unchanged.

Example

```
var number = 20;
function hello(abc){ // Here, abc == number; hence not changed.
  abc++;
}

hello(number)
console.log(number)
```

2) In Case of Reference Types

- If we manipulate an object then we need to work on the reference of that object, rather than the actual object. In short, a variable that stores an object is accessed by reference.
- When we assign a reference value from one variable to another, the value stored in the variable is also copied into the location of the new variable.
- The values stored in both variables are the address of the actual object, stored in the heap memory. As a result, both variables are pointing to the same object.
- When we assigned a reference value to a variable then an address/location will be created in the heap memory location. So, whenever we want to access the value of the variable, we get that value from the heap memory by the 'address/location' not by the 'variable value / variable name'.

Example -- Arrays

```
var arr = [10,20,30,40] // Reference Type
var arr2 = arr; // Both 'arr' and 'arr2' are now pointing to the same object on the heap.
arr2.push(50);

console.log(arr2); // [10,20,30,40,50]
console.log(arr); // [10,20,30,40,50]
```

Example -- Objects

```
var obj = { // Reference Type
  name : "Mohit",
  age :34
}

var obj2 = obj; // Both 'obj' and 'obj2' are now pointing to the same object on the heap.
obj2.name = "Mohit Singh"
console.log(obj); // {name : "Mohit Singh",age : 34}
console.log(obj2); // {name : " Mohit Singh ",age : 34}
```

Here, both obj and obj2 are pointing to the same object, therefore, the change is also reflected in both the objects.

Example

```
var obj = {name : "Mohit"};
```

```
function hello(abc){ // Here, abc = obj; hence both changes  
  abc.name = "Mohit Singh"  
}
```

```
hello(obj)  
console.log(obj)
```

