JavaScript data types

JavaScript is a dynamic language with dynamic types. Variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned (and re-assigned => let and var) values of all types.

JavaScript is a weakly typed language, which means it allows implicit type conversion when an operation involves mismatched types, instead of throwing type errors.

There are two types of data types in JavaScript.

- ➢ Primitive data type (primitive data type are imutable)
- ➢ Non-primitive data type (mutable)

All primitive types, except null, can be tested by the typeof operator. typeof null returns "object". All primitive types, except null and undefined, have their corresponding object wrapper types, which provide useful methods for working with the primitive values. When a property is accessed on a primitive value, JavaScript automatically wraps the value into the corresponding wrapper object and accesses the property on the object instead.

Primitive type

Type	typeof return value	Object Wrapper	Description
Null	"object"	N/A	Is inhabited by exactly one value: null. It indicates the absence of an object.
Undefined	"undefined"	N/A	Indicates the absence of a value
Boolean	"boolean"	Boolean	Represents a logical entity and is inhabited by two values: true and false. Boolean values are usually used for conditional operations.
Number	"number"	Number	All JavaScript numbers are stored as decimal numbers (floating point). JavaScript treats all numbers as floats behind the scenes, even integers like 122
BigInt	"bigint"	BigInt	Represent integers with arbitrary magnitude. With BigInts, you can safely store and operate on large integers.
String	"string"	String	Represents sequence of characters e.g. "hello"
Symbol	"symbol"	Symbol	A Symbol is a unique and immutable primitive value and may be used as the key of an Object property.

Non-primitive data type

JavaScript non-primitive types are **objects**. An object holds a reference/address of a single key-value pair or many key-value pairs. Whenever we refer to an object, we refer to an address in memory which contains the key-value pair. If we assign an object ‘object1’ to another object ‘object2’, we are actually assigning the address of ‘object1’ to ‘object2’ instead of the key-value pair which the ‘object1’ contains in memory.

The object data type can contain:

- An object
- An array
- A date (new Date() constructor)