**Data Types**

Types of data assigned in a variable name using assignment operator called Data Type.  
Ex: var myName = “Chandan”  
Here, we are assigning a Data ‘Chandan’ in a variable name ‘myName’ via assignment operator ‘*=*’ using ‘var’ keyword.

So, we use ‘typeof()’ method/function to get the exact Data Type of an assigned data in variable name.  
Ex: typeof(myName), Output:- string  
So, ‘Chandan’ is an string data type.

There are 6 primitives Data Types in Javascript:-

1. Undefined: typeof instance === “undefined”
2. Boolean: typeof instance === “boolean”
3. String: typeof instance === “string”
4. BigInt: typeof instance === “bigint”
5. Symbol: typeof instance === “symbol”
6. Number: typeof instance === “number”

For more info: https://developer.mozilla.org/en-us/docs/web/javascript/data\_structures

Note:-

* To assign any type of data type in a variable name we use ‘var’, ‘let’ or ‘const’ keyword in javascript.
* In Js, ‘Int’ data type called ‘number’ data type.
* In Js, 10 + “20” = 1020 but 10 – “20” = -10. It’s because this is First bug in JavaScript.
* A Variable name with ‘no value’ are type of ‘undefined’ data type.  
  **Ex:** var myName; , Output:- undefined  
  **Ex:** console(typeof(myname)); , Output:- undefined
* A variable name with ‘null’ value are type of ‘object’ but there is no data type of name object in JavaScript. So this an another bug in JS.  
  **Ex:** var myName = null; , Output:- undefined  
  **Ex:** console(typeof(myname)); , Output:- object
* In Js, “Chandan” + “Kumar” = ChandanKumar (This is called concatenation and typeof(“Chandan” + “Kumar”) = its data type is ‘**string**’ ) but “Chandan” – “Kumar” = NaN (Not A Number and its type0f(“Chandan” – “Kumar”) = data type is ‘**number**’ which is a another bug in JS).

What is NaN?

NAN is a property of the global object. In other words, it is a variable in global scope. The initial value of NAN is Not-A-Number.  
  
Example: var number = 89345;  
 var myName = “Chandan”;  
isNaN(number); Output:- false  
isNaN (myName); Output:- true

|  |  |
| --- | --- |
| Input | Output |
| NaN===NaN; | False |
| Number.NaN===NaN; | false |
| isNaN(NaN); | true |
| Number.isNaN(NaN); | true |

|  |  |
| --- | --- |
| Data | Data Types |
| 10 | number |
| “Chandan” | String |
| true | Boolean (value = 1) |
| false | Boolean (value = 0) |

**Reference (Array, Object) & Primitive Types (String, Number)**

Primitive Types:-

In primitive type value of a variables can be copied and changed but not possible in Reference type.

**Example:**

let num1 =100;  
let num2 = num1; ***//here, value of num1 getting copied to num2.***

num2 = 50; ***//Got Changed to this variable only.***

console.log(num1);  
console.log(num2);

Output:  
100  
50

Reference Types:-

In reference type, variables(array, object) get assigned or point to another variable.

**Example:**

let person1 = {

name = “Chandan”

}

let person2 = person1; ***//Here, person1 not getting copied to person2, it’s just getting assigned or pointing to person2.***

person2.name = “Amar”; ***//Got Changed to both variables***

console.log(person1);  
console.log(person2);

Output:  
{name: “Amar” }  
{name: “Amar” }

**Note:**

If we want to copy one object inside another one, we’ll use spread operator.

**Example:**

let person1 = {

name = “Chandan”

}

let person2 = {

...person1

};

person2.name = “Amar”; ***//Got Changed to this variable only***

console.log(person1);  
console.log(person2);

Output:  
{name: “Chandan” }  
{name: “Amar” }