DATATYPES IN JAVASCRIPT

JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date

String: Strings represent sequences of characters, enclosed within single (' '), double (" "), or backtick ( ) quotes. For example:

let greeting = "Hello, world!";

Number: Numbers in JavaScript can be integers or floating-point numbers. They can be written with or without decimals. For example;

let num = 42;

let pi = 3.14;

BigInt: Introduced in ECMAScript 2020, BigInt is used for representing whole numbers larger than 2^53 - 1 or smaller than -(2^53 - 1). It is followed by the letter 'n'. For example:

const bigNum = 1234567890123456789012345678901234567890n;

Boolean: Boolean represents a logical entity and can have two values: true or false. For example:

let isTrue = true;

let isFalse = false;

Undefined: Represents the absence of a value. It's typically the default value assigned to variables that have not been assigned a value yet. For example:

let nullVar = null;

Symbol: Introduced in ECMAScript 2015, symbols represent unique identifiers. They are often used as property keys for objects. For example:

```
const sym = Symbol('description');
```

Object: Objects are collections of key-value pairs. They can contain various data types and structures, and they provide methods for manipulation. For example:

```
let person = {
  name: "John",
  age: 30,
  hobbies: ["reading", "traveling"],
  address: {
    city: "New York",
    country: "USA"
  }
};
```

The Object data type in JavaScript can contain different types of values, including:

1. **An Object: This refers to a collection of key-value pairs, as shown in the example above.**
2. **An Array: Arrays are a special type of object used for storing multiple values in a single variable. For example:**

```
let colors = ["red", "green", "blue"];
```

A Date: Date objects in JavaScript are used to work with dates and times. They can represent a particular moment in time, and they provide methods for date and time manipulation. For example:

```
let today = new Date();
```

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as

named containers. You can place data into these containers and then refer to the data simply by

naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with

the var keyword as follows.

<script type="text/javascript">

<!--

var money;

var name;

//-->

</script>

You can also declare multiple variables with the same var keyword as follows −

<script type="text/javascript">

<!--

var money, name;

//-->

</script>

Storing a value in a variable is called variable initialization. You can do variable initialization at

the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value 2000.50 to it later.

For another variable, you can assign a value at the time of initialization as follows.

<script type="text/javascript">

<!--

var name = "Ali";

var money;

money = 2000.50;

//-->

</script>

Note − Use the var keyword only for declaration or initialization, once for the life of any variable

name in a document. You should not re-declare same variable twice.

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any

data type. Unlike many other languages, you don't have to tell JavaScript during variable

declaration what type of value the variable will hold. The value type of a variable can change

during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables

have only two scopes.

Global Variables − A global variable has global scope which means it can be defined

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.

Within the body of a function, a local variable takes precedence over a global variable with the

same name. If you declare a local variable or function parameter with the same name as a global

variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
var myVar = "local"; // Declare a local variable
document.write(myVar);
}
//-->
</script>
```

This produces the following result −

local

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

You should not use any of the JavaScript reserved keywords as a variable name. These

keywords are mentioned in the next section. For example, break or boolean variable names

are not valid.

JavaScript variable names should not start with a numeral 0 − 9. They must begin with a letter

or an underscore character. For example, 123test is an invalid variable name but _123test

is a valid one.

JavaScript variable names are case-sensitive. For example, Name and name are two

different variables.

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used

as JavaScript variables, functions, methods, loop labels, or any object names.

abstract

boolean

break

byte

else

enum

export

extends

instanceof

int

interface

long

switch

synchronized

this

throw

case

catch

char

class

const

continue

debugger

default

delete

do

double

false

final

finally

float

for

function

goto

if

implements

import

in

native

new

null

package

private

protected

public

return

short

static

super

throws

transient

true