Data Types

Primitive	Non-Primitive
Number - represents integer and floating values	Object - represents key-value pair
String - represents textual data	
Boolean - logical entity with values as true or false.	
Undefined - represents a variable whose value is not yet assigned.	
Null - represents the intentional absence of value.	
Symbol - represents a unique value.	

Creating an Object

1. **Using curly brackets -**
    - Create empty object as - `var obj = { } ;`
    - Object with some initial properties as - `var obj = { key1: value1, ... , keyN:valueN }`

2. **Using new operator -**
    - Create empty object as - `var obj = new Object( );`
    - Object with properties as - `var obj = new Object( { key1: value1, ..., keyN: valueN } )`

Example :

```
var obj = {
    key1: "value1",
    key2: 12345,
    "key3": true,
    key4: function( ) {
        //code
    }
}
```

Example :

```
var ball = {
        sport : "Cricket",
        colour : "Yellow" ,
        radius : 3 ,
        print : function( ){
                console.log("Coding Ninjas");
            }
    }

console.log( ball.sport );  // Cricket
console.log( ball["radius"] );  // 3
console.log( ball.size );  // undefined
ball.print( ); // Coding Ninjas
```

Deleting Property

You can remove property of object using **delete** operator followed by the property name. You can either use **dot operator** or **square bracket** notation.

Syntax :

```
delete obj.objectName ;
        OR
delete obj["objectName"] ;
```

Example:

```
delete ball.radius ;
```

How are Objects Stored

There are two things that are very important in objects -

- Objects are **stored in a heap.**
- Objects are **reference types.**

But, if you assign **one object to another,** then the value of **'item1' gets assigned to 'item2',** and therefore, they both will point to the same location -

Example :

```
var item1 = { name: "Coding Ninjas" } ;
var item2 =  { name: "Coding Ninjas" } ;
item1 = item2;

console.log(item1 == item2) ;   // Returns true
console.log(item1 === item2) ; // Returns true
```

Example :

```
for( key in ball) {
        console.log( key , ":", ball[key] );
}
```

Output :

```
sport : Cricket
colour : Yellow
radius : 3
print : function(){
        console.log("Coding Ninjas");
 }
```

```
var arr = [10, 20, 30] ;
arr["four"] = 40 ;
console.log(arr) ;
```

Output : Array(3) [10, 20, 30]

- But, it also **contains the property "four: 40"**, but it **does not show** in the array. But if you use the **for-in** loop to traverse it, you can traverse all the properties.

```
for(var i in arr) {
    console.log( i, ":", arr[i]);
}
```

Output :
0 : 10
1 : 20
2 : 30
four: 40

this keyword

Define a function to get the full name of a person in the object-person

```
var person = {
        firstName: "Tony",
        lastName : "Stark",
        age : 40 ,
        getname: function( ) {
                return this.firstName + " " + this.lastName;
               }
        };
        console.log(person.getname( )) ; //Tony Stark
```

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Constructor

```
function Student(first, last, age) {
        this.firstName = first;
        this.lastName = last;

        this.age = age;
        }

var stu1 = new Student("John", "Doe", 50);
var stu2 = new Student("Sally", "Rally", 48);
```

It is important always to use the **new keyword** when invoking the constructor.
If new is not used, the constructor may clobber the 'this', which was accidentally passed in most cases as the global object (window in the browser or global in Node.). Without the **new** function will not work as a constructor.

Class

Example :

```
class Person {
        constructor(first, last) {
                this.firstName = first;
                this.lastName = last;
        }
}
```

```
let p1 = new Person("Rakesh", "Kumar") ;
```

Class Expression

A class expression is another way to define a class, which is similar to a function expression. They can be **named and unnamed both**, like -

```
let Person = class { };
        OR
let Person = class Person2 { };
```

Creating a regular expression with modifiers

Example :

```
let reg = /hello/g ;
let reg = /hello/i ;
let reg = /hello/m ;
```

- /hello/i is a regular expression. “hello” is a pattern, and “i” is a modifier that modifies the search to be case-insensitive. If we write /hello/g, here “g” performs a global match that will find all matches rather than stopping after the first match.

Regular Expressions Methods

Regular expressions are used with the RegExp methods like test( ) and exec( ) and with the string methods replace( ) and split( ) .

Method	Description
exec( )	Tests for a match in a string. Returns the first match
test( )	Tests for a match in a string. Returns true or false
search( )	Returns index of first match else -1
replace( )	Replaces the matched substring with a replacement substring.
split( )	Break a string into an array of substrings

```
let str = "Hi, Am I talking to Rishabh ? Yes This is Rishabh "
let reg = /Rishabh/

console.log(reg.exec(str))
console.log(reg.test(str))
console.log(str.search(reg))
console.log(str.split(reg,"Rishabh"))
console.log(str.replace(reg,"Ardy"))
```

```
[
  'Rishabh',
  index: 20,
  input: 'Hi, Am I talking to Rishabh ? Yes This is Rishabh ',
  groups: undefined
]
true
20
[]
Hi, Am I talking to Ardy ? Yes This is Rishabh
```

Exception Handling Statements

- try...catch statements
- throw statement
- try...catch...finally statements

Try and Catch Statements

Example :

```
var a = [1,2,3];
try{
   console.log(b[4]);
}catch(e){
   console.log("Error");
}
```

Output : Error

- ❖ Because b array was not defined anywhere

Throw Statement

Example :

```
throw "Custom Error" ;
```

Output : `Uncaught Custom Error`

Example :

```
var a = [1,2,3];
try{
   console.log(b[4]);
}catch(e){
   throw "Error in code";
}
```

Output : `Uncaught Error in code`

Try, catch and finally

Example : var password = 1234 ;

```
try{
  if(password == 1234)
    console.log("Correct password");
 }catch(err){
    throw "Incorrect password";
 }finally{
     console.log("Welcome to Coding Ninjas");
 }
```

Output : Correct password
Welcome to Coding Ninjas

Error Object

JavaScript has a built-in error object that provides error information when an error occurs.
The error object provides two useful properties: name and message.

Error Name	Description
EvalError	An error has occurred in the eval( ) function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

Strict mode

Using strict mode globally

Example :

```
x = 10 ;
console.log(x) ; // Outputs 10
```

```
"use strict"
x = 10 ;
console.log(x) ; // ReferenceError: x is not defined
```

- Creating variables without a literal is **not allowed in strict mode**

Using strict mode within a function

Example :

```
function abc(a, a) {
    console.log(a + a);
}
abc(10, 20) ;  // Output 40
```

```
function abc(a, a) {
    'use strict' ;
    console.log(a + a);
}
abc(10, 20);  // Duplicate parameter name not allowed in this context
```

Not Allowed in Strict Mode

Names that can not be a variable name under strict mode-

- eval
- arguments

Example :
```
"use strict" ;
var eval = 10 ;
```
Output : `SyntaxError: Unexpected eval or arguments in strict mode`

```
"use strict" ;
var arguments = 10 ;
```
Output : `SyntaxError: Unexpected eval or arguments in strict mode`

JavaScript Debugging Methods

- Using debugger keyword
- Using console.log( )

JSON

The JavaScript JSON is an acronym for **JavaScript Object Notation.** It provides a format for storing and transporting data. It is a lightweight, human-readable collection of data that can be accessed logically.

- It generates and stores the data from user input.
- It can transport the data from the server to the client, client to server, and server to server.
- It can also build and verifying the data.

❖ JSON is often used when data is sent from a server to a web page.

```
{
 "student": {
       "name":  "Sam",
       "fees": 56000,
       "institution": “Coding Ninjas”
   }
}
```

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

Methods	Description
JSON.parse( )	This method takes a JSON string and converts it into a JavaScript object.
JSON.stringify( )	This method converts a JavaScript Object to a JSON string.