Andhra Pradesh State Skill Development Corporation

Skill AP
APSSDC

Skill AP
Learn Anytime Anywhere
Andhra Pradesh State Skill Development Corporation

Web designing using ReactJs

JavaScript

Java Script

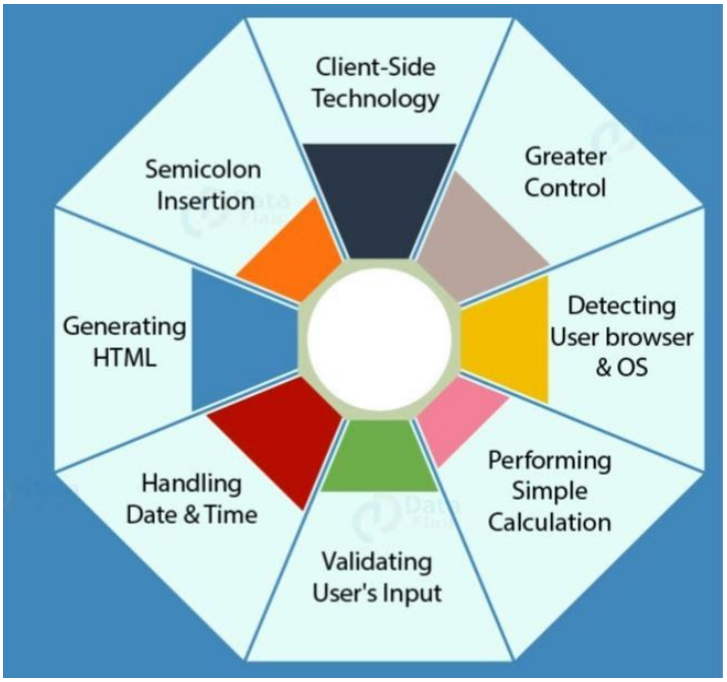
JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive. Where HTML and CSS are languages that give structure and style to web pages, JavaScript gives web pages interactive elements that engage a user.

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

History of JavaScript

- For implementing the dynamic behaviour of a web site.
- It was called a live script in its initial stages.
- Brendan Eich (Netscape navigator) was introduced this JavaScript.(September 1995)
- JavaScript is an interpreter language.
- He tied up with ECMA (Ecmascript) - ECMASCRIPT - 262 (ES-262)
- ECMASCRIPT -2018 is the latest version (ES-9)
- For implementing high end JS modules, We've to focus on ES-6.

Features of JavaScript

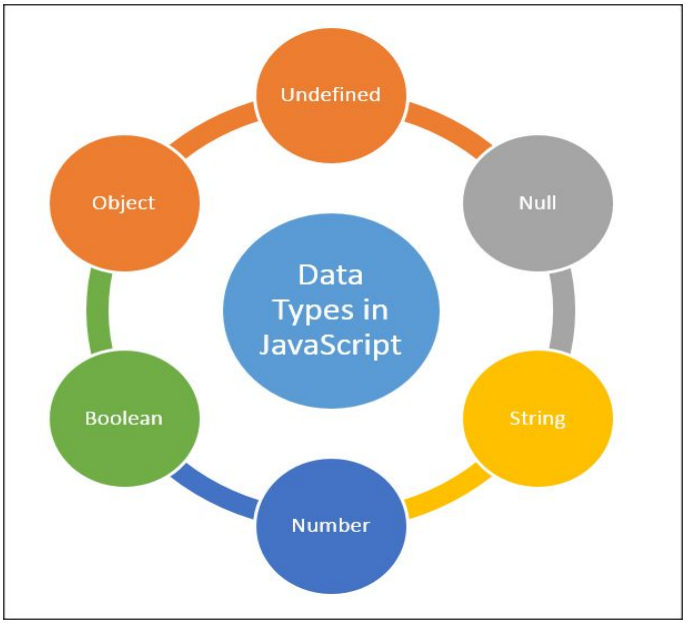


- Client-side technology
  - *JavaScript is a client-side technology. It is mainly used for giving client-side validation. It is an object-based scripting language.*

- Semicolon insertion
  - *All statements in JavaScript must terminate with a semicolon. Most of the JavaScript control statements syntax is the same as control statements in C.*
- Generating HTML
  - *Every time you click on the button, the browser generates and displays a new HTML code. Thanks to JavaScript, this operation performs on the client- machine and therefore you don't have to wait while the information backs and forth between your browser and the web server.*
- Handling Date & Time
  - *The JavaScript says, "Nice Morning, isn't it? Or "Good Afternoon", depending on the current time. It also tells you today's date. Nice Morning isn't it? Today is Wednesday, 20 February 2019.*
- Validating User Input
  - *In the JavaScript calculator, try to type some letters instead of numeric input, you will get an error: Invalid input character. Note that JavaScript helps the browser perform output validation without wasting the user's time by the web server access.*
- Performing Simple calculations
  - *Using a JavaScript calculator, we perform simple calculations on the client side.*
- Detecting user browser & OS
  - *The feature to detect the user's browser and OS enables your script to perform platform–dependent operations.*
- Greater control
  - *It gives the user more control over the browser. Example — You can change the background color of the page as well as text on the browser's status bar. Here, the back button is implemented with JavaScript. Click it and you will return to the page from which you have arrived.*

Data types

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



- **Number**
  - All kinds of numbers (Integers, floating numbers and double precision numbers) come under this category.

```
var num=10;
```

- **String**

```
var str="Hello";
```

- **Boolean**

```
var isNum=true;
```

- **Function**

```
function _function-name_{
{// Function body }
}
```

- **Object**

```
var car = {type:"Fiat", model:"500", color:"white"};
```

- **Null**

```
var value=null;
console.log(typeof(value))
```

Output: Object

- **undefined**

```
var val;
console.log(val);
```

Output: Undefined

Conversion functions

Most of the time, operators and functions automatically convert the values given to them to the right type.For example, alert automatically converts any value to a string to show it. Mathematical operations convert values to numbers.There are also cases when we need to explicitly convert a value to the expected type.

Numeric conversion *Numeric conversion happens in mathematical functions and expressions automatically.*

- Number()

```
var str="123";
str=Number(str);
console.log(typeof(str));
```

Output: Number

- parseInt()

```
var val=123.89
val=parseInt(val);
console.log(val);
```

Output: 123

- parseFloat()

```
var val=123.89
val=parseFloat(val);
console.log(val);
```

Output: 123.89

String Conversion

- String conversion happens when we need the string form of a value.
- For example, alert(value) does it to show the value.
- We can also call the String(value) function to convert a value to a string:

```
let value = true;
alert(typeof value); // boolean
```

```
value = String(value); // now value is a string "true"
alert(typeof value); // string
```

Boolean conversion

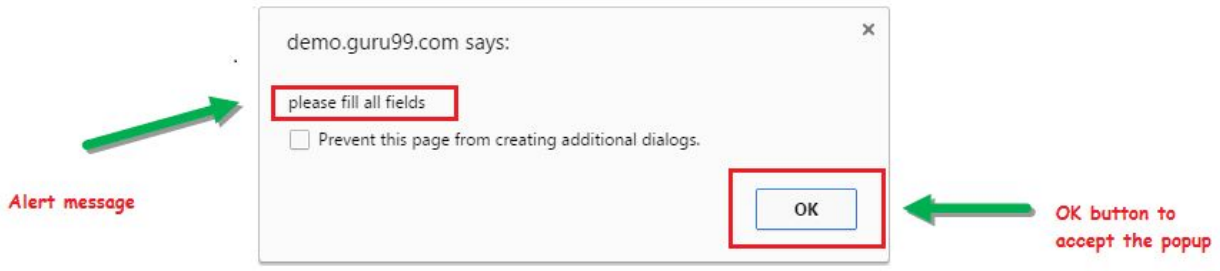
- Boolean conversion is the simplest one.
- It happens in logical operations (later we'll meet condition tests and other similar things) but can also be performed explicitly with a call to Boolean(value).
- The conversion rule:
  - Values that are intuitively "empty", like 0, an empty string, null, undefined, and NaN, become false. Other values become true.

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false

alert( Boolean("hello") ); // true
alert( Boolean("") ); // false
```

Alerts in JavaScript

In Javascript, popup boxes are used to display the message or notification to the user. There are three types of pop up boxes in JavaScript namely Alert Box, Confirm Box and Prompt Box. Alert Box: It is used when a warning message is needed to be produced.



The following are the kinds of alerts.

- alert()

```
alert("Hi");
```

- prompt()

```
prompt("Enter your name");
```

- confirm()

```
confirm("Are you sure?");
```

Example:

```
if(confirm("Are you sure?")==true){
    console.log("Clicked on okay");
} else {
    console.log("Clicked on cancel");
}
```

Console statements

The Console can be used to log information as part of the JavaScript development process, as well as allow you to interact with a web page by carrying out JavaScript expressions within the page's context. Essentially, the Console provides you with the ability to write, manage, and monitor JavaScript on demand.

- **console.log()**

```
console.log("This is for displaying an output");
```

- **console.info()**

```
console.info("This is also for displaying the output");
```

- **console.warn()**

```
console.warn("This is a warning");
```

- **console.error()**

```
console.error("Oops! this is error2");
```

Functions:

- **Functions without parameters** (static functions):

```
// defining a function
function add(){
    var a=1;
    var b=2;
    return a+b;
}

//Calling the above function
add()
```

- **Functions with parameters** (Dynamic functions)

```
function multiply(x,y){
    return x*y;
}

multiply(2,3)
```

- **Function without name** (Anonymous functions)

```
var object=function(a,b){
    return a*b;
}

object(3,4)
```

Output: 12

- **Arrow functions**

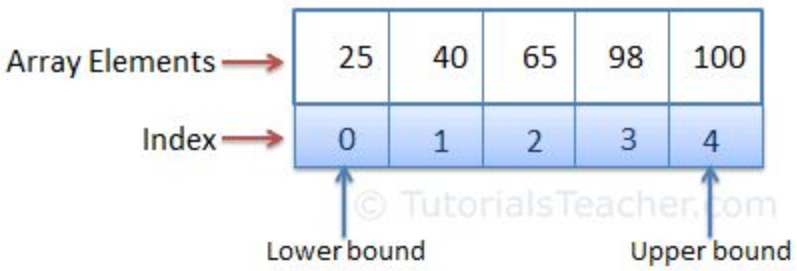
```
var multiply=(x,y)=>{
    return (x*y)
}

multiply(5,6);
```

Output: 30

Array & its functions

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An **array** is used to store a collection of data, but it is often more useful to think of an **array** as a collection of variables of the same type.




```
typeof([])
// Object

typeof({})
//Object

Array.isArray([])
// true

Array.isArray({})
// false

array

// [4, 5, 5, 4, 4, 3, 2, 1]
```

```
array.length
// 8
```

Array.find()

```
array
// [5, 5, 5, 4, 4, 3, 2, 1]

let graterFour=(n)=>{
      return n>4
}

array.find(graterFour)
// 5
```

Array.slice()

```
array
// [4, 5, 5, 4, 4, 3, 2, 1]

array.slice(0,3)
// [4, 5, 5]
```

Array.filter()

```
array
// [4, 5, 5, 4, 4, 3, 2, 1]

array.filter(n=>n>2)
// [4, 5, 5, 4, 4, 3]
```

Array.includes()

```
array
// [4, 5, 5, 4, 4, 3, 2, 1]
array.includes(1)
// true
```

Array.pop()

```
array
// [4, 5, 5, 4, 4, 3, 2, 1]

array.pop()
// 1
```

Array.push

```
array
// [4, 5, 5, 4, 4, 3, 2]

array.push(1)
// [4, 5, 5, 4, 4, 3, 2, 1]
```

Array.toString()

```
array
// [4, 5, 5, 4, 4, 3, 2, 1]

array.toString()
// "4,5,5,4,4,3,2,1"
```

Array.splice()

```
var array=[1,2,3,4,5,6];
delete array[1];

array : [1,empty,3,4,5,6] // The element gets deleted but the position replaced with `empty`.

// We have to use splice for deleting the item permanently.
array.splice(1,1);
array : [1,3,4,5,6]  // `1` indicated the index value, and `1` indicates the number of elements to be deleted.
```

Array.indexOf()

```
array
  =>(5) [1, 3, 4, 5, 6]
array.indexOf(null)
  -1
array.indexOf(3)
  1
```

Array.sort() It returns an array by sorting the elements of it in ascending order.

```
var array=[1,5,4,5,3,4,2];
array.sort()
  => (7) [1, 2, 3, 4, 4, 5, 5]
array
  => (7) [1, 2, 3, 4, 4, 5, 5]
array.reverse()
  => (7) [5, 5, 4, 4, 3, 2, 1]
```

Array.shift()

```
array
// [4, 5, 5, 4, 4, 3, 2, 1]

array.shift()
// 4

array
// [5, 5, 4, 4, 3, 2, 1]

array.unshift(4)
// 8  (Size of an array)
```

```
array
[4, 5, 5, 4, 4, 3, 2, 1]
```

Array iteration using map,for-in and for-of

- **Map()**

  Syntax:

  ```
  arrayName.map({arrow function})
  ```

  ```
  var names=["mark zuckerberg","Jack dorsey","Tim","Jack ma"];
  // iter holds the values and index holds the index values of an array
  names.map((iter,index)=>{
     console.log(iter);
     console.log(index);
  })
  ```

- **for-in**

  ```
  var names=["mark zuckerberg","Jack dorsey","Tim","Jack ma"];
  for(i in names){
     console.log(i);
  }
  ```

- **for-of**

  ```
  var names=["mark zuckerberg","Jack dorsey","Tim","Jack ma"];
  for(i of names){
          console.log(i);
   }
  ```

- **for-each**

  ```
  var array=[1,2,3,4,5,6,5,4,3,2]
  array.length
  // 10
  ```

```
array.forEach((item,index,array)=>{
   console.log(item+"is having"+index+"index in"+array)
})
```

Destructuring Assignment

Array destructuring Destructuring assignment is a special syntax that allows us to “unpack” arrays or objects into a bunch of variables, as sometimes that’s more convenient. Destructuring also works great with complex functions that have a lot of parameters, default values, and so on.

```
let arr = ["Ilya", "Kantor"]

// destructuring assignment
// sets firstName = arr[0]
// and surname = arr[1]
let [firstName, surname] = arr;

alert(firstName); // Ilya
alert(surname);  // Kantor

let [firstName, surname] = "Ilya Kantor".split(' ');


// second element is not needed
let [firstName, , title] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];

alert( title ); // Consul
```

Objects destructuring

```
let options = {
    title: "Menu",
    width: 100,
    height: 200
    };

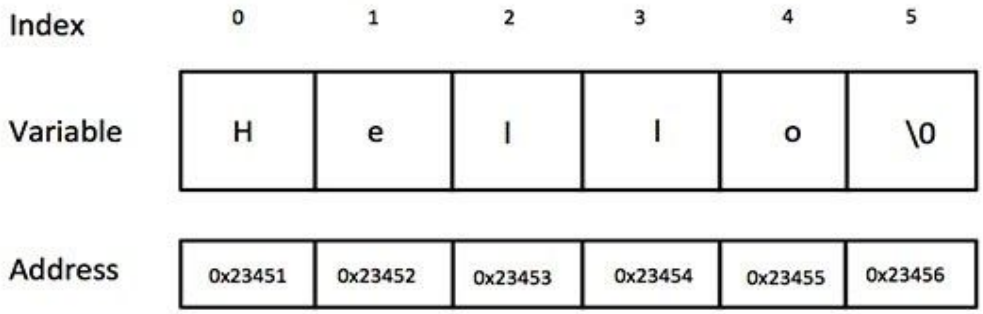
// { sourceProperty: targetVariable }
let {width: w, height: h, title} = options;

// width -> w
// height -> h
// title -> title

alert(title);  // Menu
alert(w);      // 100
alert(h);      // 200
```

Strings in JavaScript

A String is a group of characters. In other words, we can call a string as a character array.



String functions

concat() : By using concat method, we can join two or more strings.

```
var str1="Hanuman";
var str2="Kumar";

str1.concat(str2)
// HanumanKumar

str1.concat(" ",str2)
// Hanuman Kumar
```

charAt() : The charAt() method returns the character at the specified index in a string.

```
var str1="Hanuman Kumar";
str1.charAt(0)
// H
```

indexOf() : The indexOf() method returns the position of the first occurrence of a specified value in a string

```
str
// Hanuman

str.indexOf("n")
// 2

str.indexOf("Kumar")
// -1 because the specified string does not exist in the string.
```

replace() : We can replace a string with customized values.

```
str
// Hanuman

str.replace("Hanuman", "Rajesh")
// Rajesh
```

```
str
// Hanuman

str=str.replace("Hanuman", "Rajesh")
// Rajesh

str
// Rajesh
```

search() The search() method searches a string for a specified value, and returns the position of the match.

```
str
// Rajesh

str.search("R")
// 0

str.search("eh")
// -1
```

substr() The substr() method extracts parts of a string, beginning at the character at the specified position, and returns the specified number of characters.

```
str
// Rajesh

str.substr(1,4)
// ajes
```

substring() The substring() method extracts the characters from a string, between two specified indices, and returns the new substring.

```
str
// Rajesh

str.substring(1,4)
// aje
```

split() The split() method is used to split a string into an array of substrings, and returns the new array.

```
var str="Hanuman Kumar";
str.split(" ");
// ["Hanuman","Kumar"]
```

includes() The includes() method determines whether a string contains the characters of a specified string.

```
str
// Hanuman Kumar
```

```
str.includes("Hanuman")
// true

str.includes("Hi")
// false
```

startsWith() The startsWith() method determines whether a string begins with the characters of a specified string.

```
str
// Hanuman Kumar

str.startsWith("K")
// flase

str.startsWith("H")
// true
```

endsWith() The endsWith() returns true if a string ends with the characters of a specified string; otherwise false.

```
str
// Hanuman Kumar

str.endsWith("Kumar")
// true

str.endsWith("Rajesh")
// false
```

toUpperCase() converts all the characters of a string to uppercase.

```
str
// Hanuman Kumar

str.toUpperCase()
// HANUMAN KUMAR
```

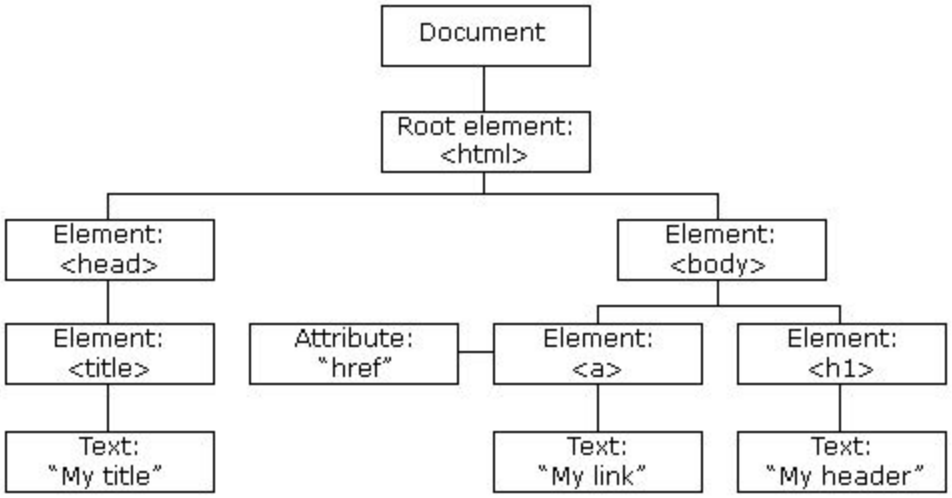
toLowerCase() converts all the characters of a string to uppercase.

```
str
// Hanuman Kumar

str.toLowerCase()
// hanuman kumar
```

Document Object Model (DOM):

- We can implement JavaScript code anywhere in an html document.
- We've to use script tags for that (<script> </script>)
- By using the document keyword, We can manipulate the data in the html document.



- Document object model functions are:

```
document.getElementById("id");
```

- Collections

```
document.getElementsByClassName("className");
document.getElementsByTagName("Tag name");
```

- Common statements for DOM

```
document.querySelector("#selector | .selector | Tag Name Selector");
document.querySelectorAll(".selector | Tag Name Selector");
```

innerHTML

innerHTML triggers the information that exists within the tags. Example:

```
<div id="second">
   Sample division
</div>

var second=document.getElementById("second").innerHTML;
console.log(second);
```

Output: Sample division

Writing information into divisions using innerHTML

```
document.getElementById("second").innerHTML="Hello everyone";
```

Creating and appending HTML attributes We can create the html attributes by using javascript code part also. For that we need to use document.createElement. And we can append the information into the created elements by using textContent.

Syntax:

```
document.createElement("html tagname")
```

Example:

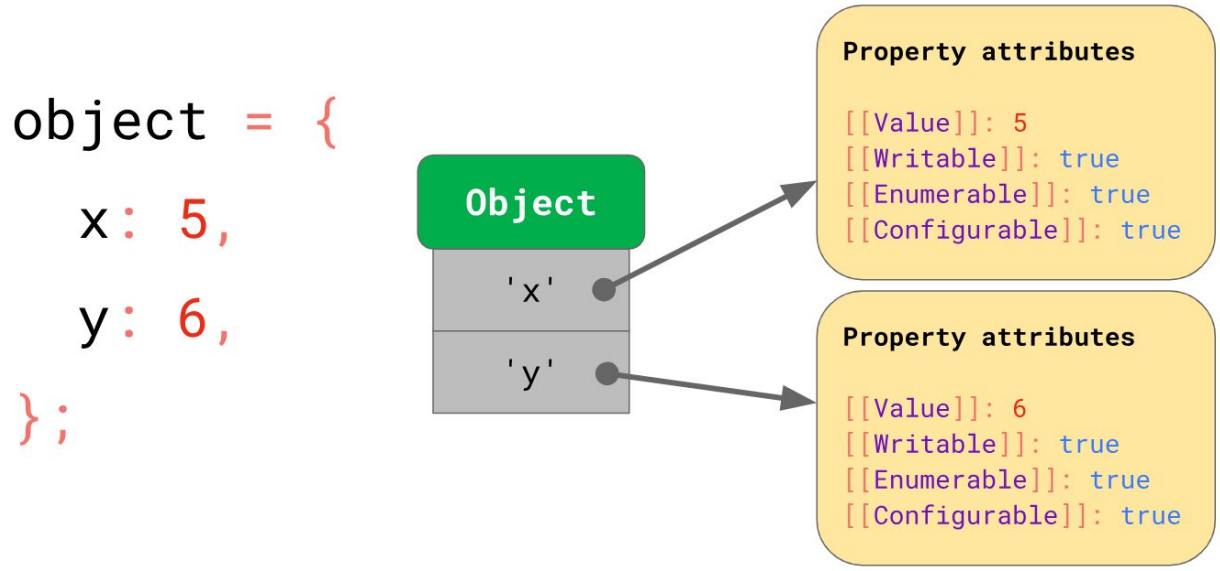
```
//Selecting a section from html;
var secondDiv=document.getElementById("second");

// Creating a html attribute
var heading2=document.createElement("h2");
heading2.textContent="heading";

//Appending child attribute to parent
secondDiv.appendChild(heading2);
```

Objects in Javascript

Object is nothing but its key,value pair. When we going to get the value from an object, we've to use the key of that object



Example:

```
var student={
   name:"Hemanth sai",
   role:"Student",
   branch:"CSE:
}
```

student.name => Hemanth sai

Arrays in the object:

```
var student={
   friends:["sai vasanth","savya sree", "nitish bharti","venkatesh"]
}

_____
// Here `0` is the index value
student.friends[0]

_____
Output: sai vasanth


_____
student.friends[student.friends.length-1]

____
Output: Venkatesh
```

Objects in Object:

```
var student={
   student1:{
      name:"Swetha"
   },
   student2:{
      name:"venkatesh"
   }
}

_____
student.student1.name

_____
Output: Swetha
```

Methods with Object:

```
let user = {
name: "Hanuman",
age: 26
};

user.sayHi = function() {
alert("Hello!");
};

user.sayHi(); // Hello!
```

- Here we've just used a Function Expression to create a function and assign it to the property user.sayHi of the object.
- Then we can call it as user.sayHi(). The user can now speak!
- A function that is a property of an object is called its method.
- So, here we've got a method sayHi of the object user. Of course, we could use a pre-declared function as a method, like this:

```
let user = {
// ...
```

```
};

// first, declare
function sayHi() {
  alert("Hello!");
};

// then add as a method
user.sayHi = sayHi;

user.sayHi(); // Hello!

// these objects do the same

user = {
  sayHi: function() {
    alert("Hello");
  }
};

// method shorthand looks better, right?
user = {
  sayHi() { // same as "sayHi: function()"
    alert("Hello");
  }
};
```

Object Functions

The Object.keys() method returns an array of a given object's own enumerable property names, iterated in the same order that a normal loop would.

```
```javascript
const myDaya = {
name: 'Hanuman Kumar P',
role: 'Multi-skill Trainer',
organization: 'APSSDC'
};

console.log(Object.values(myData));
// expected output: Array ["name", "role", "organization"]
```

Example 2:

```
// simple array
const arr = ['a', 'b', 'c'];
console.log(Object.keys(arr)); // console: ['0', '1', '2']
```
```

```
// array-like object
const obj = { 0: 'a', 1: 'b', 2: 'c' };
console.log(Object.keys(obj)); // console: ['0', '1', '2']

// array-like object with random key ordering
const anObj = { 100: 'a', 2: 'b', 7: 'c' };
console.log(Object.keys(anObj)); // console: ['2', '7', '100']

// getFoo is a property which isn't enumerable
const myObj = Object.create({}, {
getFoo: {
    value: function () { return this.foo; }
}
});
myObj.foo = 1;
console.log(Object.keys(myObj)); // console: ['foo']
```

The Object.values() method returns an array of a given object's own enumerable property values, in the same order as that provided by a for...in loop.

```
const myDaya = {
name: 'Hanuman Kumar P',
role: 'Multi-skill Trainer',
organization: 'APSSDC'
};

console.log(Object.values(myData));
// expected output: Array ["Hanuman Kumar P", "Multi-skill Trainer", "APSSDC"]
```

The Object.entries() method returns an array of a given object's own ienumerable string-keyed property [key, value] pairs, in the same order as that provided by a for...in loop.

The order of the array returned by Object.entries() does not depend on how an object is defined. If there is a need for certain ordering, then the array should be sorted first, like Object.entries(obj).sort((a, b) => b[0].localeCompare(a[0]));.

```
const myData = {
name: 'Hanuman Kumar P',
age: 26,
organization: "APSSDC"
};

for (const [key, value] of Object.entries(myData)) {
   console.log(`${key}: ${value}`);
}

// expected output:
// "name: Hanuman Kumar P"
// "age: 26"
// "organization": "APSSDC"
```

Regular Expressions

```
// Defining a regular expression
var reg=/abc/
console.log(reg.test("abcdefghijk"))
// true

// test() return a boolean value as result


var reg=/^a....s$/;
consle.log(reg.test("abbbbs"))

// true
```

Here ^ is for defining the starting position of a string and $ is for ending position. The below example shows how to implement regular expressions using the constructor method.

```
var reg=new RegExp(/^a...b$/);
console.log(reg.test("assb"));
// false
```

Meta characters

[] => specify a set of characters you wish to match.

```
var reg=/[abc]/;
console.log(reg.test("aabbddffccgh"))
// true

console.log(reg.test("a"))
// true

console.log(reg.test("z"))
// false
```

[a-h] is same as [abcdefgh]

[0-9] is same as [0,1,2,3,4,5,6,7,8,9]

[^abc] => Invert character set (*Matching any character except a or b or c*)

```
var reg=/[^abc]/;
console.log(reg.test("abc"))
// false

console.log(reg.test("bc"))
// false

console.log(reg.test("c"))
// false
```

```
console.log(reg.test("xyz"))
// true
```

. Matching any single character except \n

```
var reg=/.../;
console.log(re.test('aaa'))
// true

console.log(reg.test('aa'))
// false

console.log(reg.test('a'))
// false

console.log(reg.test('bbcsdfgg'))
// true
```

^ For checking starting position

```
var reg=/^ab/;

console.log(reg.test("abccsddf"))
// true

console.log(reg.test("baccdss"))
// false
```

$ For checking ending position

```
var reg=/ab$/;

console.log(reg.test("abccsddf"))
// false

console.log(reg.test("grhdjersdab"))
// true
```

* matches zero or more occurrences of the pattern left to it.

```
var reg=/han*u/;
console.log(reg.test("hanuman"))
// true

console.log(reg.test("hannuman"))
// true

console.log(reg.test("haguman"))
// false
```

+ matches one or more occurrences of the pattern left to it.

```
var reg=/han+u/
console.log(reg.test("hanuman"))
// true

console.log(reg.test("hanman"))
// false
```

* vs +

```
console.log(/han*u/.test("hau"))
// true

console.log(/han+u/.test("hau"))
// false
```

? Optional (Checks from left)

```
var reg=/han?u/
console.log(reg.test("hau"))
// true

console.log(reg.test("hanu"))
// true

console.log(reg.test("hannu"))
// false (Exceeds number of characters)

console.log(reg.test("hamu"))
// false (`u` is not followed by n)
```

{} for defining the range

```
var reg=/h{2,3}/;
console.log(reg.test("hanu"))
// false

console.log(reg.test("hhanu"))
// true

console.log(reg.test("hhhhanu"))
// true

reg=/^h{2,3}anu$/
console.log(reg.test("hhhhanu"))
// false

console.log(reg.test("hhhanu"))
// true
```

| Or

```
var reg=/a|b/
console.log(reg.test("cde"))
// false

console.log(reg.test("ade"))
// true
```

() Grouping

```
var reg=/(a|b|c)xyz/
console.log(reg.test("ab xyz"))
//false

console.log(reg.test("axyz"))
// true
```

Special sequences

Special sequences make commonly used patterns easier to write. Here's a list of special sequences:

\d - Matches any decimal digit. Equivalent to [0-9]

\D - Matches any non-decimal digit. Equivalent to [^0-9]

\s - Matches where a string contains any whitespace character. Equivalent to [\t\n\r\f\v]

\S - Matches where a string contains any non-whitespace character. Equivalent to [^ \t\n\r\f\v]

\w - Matches any alphanumeric character (digits and alphabets). Equivalent to [a-zA-Z0-9_]. By the way, underscore _ is also considered an alphanumeric character.

\W - Matches any non-alphanumeric character. Equivalent to [^a-zA-Z0-9_]

Regular Expression Flags:

Flags	Description
g	Performs a global match (find all matches)
m	Performs multiline match
i	Performs case-insensitive matching

Example : Validating Email using Regular expressions

```
// program to validate the email address

function validateEmail(email) {

    // regex pattern for email
    const re = /\S+@\S+\.\S+/g;

    // check if the email is valid
    let result = re.test(email);
    if (result) {
        console.log('The email is valid.');
    }
    else {
        let newEmail = prompt('Enter a valid email:');
        validateEmail(newEmail);
    }
}

// take input
let email = prompt('Enter an email: ');

validateEmail(email);
```

Output :

```
// Enter an email: hellohello
// Enter a valid email: learningJS@gmail.com
// The email is valid.
```

JSON

- **J**ava**S**cript **O**bject **N**otation
- Lightweight data-interchange format (This is a format to transfer information between the nodes)
- This is very easy to read,understand and write.
- This is very easy for the machines to parse and generate data.
- This is a subset of javascript programming language standards (ECMASCRIPT - 262).
- For accessing JSON data from a HTML document, we need to use a server.
  - npm server
  - web server for chrome (200 OK)
    - Web server for chrome is a static server invented google team. This allows us to access any information from external files into a HTML document.
    - **Installation of 200 OK**
      - *Type **web server for chrome** in your chrome browser* and press Enter.

- *We have to choose a web server for chrome from chrome.google.com from listing*.
- *We have to click on the add to chrome button. Soon after we will see a dialogue box and then click on the add app*.

Promises:

A promise is an object that may produce a single value some time in the future. The client and the server communicate with each other, client needs to send a request to the server, the server needs to respond to the client request simultaneously. This promise gives us a value that is resolved or not resolved. The states for the promise are

- Fulfilled : meaning that the operation was completed successfully.
- Rejected : meaning that the operation failed.
- Pending : initial state, neither fulfilled nor rejected.

The methods promise.then(), promise.catch(), and promise.finally() are used to associate further action with a promise that becomes settled. These methods also return a newly generated promise object, which can optionally be used for chaining; for example, like this:

```
const myPromise =
(new Promise(myExecutorFunc))
.then(handleFulfilledA,handleRejectedA)
.then(handleFulfilledB,handleRejectedB)
.then(handleFulfilledC,handleRejectedC);

// or, perhaps better ...

const myPromise =
(new Promise(myExecutorFunc))
.then(handleFulfilledA)
.then(handleFulfilledB)
.then(handleFulfilledC)
.catch(handleRejectedAny);
```

Handling a rejected promise too early has consequences further down the promise chain. Sometimes there is no choice because an error must be handled immediately; in such cases we must throw something, even if it is a dummy error message like throw -999, to maintain error state down the chain. On the other hand, in the absence of an immediate need it is simpler to leave out error handling until a final .catch() statement.

The promise is a chain relation, We can build multiple coding blocks. The syntax:

```
promise(request).then(response=>{
   return response;
})
```

Promises examples:

- Fetch
- Cache

Example:

```
fetch('data.json').then(response=>{
    return response.json();
}).then(data=>{
    console.log(data);
})
```

The promise concept is the replacement for XMLHttpRequest. The XMLHttpRequest provides a way to communicate with client and server