Full Stack Web Development with Python and DJango

JavaScript

Study Material

Full Stack Web Development with Python and DJango

JavaScript

HTML is for Nouns
like Anushka,input fields,buttons,forms etc

CSS is for Adjectives
Like Styling with colors,borders,background images etc

Java Script is for Verbs/Actions
Like Dance,Eat....

Java Script is Full pledged Programming Language.
The main purpose of java script is to add functionality(actions) to the HTML.

Usually we can Java Script in the Front-end and we can use Node JS for Back-end.

Agenda:

1. Java Script Developer's Console
2. The 5 Basic Javascript Primitive Data Types
3. Declaring variables with var keyword
4. The 3 most commonly used Java Script Functions

1)JavaScript Developer's Console

We can use this developer's console to test our java script coding snippets. This is just for testing purpose only and usaully not recommended for main coding.

How to launch Java Script Console:

Browser-->Right Click-->Inspect-->Console

Short-cut: Ctrl+Shift+j

Note:

1. To clear console we have to use clear() function.
2. ; at end of statement is not mandatory in newer versions

2)The 5 Basic JavaScript Primitive Data Types

Java Script defines the following 5 primitive data types

1. Numbers:

10
-10
10.5

All these are of "number" type
Java Script never cares whether it is integral or float-point or signed and unsigned.

General Mathematical operators are applicable for numbers

10+20
10-20
10/20
10*20
10%3
102**

General Operator precedence also applicable.
10+20*3===>70
10*(2+3)===>50

Note: We can check the type of variable by using typeof keyword

typeof x;

2. string:

Any sequence of characters within either single quotes or double quotes is treated as string.

'durga'
"durga"

We can apply + operator for Strings also and it acts as concatenation operator.

Rule: If both arguments are number type then + operator acts as arithmetic addition operator.
If atleast one argument is of string type then + operator acts as concatenation operator.

10+20==>30
'durga'+10==>durga10
'durga'+true==>durgatrue

We can use esacape characters also in the string.

Eg: 'durga\nsoft'
'durga\tsoft'
'This is \' symbol'
'This is \" symbol'
'This is \\ symbol'

Q. How to find the number of characters present in the string?

We can find by using length variable

Eg:
'durgasoft'.length

Q. How to access characters of the String?

By using index

index of first character is zero

'durga'[2]==>r
'durga'[200]===>undefined but no error
'durga'[-1] ==>undefined

Note: **If we are trying to access string elements with out of range index or negative index then we will get undefined value and we won't get any Error.**

3. boolean:

The only allowed values are: true and false (case sensitive)

3)JavaScript Variables

Variables are containers to store values.

Syntax:
var variableName=variableValue

Eg:
var name="durga"
var age=60
var isMarried=false

Note:
Java script variables follow CamelCase Convention

studentMobileNumber--->Camel case(Java,JavaScript etc)
student_mobile_number-->Snake Case(Python)
student-mobile-number-->Kebab Case(Lisp)
Based on provided value automatically type will be considered for variables.

Eg:
var x =10
typeof x ===>number
x = false
typeof x===>boolean

Hence Java Script is Dynamically Typed Programming Language like Python

null and undefined:

Variables that are declared but not initialized, are considered as undefined

Eg:

var x;
typeof x==>undefined

null value means nothing.
If the value of a variable null means that variable not pointing to any object.

var currentplayer='durga'
currentplayer=null //game over

The 3 most commonly used methods of Java Script:

1. alert():

To display alerts to the end user

```
alert('Hello there')
alert(100000)
alert(10.5)
```

2. console.log():

To print messages to the developer's console

Eg:

```
console.log('Hello there')
console.log(10*20)
```

These console message not meant for end user.

3. prompt():

To get input from the end user

```
prompt('What is Your Name:')
```

Here we are not saving the value for the future purpose. But we can save as follows

```
var name= prompt('What is Your Name:')
```

Based on our requirement we can use this name

```
console.log('Hello '+name+' Good Evening')
alert('Hello '+name+' Good Evening')
```

How to write Javascript to a seperate file and connect to html:

demo.js:

```
alert('Hello everyone good evening')
```

html:

We can link javascript file to html by using the following <script> tag.

```
<script type="text/javascript" src="demo.js"></script>
```

We can take this script tag either inside head tag or body tag. If we are taking inside head tag then javascript code will be exeucted before processing body.

If we are taking inside body tag then javascript code will be executed as the part of body execution.

Demo Application: Age and Death Calculator:

demo.js:

```
1) var name=prompt('Enter Your Name:');
2) var age=prompt('Enter Your Age:');
3) agedays=age*365.25
4) remainingdays=(60-age)*365.25;
5) alert("Hello "+name+"...\nYour Current Age:"+agedays+" days\nYou will be there on the e
   arth only "+remainingdays+" days. No one can change including God also");
```

demo.html:

```
1)  <!DOCTYPE html>
2)  <html lang="en" dir="ltr">
3)    <head>
4)      <meta charset="utf-8">
5)      <title></title>
6)    </head>
7)    <body>
8)      <h1>The power of Java Script</h1>
9)      <script type="text/javascript" src="demo.js"> </script>
10)   </body>
11) </html>
```

Operators:

1. Arithmetic Operators:

+,-,*,/,%,**

2. Comparison Operators:

<,<=,>,>=,==,!=,===, !==

10<20===>true
10<=20 ==>true
10>20 ===>false
10>=20 ==>false
10==20==>false
10 != 20 ==>true

Difference between == and ===:

In the case of == operator internally type coersion will be performed. Hence if arguments are different types first both arguments will be converted to same type and then comparison will be performed. Hence both arguments need not be same type.

Eg:

10 == "10" ====>true
10==10 ==>true

Here only values are important but not types.

But in the case === operator,type coersion won't be performed. Hence argument types must be same,otherwise we will get false.

10 === 10 ==>true
10 === "10" ===>false

Here both content and type are important.

Note:

== -->Normal equality operator where types are not important but values must be same

=== --->Strict equality operator where both types and values must be same

It is recommended to use === operator because it is more safer and more specific.

Example:

true=="1" ===>true
false=="0" ===>true
null==undefined ===>true
true==="1" ===>false
false==="0" ===>false
null===undefined ===>false

NaN(Not a Number):

If the result is undefined then we will get NaN

Eg: 0/0 ===>NaN

for any x value including NaN the following expressions returns false

x<NaN
x<=NaN
x>NaN
x>=NaN
x==NaN

For any x value including NaN the following expression returns true

x != NaN

Eg:

NaN==NaN ===>false
NaN != NaN ===>true

Logical Operators:

&& -->AND
|| -->OR
! --->Not

X && Y==>If both arguments are true then only result is true. i.e if atleast one argument is false then the result is always false

X || Y ===>If atleast one argument is true then the result is true. i.e if both arguments are false then only result is false.

Note: For Logical Operators

1. zero value always treated as false
non-zero value always treated as true

2. empty string treated as false where as non-empty string treated as true

3. null,undefined,NaN are treated as false

Examples:

```
var x =10;
var y =20;

x<10 && x != 5  ===>false

y>9 || x== 10 ==>true

!(x==y)  ==>true

!(x=="10" || x=== y) && !( y!= 8 && x<=y)) ==>false

!(x !==1)&& y === "20" ==>false
```

Conditional Statements:

Based on available options, one option will be selected and executed in conditional statements/selection statements.

1. if
2. if else
3. else if

Syntax:

```
if(b){
  action if b is true;
}
else{
 action if b is false;
}
```

Eg 1: Write Java Script code to check given number is even or not?

demo.html:

```
1)  <!DOCTYPE html>
2)  <html lang="en" dir="ltr">
3)    <head>
4)      <meta charset="utf-8">
```

```
5)      <script type="text/javascript" src="demo.js"></script>
6)
7)     <title></title>
8)    </head>
9)    <body>
10)    <h1>The power of Java Script</h1>
11)
12)   </body>
13)
14) </html>
```

demo.js:

```
1)  var num = Number(prompt('Enter Any Number:'))
2)  if(num%2===0){
3)    console.log('Given Number is Even')
4)    alert('Given Number is Even')
5)  }
6)  else{
7)    console.log('Given Number is Odd')
8)    alert('Given Number is Odd')
9)  }
```

Q2. Write Java Script code to print proper meaningful message based on provided age regarding matrimonial website?

demo.js:

```
1)  var age = Number(prompt('Enter Your Age:'))
2)  if(age>60){
3)    alert('Plz wait some more time..Defenitely You will get Best Match')
4)  }
5)  else if(age<18){
6)    alert("Your age already crossed marriage age..No chance of getting marriage")
7)  }
8)  else{
9)      alert('Thanks for registration..You will get match details soon by email')
10) }
```

Q3. Write Java Script code to print proper meaningful message based on provided brand regarding beer application?

```
1)  var brand = prompt('Enter Your Favourate Brand:')
2)  if(brand=="KF"){
3)    alert("It is Children's Brand")
4)  }
5)  else if(brand=="KO"){
6)    alert("It is too light")
```

```
}
else if(brand=="RC"){
  alert("It is not that much kick")
}
else if(brand=="FO"){
  alert("Buy One get One FREE")
}
else{
   alert('Other brands are not recommended')
}
```

Q. Number Guess Application:

```
var sno=4
var num = Number(prompt('Enter your guess between 1 to 9:'))
if(num>sno){
  alert("It is too high..Guess again")
}
else if(num<sno){
  alert("It is too low guess again")
}
else{
   alert('Your Guess Matched')
}
```

Iterative Statements:

If we want to execute a group of statements iteratively, then we should go for iterative statements.

DRY Principle: Don't Repeat Yourself
It is highly recommended to follow DRY principle, otherwise it increases development time
It increases length of the code and reduces readability

In JavaScript there are 2 types of iterative statements
1. While Loop
2. For Loop

1. While Loop:

As long as some condition is true execute code then we should go for while loop.

Syntax:

```
while(condition){
        body
 }
```

Eg 1: To print Hello 10 times to the console

demo.js:

```
1)  var count=1
2)  while(count<=10){
3)    console.log("Hello")
4)    count++
5)  }
```

Eg 2: To print first 10 numbers

demo.js:

```
1)  var count=1
2)  while(count<=10){
3)    console.log(count)
4)    count++
5)  }
```

Eg 3: To print each character present inside a string?

demo.js:

```
1)  var s="durga"
2)  var i =0
3)  while(i<s.length){
4)    console.log(s[i])
5)    i++
6)  }
```

Eg 3: To print all numbers divisible by 3 AND 5 between 5 and 100?

demo.js:

```
1)  var n=5
2)  while(n<=100){
3)    if(n%3==0 && n%5==0){
4)      console.log(n)
5)    }
6)    n++
7)  }
```

Eg 4: Write program to read actress name from the end user until entering 'sunny' by using while loop.

```
1)  var name=prompt("Enter Your Favourite Actress:")
2)  while(name !== "sunny"){
```

```
3)    name=prompt("Enter Your Favourite Actress:")
4)  }
5)  alert("Thanks for Confirmation as your Favourite actress: Sunny")
```

Note: If we don't know the number of iterations in advance and if we want to execute body as long as some condition is true then we should go for while loop.

Iterative Statements : For Loop:

If we know the number of iterations in advance then we should use for loop.

Syntax:

```
for(initialization section; conditional check; increment/decrement section)
{
   body;
}
```

Eg 1: To print "Hello" 10 times

```
for(var i=0;i<10;i++){
  console.log("Hello");
}
```

Eg 2: To print First 1 to 10

```
for(var i=1;i<=10;i++){
  console.log(i);
}
```

Eg 3: To print all numbers which are divisible by 7 from 1 to 100:

```
for(var i=1;i<=100;i++){
  if(i%7==0){
    console.log(i)
  }
}
```

Eg 4: To print each character from the given string

```
var word=prompt("Enter Some Word:")
for(var i=0;i<word.length;i++){
  console.log(word[i])
}
```

while vs for loop:

If we don't know the number of iterations in advance and as long as some condition is true keep on execute body then we should go for while loop.

If we know the number of iterations in advance then we should use for loop.

Secret Agent Application:

Rules:

1. The first character of Name should be 'd'
2. The Last character of Favourite Actor should be 'r'
3. The lucky number should be 7
4. The length of the dish should be >=6

If the above conditions are satisfied then user is valid secret agent and share information about operation, otherwise just send thanks message.

demo.js:

```
1)  var name=prompt("Enter Your Name:")
2)  var actor=prompt("Enter Your Favourite Actor:")
3)  var lucky=prompt("Enter Your Lucky Number:")
4)  var dish=prompt("Enter Your Favourite Dish:")
5)
6)  var nameConition=false
7)  var actorCondition=false
8)  var luckyConition=false
9)  var dishConition=false
10)
11) if(name[0]=="d") {
12)   nameConition=true
13) }
14) if(actor[actor.length-1]=="r"){
15)   actorCondition=true
16) }
17) if(lucky==7){
18)   luckyConition=true
19) }
20) if(dish.length>=6){
21)   dishConition=true
22) }
23) alert("Hello:"+name+"\nThanks For Your Information")
24) if(nameConition && actorCondition && luckyConition && dishConition){
25)   console.log("Hello Secret Agent our next operation is:")
26)   console.log("We have to kill atleast 10 sleeping students in the class room b'z these are b
      urdent to country")
27) }
```

Functions

If any piece of code repeatedly required in our application,then it is not recommended to write that code seperately every time. We have to seperate that code into a function and we can call that function where ever it is required.

Hence the main advantage of functions is code Reusability.

Syntax of Java Script function:

```
function functionName(arguments){
  body
  return value;
}
```

Eg 1: To print "Good Morning" message

```
1)  function wish(){
2)    console.log("Good Morning!!!")
3)  }
4)
5)  wish()
6)  wish()
7)  wish()
```

Functions with Arguments:

A function can accept any number of arguments and these are inputs to the function.
Inside function body we can use these arguments based on our requirement.

Eg: Write a function to accept user name as input and print wish message.

```
1)  function wish(name){
2)    console.log("Hello "+name+" Good Morning!!!")
3)  }
4)
5)  var name= prompt("Enter Your Name:")
6)  wish(name)
```

Functions with default arguments:

We can provide default values for arguments. If we are not passing any value then only default values will be considered.

Eg:

```
1) function wish(name="Guest"){
2)   console.log("Hello "+name+" Good Morning!!!")
3) }
4)
5) wish("Durga")
6) wish()
```

Function with return values:

A function can return values also.

Eg: Write a Javascript function to take a number as argument and return its square value

```
1) function square(num){
2)   return num*num;
3) }
4) var result=square(4)
5) console.log("The Square of 4:"+result)
6) console.log("The Square of 5:"+square(5))
```

Eg: Write a Javascript function to take 2 numbers as arguments and return sum.

```
1) function sum(num1,num2){
2)   return num1+num2;
3) }
4) var result=sum(10,20)
5) console.log("The sum of 10,20 :"+result)
6) console.log("The sum of 100,200 :"+sum(100,200))
```

Eg: Write a Javascript function to take a string as argument and return Capitalized string.

```
1) function capitalize(str){
2)   return str[0].toUpperCase()+str.slice(1);
3) }
4) console.log(capitalize('sunny'))
5) console.log(capitalize('bunny'))
```

Eg: Write a Javascript function to check whether the given number is even or not?

```
1) function isEven(num){
2)   if(num%2==0){
```

```
    return true;
  }
  else{
    return false;
  }
}
console.log(isEven(15))
console.log(isEven(10))
```

Eg: Write javascript function to find factorial of given number?

```
function factorial(num){
  result=1;
  for (var i = 2; i <= num; i++) {
    result=result*i;
  }
  return result;
}
console.log("The Factorial of 4 is:"+factorial(4))
console.log("The Factorial of 5 is:"+factorial(5))
```

Eg: Write a javascript function to convert from Snake case to Kebab case of given string.

snake case: total_number
Kebab case: total-number

```
function snakeToKebab(str){
  var newstring=str.replace('_','-')
  return newstring;
}
console.log(snakeToKebab('total_number'))
```

Note: Inside function if we are writing any statement after return statement,then those statements won't be executed, but we won't get any error.

Eg:

```
function square(n){
  return n*n;
  console.log("Function Completed!!!")
}
console.log(square(4));
```

Output: 16

JavaScript Scopes:

In Javascript there are 2 scopes.

1. Global Scope
2. Local Scope

1. Global Scope:

The variables which are declared outside of function are having global scope and these variables are available for all functions.

Eg 1:

```
1) var x=10
2) function f1(){
3)   console.log(x);
4) }
5) function f2(){
6)   console.log(x);
7) }
8) f1();
9) f2();
```

2. Local Scope:

The variables which are declared inside a function are having local scope and are available only for that particular function. Outside of the function we cannot these local scoped variables.

Eg 2:

```
1) function f1(){
2)   var x=10
3)   console.log(x);//valid
4) }
5) f1();
6) console.log(x);//Uncaught ReferenceError: x is not defined
```

Eg 3:

```
1) var x=10
2) function f1(){
3)   x=777;
4)   console.log(x);
5) }
6) function f2(){
7)   console.log(x);
```

```
8)  }
9)  f1();
10) f2();
```

Output:
777
777

Eg 4:

```
1)  var x=10
2)  function f1(){
3)    x=777;
4)    console.log(x);
5)  }
6)  function f2(){
7)    console.log(x);
8)  }
9)  f2();
10) f1();
```

Output:
10
777

Eg 5:

```
1)  var x=10
2)  function f1(){
3)    var x=777;
4)    console.log(x);
5)  }
6)  function f2(){
7)    console.log(x);
8)  }
9)  f1();
10) f2();
```

Output:
777
10

Q. If local and global variables having the same name then within the function local variable will get priority.How to access global variable?

Higher Order Functions:

We can pass a function as argument to another function.A function can return another function. Such type of special functions are called Higher Order Functions.

Eg: setInterval()

setInterval(function, time_in_milliseconds)
The provided function will be executed continously for every specified time.

setInterval(singAsong, 3000)

singAsong function will be executed for every 3000 milli seconds.

We can stop this execution by using clearInterval() function.

Eg: demo.js

```
1)  function singAsong(){
2)    console.log('Rangamma...Mangamma..')
3)    console.log('Jil..Jil...Jigel Rani..')
4)  }
```

On developer's console:

```
setInterval(singAsong,3000)
1
demo.js:2 Rangamma...Mangamma..
demo.js:3 Jil..Jil...Jigel Rani..
demo.js:2 Rangamma...Mangamma..
demo.js:3 Jil..Jil...Jigel Rani..
demo.js:2 Rangamma...Mangamma..
demo.js:3 Jil..Jil...Jigel Rani..
clearInterval(1)
undefined
```

Anonymous Functions:

Some times we can define a function without name, such type of nameless functions are called anonymous functions.

The main objective of anonymous functions is just for instant use (one time usage)

Eg:

```
setInterval(function(){console.log("Anonymous Function");},3000);
8
Anonymous Function
Anonymous Function
Anonymous Function
Anonymous Function
..
clearInterval(8);
```

Coding Examples from codebat.com:

Problem-1: sleep_in

Write a function called sleep_in that takes 2 boolean parameters: weekday and vacation.

The parameter weekday is True if it is a weekday, and the parameter vacation is True if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return True if we sleep in.

```
sleep_in(false, false) --> true
sleep_in(true, false) --> false
sleep_in(false, true) --> true
```

```
1)  function sleep_in(weekday,vacation) {
2)   return !weekday || vacation;
3)  }
4)  console.log("Is Employee Sleeping:"+sleep_in(true,true))
5)  console.log("Is Employee Sleeping:"+sleep_in(true,false))
6)  console.log("Is Employee Sleeping:"+sleep_in(false,true))
7)  console.log("Is Employee Sleeping:"+sleep_in(false,false))
```

Problem-2: monkey_trouble

We have two monkeys, a and b, and the parameters a_smile and b_smile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Return True if we are in trouble.

```
monkey_trouble(true, true) --> true
monkey_trouble(false, false) --> true
```

monkey_trouble(true, false) --> false

Solution:

```
1)  function monkey_trouble(aSmile,bSmile){
2)   return (aSmile && bSmile) || (!aSmile && !bSmile)
3)  }
4)  console.log("Is Person In Trouble:"+monkey_trouble(true,true))
5)  console.log("Is Person In Trouble:"+monkey_trouble(true,false))
6)  console.log("Is Person In Trouble:"+monkey_trouble(false,true))
7)  console.log("Is Person In Trouble:"+monkey_trouble(false,false))
```

Output:

```
Is Person In Trouble:true
demo.js:5 Is Person In Trouble:false
demo.js:6 Is Person In Trouble:false
demo.js:7 Is Person In Trouble:true
```

Problem-3: Warmup-2 > string_times

Given a string and a non-negative int n, return a larger string that is n copies of the original string.

string_times('Hi', 2) --> 'HiHi'
string_times('Hi', 3) --> 'HiHiHi'
string_times('Hi', 1) --> 'Hi'

Solution:

```
1)  function string_times(str,n){
2)   result="";
3)   var count=1;
4)   while(count<=n){
5)    result=result+str;
6)    count++;
7)   }
8)   return result;
9)  }
10) console.log(string_times("durga",3))
11) console.log(string_times("hello",2))
```

Output:

```
durgadurgadurga
hellohello
```

Problem-4: Logic-2 > lucky_sum

Given 3 int values, a b c, return their sum. However, if one of the values is 13 then it does not count towards the sum and values to its right do not count. So for example, if b is 13, then both b and c do not count.

lucky_sum(1, 2, 3) --> 6
lucky_sum(1, 2, 13) --> 3
lucky_sum(1, 13, 3) --> 1

Solution:

```
function lucky_sum(a,b,c){
  if(a==13){
    return 0;
  }
  if(b==13){
    return a;
  }
  if(c==13){
    return a+b;
  }
}
console.log(lucky_sum(13,10,5))//0
console.log(lucky_sum(5,13,6))//5
console.log(lucky_sum(7,5,13))//12
```

Problem-5:Logic-1 > caught_speeding

You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.

caught_speeding(60, False) --> 0
caught_speeding(65, False) --> 1
caught_speeding(65, True) --> 0

Solution:

```
function caught_speeding(speed,isBirthday){
  if (isBirthday) {
    speed=speed-5;
  }
  if (speed<=60) {
    return 0;
  }
  else if (speed>=61 && speed<=80) {
```

```
9)      return 1;
10)   }
11)   else{
12)    return 2;
13)   }
14) }
15) console.log("Getting Ticket With Number:"+caught_speeding(60, false))//0
16) console.log("Getting Ticket With Number:"+caught_speeding(65, false))//1
17) console.log("Getting Ticket With Number:"+caught_speeding(65, true))//0
```

JavaScript Arrays:

An array is an indexed collection of elements.

The main advantage of arrays concept is we can represent multiple values by using a single variable so that length of the code will be reduced and readability will be improved.

Without arrays:

```
var n1=10;
var n2=20;
var n3=30;
var n4=40;
```

With arrays:

```
var numbers=[10,20,30,40]
```

Accessing Array Elements by using index:

By using index we can access array elements. Javascript arrays follow 0-based index. i.e The index of first element is 0

Eg:

```
var friends=["durga","sunny","bunny","chinny"];
console.log(friends[0]); //durga
console.log(friends[3]); //chinny
console.log(friends[30]); //undefined
```

Note: If we are trying to access array elements by using out of range index then we will get undefined value and we won't get any error.

Updating array elements by using index:

```
var friends=["durga","sunny","bunny","chinny"];
friends[1]="mallika"
console.log(friends)// ["durga", "mallika", "bunny", "chinny"]
```

Adding new elements to the array by using index:

```
var friends=["durga","sunny","bunny","chinny"];
friends[4]="vinny";
console.log(friends)//["durga","sunny","bunny","chinny","vinny"]
friends[40]="pinny";
console.log(friends)// ["durga", "sunny", "bunny", "chinny", "vinny", empty × 35, "pinny"]
```

Note: By using index we can retrieve,update and add elements of array. But in general we can use index to access array elements.

How to create an empty array:

1st way: var numbers=[];
2nd way: var numbers=new Array();

How to find length of array:

By using length variable

```
var friends=["durga","sunny","bunny","chinny"];
console.log(friends.length)//4
```

Is Javascript array can hold only homogeneous elements?

Javascript array can hold heterogeneous elements also.

Eg:

```
var random_collection=["durga",10000,true,null]
```

Important Methods related to Javascript arrays:

Java script defines several methods which are applicable on arrays.
Being a programmer we can use these methods directly and we are not responsible to implement so that our life will become very easy.

The following are important methods

1. push()
2. pop()

3. unshift()
4. shift()
5. indexOf()
6. slice()

1. push():

We can use push() method to add elements to the end of array. After adding element this method returns length of the array.

Eg:

```
var numbers=[10,20,30,40]
numbers.push(50)
console.log(numbers)// [10, 20, 30, 40, 50]
```

2. pop():

We can use pop() method to remove and return last element of the array

```
var numbers=[10,20,30,40]
console.log(numbers.pop())// 40
console.log(numbers.pop())// 30
console.log(numbers)// [10,20]
```

3. unshift():

We can use unshift() method to add element in the first position. It is counter part of push() method.

```
var numbers=[10,20,30,40]
numbers.unshift(50)
console.log(numbers)//[50, 10, 20, 30, 40]
```

4. shift():

We can use shift() method to remove and return first element of the array. It is counter part to pop() method.

Eg:

```
var numbers=[10,20,30,40]
numbers.shift()
console.log(numbers)//[20, 30, 40]
```

5. indexOf():

We can use indexOf() to find index of specified element.
If the element present multiple times then this method returns index of first occurrence.

If the specified element is not available then we will get -1.

Eg:

```
var numbers=[10,20,10,30,40];
console.log(numbers.indexOf(10))//0
console.log(numbers.indexOf(50))// -1
```

6. slice():

We can use slice operator to get part of the array as slice.
slice(begin,end)===>returns the array of elements from begin index to end-1 index.
slice()==>returns total array.This can be used for cloning purposes.

Eg:

```
var numbers=[10,20,30,40,50,60,70,80]
var num1=numbers.slice(1,5)
console.log(num1)// [20, 30, 40, 50]
num2=numbers.slice()
console.log(num2)// [10, 20, 30, 40, 50, 60, 70, 80]
```

Multi dimensional Arrays:

Sometimes array can contain arrays as elements.i.e array inside array is possible. Such type of arrays are considered as multi dimensional arrays or nested arrays.

Eg:

```
var nums=[[10,20,30],[40,50,60],[70,80,90]]
console.log(nums[0])//[10,20,30]
console.log(nums[0][0])//10
```

Book Management Application:

demo.js:

```
1)  var books=[]
2)  var input=prompt("Which operation You want to perform [add|list|exit]:")
3)  while (input != "exit") {
4)   if (input=="add") {
5)     var newBook= prompt("Enter Name of the Book:")
6)     books.push(newBook);
7)   }
8)   else if (input=="list") {
9)     console.log("List Of Available Books:");
10)    console.log(books);
11)  }
12)  else {
13)    console.log("Enter valid option");
```

```
14)   }
15)   input=prompt("What operation You want to perform [add|list|exit]:")
16) }
17) console.log("Thanks for using our application");
```

Retrieving Elements of Array:

We can retrive elements of array by using the following ways

1. while loop
2. for loop
3. for-of loop
4. forEach method

1. while loop:

```
1)  var nums=[10,20,30,40,50]
2)  var i=0;
3)  while (i<nums.length) {
4)    console.log(nums[i]);
5)    i++;
6)  }
```

2. for loop:

```
1)  var nums=[10,20,30,40,50]
2)  for (var i = 0; i < nums.length; i++) {
3)    console.log(nums[i]);
4)    //alert(nums[i]);
5)  }
```

3. for-of loop:

It is the convinient loop to retrieve elements of array.

Eg:

```
1)  var colors=["red","blue","yellow"]
2)  for (color of colors) {
3)    console.log('******************');
4)    console.log(color);
5)    console.log('******************');
6)  }
```

4. forEach Method:

forEach() is specially designed method to retrieve elements of Array.

Syntax:
arrayobject.forEach(function)

For every element present inside array the specified function will be applied.

Eg 1:

```
1) var heroines=['sunny','mallika','samantha','katrina','kareena']
2) function printElement(element){
3)   console.log('**********************');
4)   console.log(element);
5)   console.log('**********************');
6) }
7) heroines.forEach(printElement)
```

Eg 2:

```
1) var heroines=['sunny','mallika','samantha','katrina','kareena']
2) heroines.forEach(function (element) {
3)   console.log('********************');
4)   console.log(element);
5)   console.log('********************');
6) })
```

Note: The following are also valid

```
heroines=['sunny','mallika','samantha','katrina','kareena']
heroines.forEach(console.log)
heroines.forEach(alert)
```

for loop vs forEach function:

1. for loop is general purpose loop and applicable everywhere. But forEach() function is applicable only for arrays

2. By using for loop, we can move either forward direction or backward direction. But by using forEach() function we can move only forward direction.

Eg: By using for loop we can print array elements either in original order or in reverse order.But by using forEach() function we can print only in original order.

How to delete array elements based on index:

We have to use splice() function.

Syntax:

arrayobject.splice(index,numberofElements)
It deletes specified number of elements starts from the specified index.

Eg:
```
var heroines=['sunny','mallika','samantha','katrina','kareena']
heroines.splice(3,1)
console.log(heroines);//["sunny", "mallika", "samantha", "kareena"]
```

Immutability vs Mutability:

Once we creates an array object,we are allowed to change its content.Hence arrays are Mutable.

Eg:

```
var numbers=[10,20,30,40]
numbers[0]=777
console.log(numbers)//[777,20,30,40]
```

Once we creates string object,we are not allowed to change the content.If we are trying to change with those changes a new object will be created and we cannot change content of existing object. Hence string objects are immutable.

Eg:
```
 var name='Sunny'
 name[0]='B'
 console.log(name)// Sunny
```

Note: Mutability means changeable where as Immutable means Non-Changeable.

Q1. Write a Javascript function to take an array as argument and print its elements in reverse order?

```
1)  function reverse(array){
2)    console.log('Elements in Reverse Order:')
3)    for (var i = array.length-1; i >=0 ; i--) {
4)      console.log(array[i])
5)    }
6)  }
7)  reverse([10,20,30,40,50])
8)  reverse(['A','B','C','D','E'])
```

Output:

Elements in Reverse Order:
50
40
30
20
10
Elements in Reverse Order:
E
D
C
B
A

Q2. Write a Javascript function to check whether the elements of given array are identical(same) or not?

```
1)  function identical(array){
2)  var first=array[0]
3)  for (var i = 1; i < array.length; i++) {
4)    if (array[i] != first) {
5)      return false;
6)    }
7)  }
8)  return true;
9)  }
10) console.log(identical([10,10,10,10]));//true
11) console.log(identical([10,20,30,40]));//false
```

Q3. Write a Javascript function to find maximum value of the given array?

```
1)  function max(array){
2)  var max=array[0]
3)  for (var i = 1; i < array.length; i++) {
4)    if (array[i] > max) {
5)      max=array[i]
6)    }
7)  }
8)  return max
9)  }
10) console.log(max([10,20,30,40]));//40
```

Q4. Write a Javascript function to find the sum of elements present in given array?

```
function sum(array){
var sum=0
for (num of array) {
 sum+=num
}
return sum
}
console.log(sum([10,20,30,40]));//100
```

Book Management Application:

```
var books=[]
var input=prompt("Which operation You want to perform [add|delete|list|exit]:")
while (input != "exit") {
 if (input=="add") {
  addBook();
 }
 else if (input=="list") {
  listBooks()
 }
 else if(input=="delete"){
  deleteBook()
 }
 else {
  console.log("Enter valid option");
 }
 input=prompt("What operation You want to perform [add|delete|list|exit]:")
}
console.log("Thanks for using our application");

function addBook(){
 var newBook= prompt("Enter Name of the Book:")
 books.push(newBook);
}
function listBooks(){
 console.log("List Of Available Books:");
 for (book of books) {
  console.log(book);
 }
}
function deleteBook(){
 var name=prompt("Enter Book Name to delete:")
 var index=books.indexOf(name)
 if(index== -1){
  console.log("Specified book is not available");
```

```
35)   }
36)   else{
37)     books.splice(index,1)
38)     console.log("Specified Book Deleted");
39)   }
40) }
```

JavaScript Objects:

By using arrays we can store a group of individual objects and it is not possible to store key-value pairs.

If we want to represent a group of key-value pairs then we should go for Objects.

Array: A group of individual objects
Object: A group of key-value pairs

JavaScript objects store information in the form of key-value pairs.
These are similar to Java Map objects and Python Dictionary objects.

Syntax:

var variableName={ key1:value1,key2:value2,...};

Eg:

```
1)  var movie={
2)            name:'Bahubali',
3)          year: 2016,
4)          hero:'prabhas'
5)       };
```

In the case of JavaScript objects, no guarentee for the order and hence index conept is not applicable.

How to access values from Object:

We can access values by using keys in the following 2 ways

1. obj["key"]
Here quotes are mandatory
Eg: movie["hero"] ===>valid
movie[hero]====>Uncaught ReferenceError: hero is not defined

2. obj.key
Here we cannot take quotes
Eg: movie.hero

How to create and initialize JavaScript objects:

1. To create empty object
var nums={} or
var nums=new Object()

Once we creates empty object we can add key-value pairs as follows

1st way:

```
nums["fno"]=100
nums["sno"]=200
```

2nd way:

```
nums.fno=100
nums.sno=200
```

How to update values:

```
nums["fno"]=999 or
nums.fno=999
```

Iterating Objects:

To access all key-value pairs we can use for-in loop

Eg:

```
1)  var nums={fno=100,sno=200,tno=300}
2)     for(key in nums){
3)        console.log(key); //To print only keys
4)        console.log(nums[key]); //To print only values
5)        console.log(key+":"+nums[key]); //To print both key and values
6)     }
```

Differences between Arrays and Objects:

Arrays	Object
1) Arrays can be used to represent individual values	1) Objects can be used to represent key-value pairs
2) Order will be maintained in Arrays	2) Order concept not applicable for Objects
3) By using index we can access and update data in arrays	3) By using key we can access and update data in Objects

Nested Objects and Arrays:

Inside Array, we can take objects. Similarly inside Objects we can take array.
Hence nesting of objects and arrays is possible.

Eg 1:

```
1)  var movies=[{name:'Bahubali',year:2016,hero:'Prabhas'},
2)          {name:'Sanju',year:2018,hero:'Ranveer'},
3)          {name:'Spider',year:2017,hero:'Mahesh'}
4)       ]
```

movies[0]["hero"] ===>Prabhas
movies[2]["year"] ===>2017

Eg 2:

```
1)  var numbers={
2)            fg:[10,20,30],
3)            sg:[40,50,60],
4)            tg:[70,80,90]
5)       }
```

numbers.sg[2] ==>60
numbers.tg[1] ==>80

Object Methods:

Java script object can contain methods also.

Eg:

```
1)  var myobj={
2)           A:'Apple',
3)        B:'Banana',
4)        m1:function(){console.log("Object Method");}
5)       }
```

We can invoke this method as follows:

myobj.m1()

this keyword:

Inside object methods,if we want to access object properties then we should use 'this' keyword.

```
1)  var movie={
2)              name:'Bahubali',
3)          year: 2016,
4)          hero:'prabhas',
5)          getInfo:function(){
6)          console.log('Movie Name:'+this.name);
7)          console.log('Released Year:'+this.year);
8)          console.log('Hero Name:'+this.hero);
9)          }
10)     };
11)
12) movie.getInfo()
```

Output:

Movie Name:Bahubali
Released Year:2016
Hero Name:prabhas

It is possible to refer already existing function as object method.

Eg 1:

```
1)  function demo(){
2)    console.log('Demo Function');
3)    }
4)
5)  var movie={
6)              name:'Bahubali',
7)          year:2016,
8)          hero:'Prabhas',
9)          getInfo:demo
10)       };
11) movie.getInfo()
```

Output:

Demo Function

Eg 2:

```
1)  function demo(){
2)    console.log('Demo Function:'+this.name);
3)    }
4)
5)  var movie={
```

```
6)          name:'Bahubali',
7)        year:2016,
8)        hero:'Prabhas',
9)        getInfo:demo
10)      };
11)  movie.getInfo()
```

Output:

Demo Function:Bahubali

If we call demo() function direclty then output is:

Demo Function

We can use named functions also.

```
1)  var movie={
2)          name:'Bahubali',
3)        year:2016,
4)        hero:'Prabhas',
5)        getInfo: function demo(){
6)     console.log('Demo Function:'+this.name);
7)     }
8)       };
```

Even we are not required to use function keyword also for object methods inside object and we can declare function directly without key.[But outside of object compulsory we should use function keyword to define functions]

```
1)  var movie =
2)  {
3)     name:"Rockstar",
4)     hero:"Ranbeer Kapoor",
5)     year:"2012",
6)     myFunction(){
7)        console.log("kjdf");
8)     }
9)  }
```

Even we can pass parameters also

```
1)  var movie =
2)  {
3)     name:"Rockstar",
4)     hero:"Ranbeer Kapoor",
5)     year:"2012",
6)     myFunction(a){
7)        console.log("kjdf:"+a);
8)     }
9)  }
```

```
10)
11) var a =10;
12) movie.myFunction(a)
```

Mini Application:

```
1)  var movies=[{name:'Bahubali',isWatched:'true',isHit:'true'},
2)          {name:'Sanju',isWatched:'false',isHit:'true'},
3)          {name:'Spider',isWatched:'true',isHit:'false'},
4)       ]
5)
6)  movies.forEach(function(movie){
7)   var result=""
8)   if(movie.isWatched=="true"){
9)   result=result+"I Watched "
10)  }
11)  else{
12)  result=result+"I have not seen "
13)  }
14)  result=result+movie.name
15)  if(movie.isHit=="true"){
16)  result=result+" and Movie is Hit!!!"
17)  }
18)  else{
19)  result=result+" and Movie is Flop!!!"
20)  }
21)  console.log(result)
22)  });
```

Output:

I Watched Bahubali and Movie is Hit!!!
I have not seen Sanju and Movie is Hit!!!
I Watched Spider and Movie is Flop!!!