Javascript

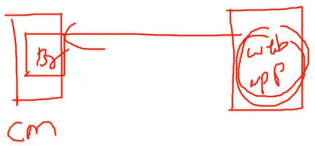
Part 1- Introduction to JavaScript

Agenda

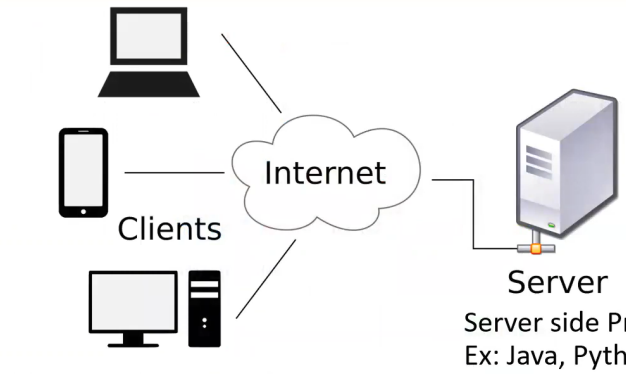
- Introduction to Web Applications
- Client & Web servers
- HTML, CSS & JavaScript
- What is JavaScript? JavaScript History
- Java vs JavaScript

Web Applications

- A web application or web app is a client–server computer program that the client runs in a web browser.
- Common web applications include webmail, online retail sales, online banking etc.



Client & Web Server



Client side Programming
Ex: HTML,CSS, JavaScript etc.

Server
Server side Programming
Ex: Java, Python, C# etc.

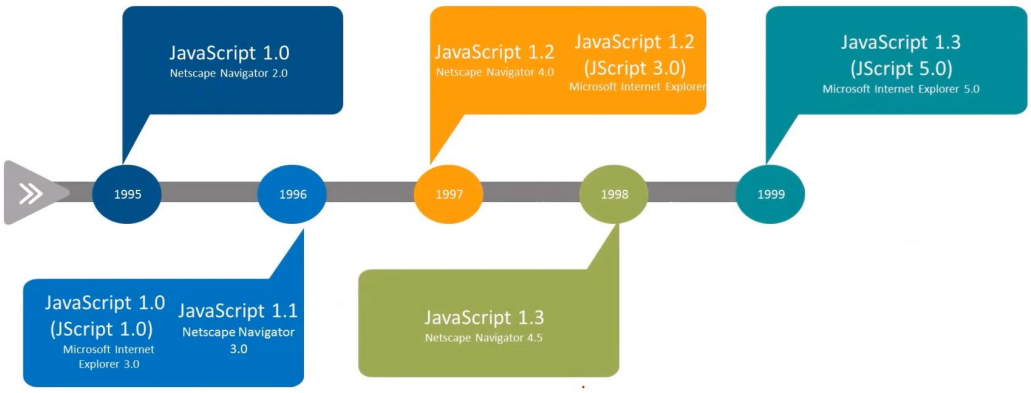
HTML, CSS & JavaScript

- Hyper Text Markup Language(HTML) is the skeleton of the web. It is used for displaying the web.
- CSS is like our clothes. We put on fashionable clothes to look better. Similarly, the web is quite stylish as well. It uses CSS which stands for Cascading Style Sheets for styling purpose.
- JavaScript which puts life into a web page. It will add dynamic nature to the web.

What is JavaScript?

- JavaScript is a high level, interpreted, programming language used to make web pages more interactive.
- It is interpreted and executed on the client machine.
- Used as default scripting language within HTML.
- Reduces the load on server as some operation.
- It is a very powerful client-side scripting language which makes your webpage more lively and interactive.
- JavaScript is Case-sensitive language.
- It will add some dynamic nature to the webpages.

JavaScript History



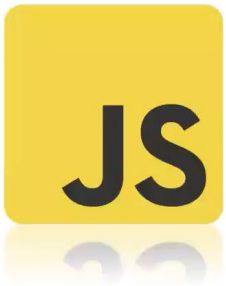
Java Vs JavaScript

Java	JavaScript
▪ It is an OOP programming language	▪ It is an OOP scripting language
▪ Runs on a virtual machine or browser	▪ Runs on a browser only
▪ Code is compiled before execution	▪ Code is interpreted/Just In Time(JIT) compiled before execution
▪ Static type checking	▪ Dynamic type checking

Part 2- What is DOM (Document Object Model)? How to write a JavaScript program ?

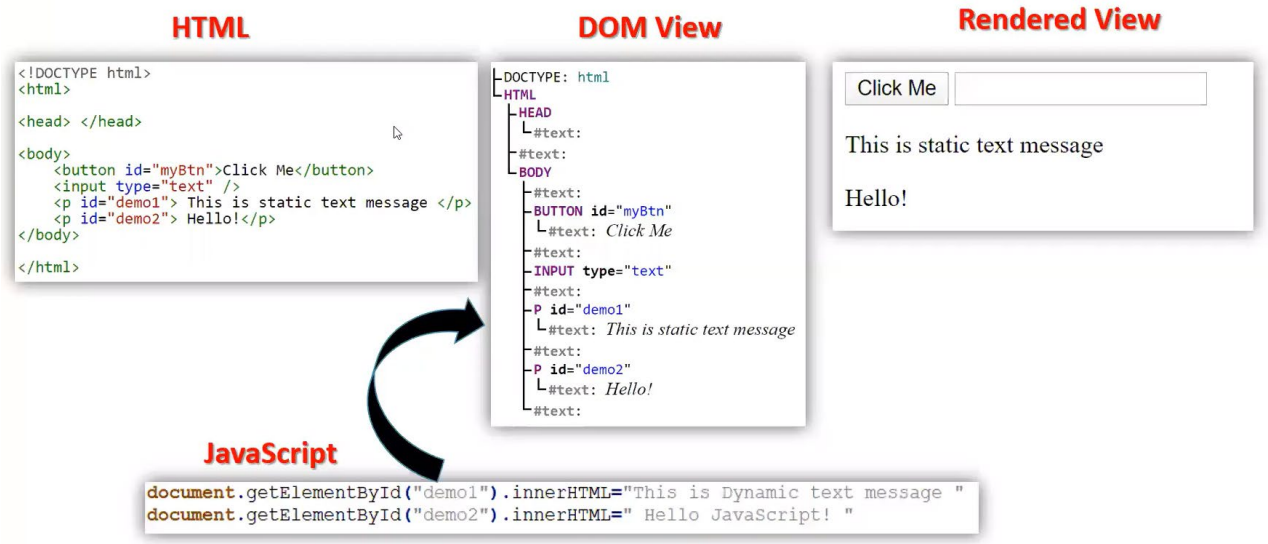
Agenda

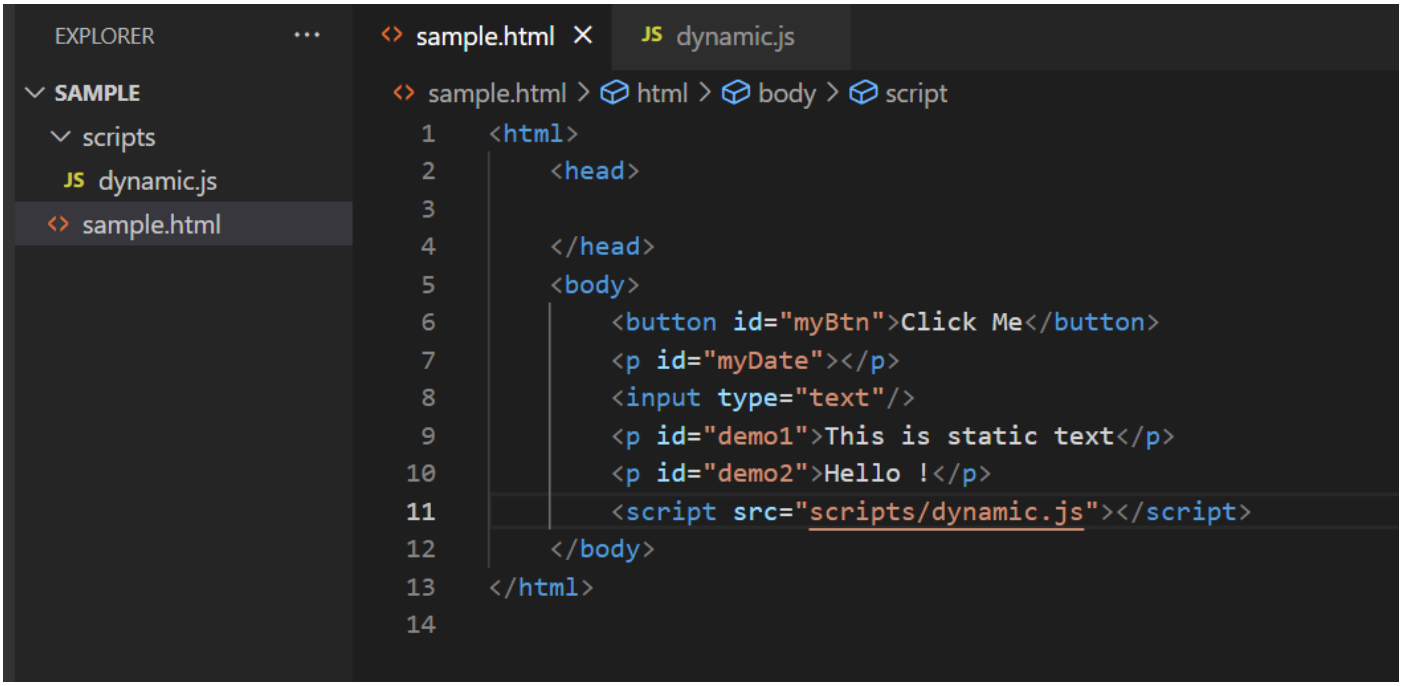
- What is DOM?
- How to write JavaScript Program



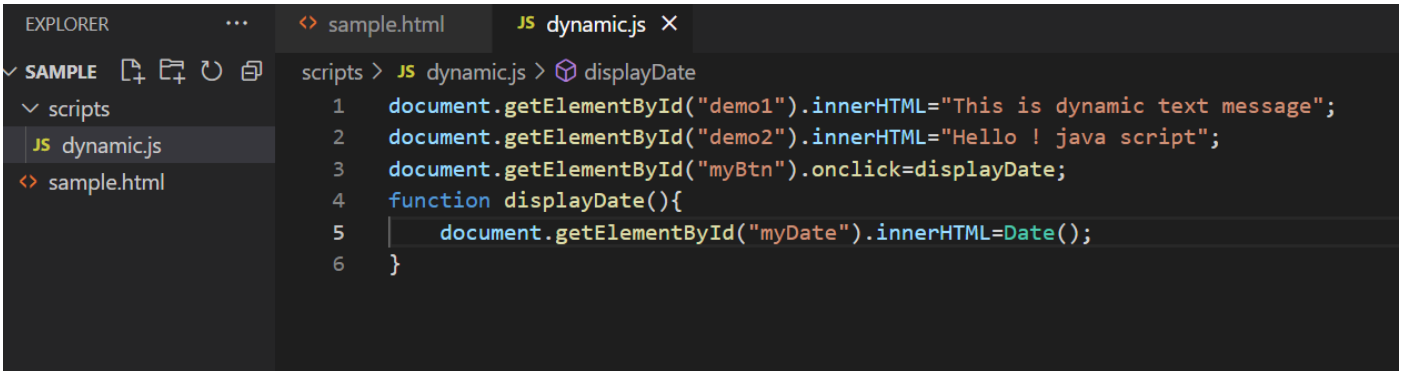
What is DOM?

- DOM is an API Interface provided by browser.
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- With the DOM, JavaScript can access and change all the elements of an HTML document.


HTML
DOM View
Rendered View
<!DOCTYPE html>
<html>
<head> </head>
<body>
<button id="myBtn">Click Me</button>
<input type="text" />
 This is static text message
 Hello!
</body>
</html>
DOCTYPE: html
HTML
HEAD
#text:
#text:
BODY
#text:
BUTTON id="myBtn"
#text: Click Me
#text:
INPUT type="text"
#text:
P id="demo1"
#text: This is static text message
#text:
P id="demo2"
#text: Hello!
#text:
Click Me
This is static text message
Hello!
JavaScript
document.getElementById("demo1").innerHTML="This is Dynamic text message "
document.getElementById("demo2").innerHTML=" Hello JavaScript! "


EXPLORER
SAMPLE
scripts
dynamic.js
sample.html
sample.html
dynamic.js
sample.html > html > body > script
<html>
<head>
</head>
<body>
<button id="myBtn">Click Me</button>

<input type="text"/>
This is static text
Hello !
<script src="scripts/dynamic.js"></script>
</body>
</html>

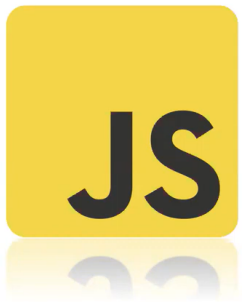

EXPLORER
SAMPLE
scripts
dynamic.js
sample.html
sample.html
dynamic.js
scripts > dynamic.js > displayDate
document.getElementById("demo1").innerHTML="This is dynamic text message";
document.getElementById("demo2").innerHTML="Hello ! java script";
document.getElementById("myBtn").onclick=displayDate;
function displayDate(){
document.getElementById("myDate").innerHTML=Date();
}

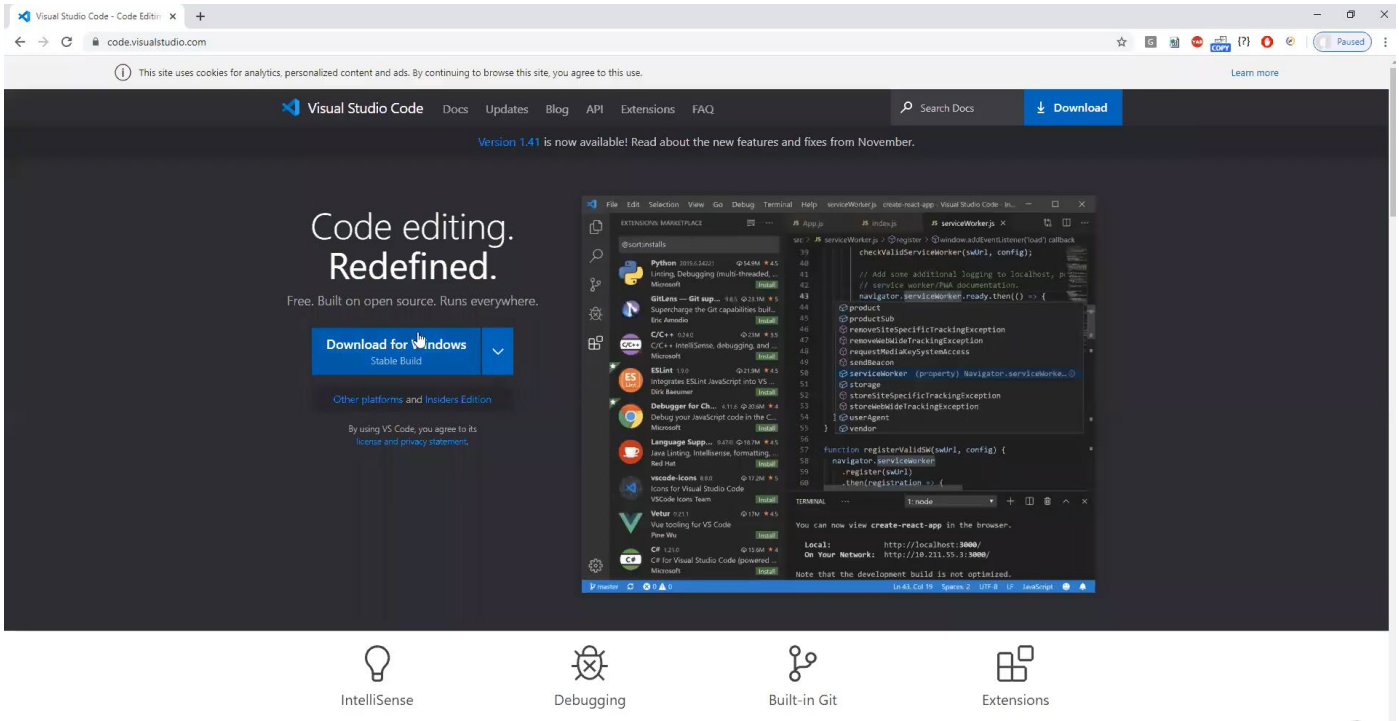
- **With the object model, JavaScript gets all the power it needs to create dynamic HTML:**
  - JavaScript can change all the HTML elements & attributes in the page
  - JavaScript can change all the CSS styles in the page
  - JavaScript can remove existing HTML elements and attributes
  - JavaScript can add new HTML elements and attributes
  - JavaScript can react to all existing HTML events in the page
  - JavaScript can create new HTML events in the page

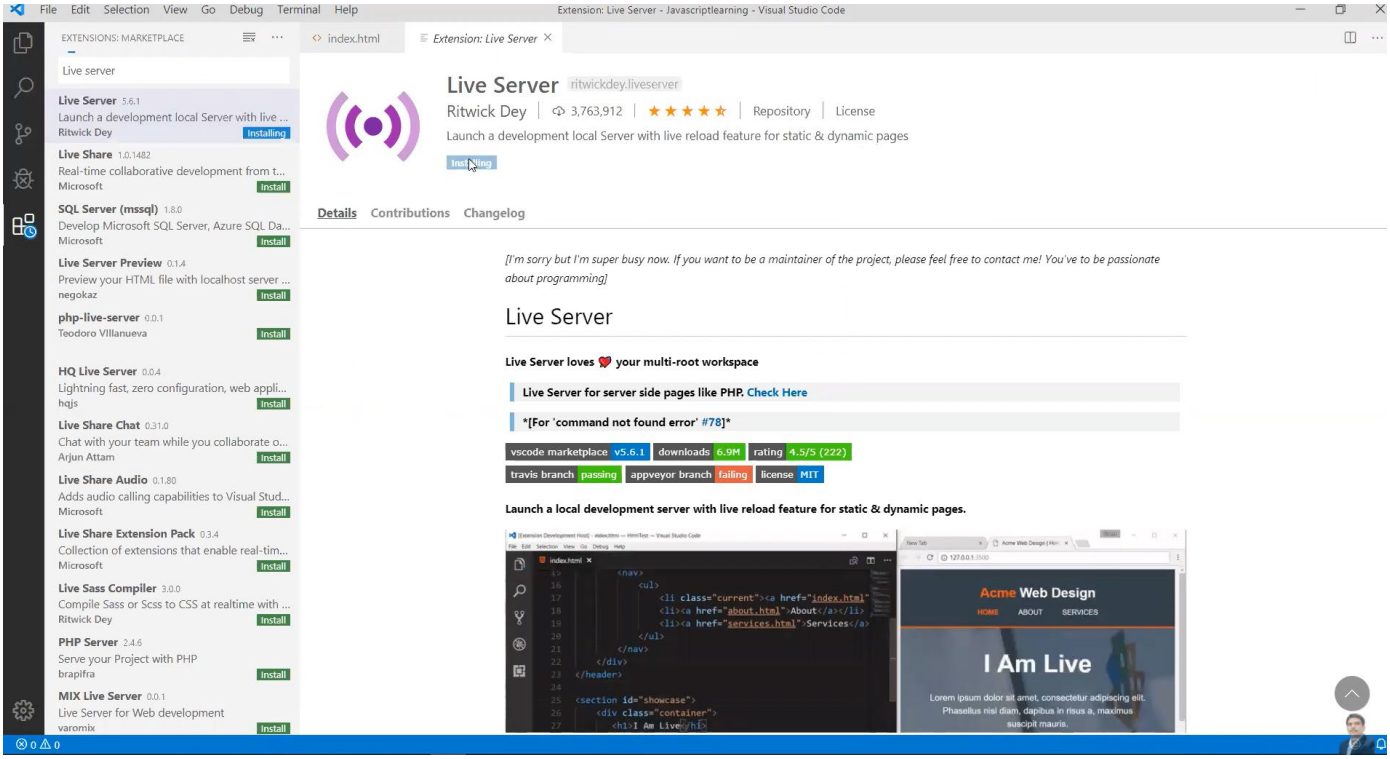
Part 3- Visual Studio Code| Live server Extension | JavaScript Comments & Statements

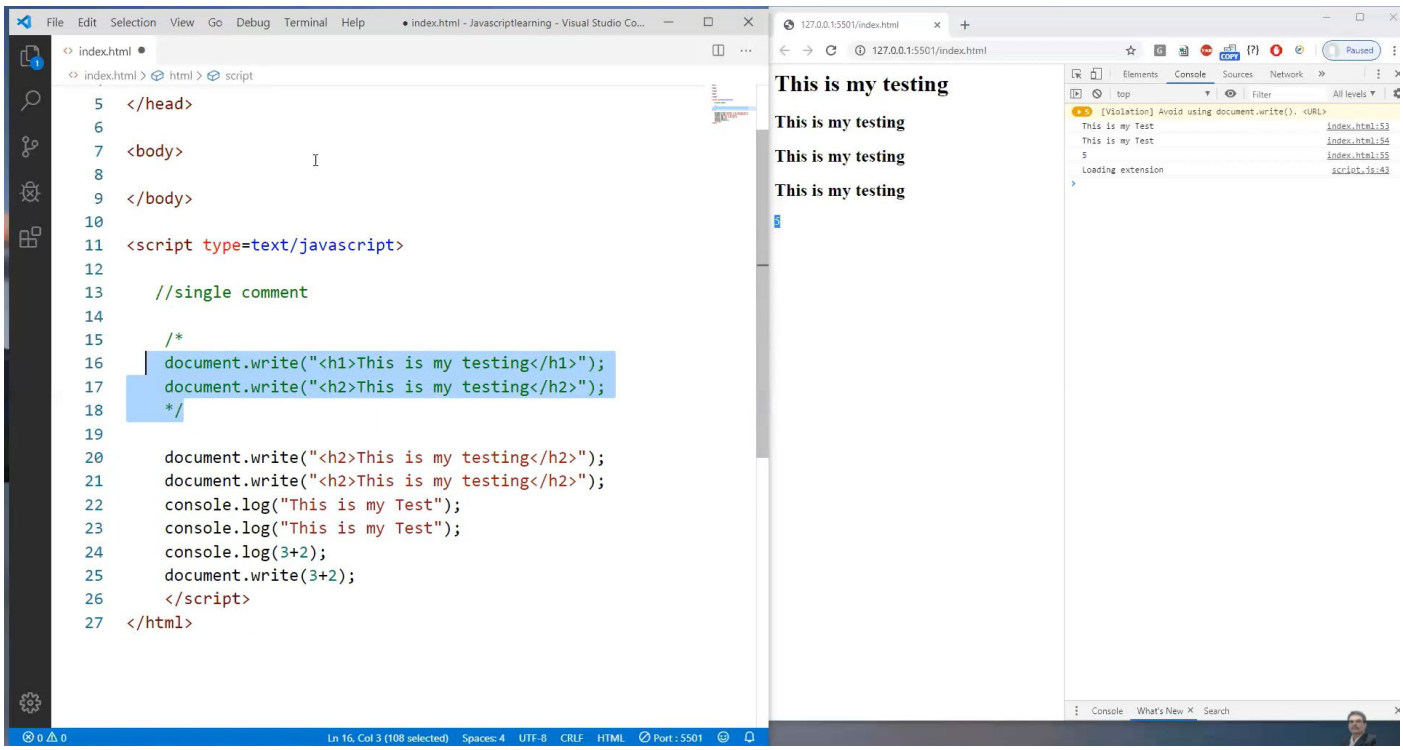
Agenda

- Visual Studio Code
- Live Server Extension
- document & console objects in JS
- Comments & Statements in JS





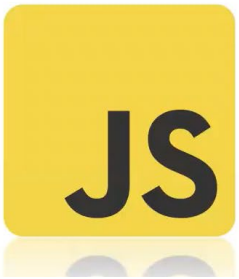

File Edit Selection View Go Debug Terminal Help
Extension: Live Server - Javascriptlearning - Visual Studio Code
EXTENSIONS: MARKETPLACE
Live server
Live Server 5.6.1
Launch a development local Server with live ...
Ritwick Dey
Installing
Live Share 1.0.1482
Real-time collaborative development from t...
Microsoft
Install
SQL Server (mssql) 1.8.0
Develop Microsoft SQL Server, Azure SQL Da...
Microsoft
Install
Live Server Preview 0.1.4
Preview your HTML file with localhost server ...
negokaz
Install
php-live-server 0.0.1
Teodoro Villanueva
Install
HQ Live Server 0.0.4
Lightning fast, zero configuration, web appli...
hqjs
Install
Live Share Chat 0.3.1.0
Chat with your team while you collaborate o...
Arjun Attam
Install
Live Share Audio 0.1.80
Adds audio calling capabilities to Visual Stud...
Microsoft
Install
Live Share Extension Pack 0.3.4
Collection of extensions that enable real-tim...
Microsoft
Install
Live Sass Compiler 3.0.0
Compile Sass or Scss to CSS at realtime with ...
Ritwick Dey
Install
PHP Server 2.4.6
Serve your Project with PHP
brapifra
Install
MIX Live Server 0.0.1
Live Server for Web development
yaromix
Install
index.html
Extension: Live Server
Live Server
ritwickdey.liveserver
Ritwick Dey | 3,763,912 | ★★★★★ | Repository | License
Launch a development local Server with live reload feature for static & dynamic pages
Installing
Details Contributions Changelog
[I'm sorry but I'm super busy now. If you want to be a maintainer of the project, please feel free to contact me! You've to be passionate about programming]
Live Server
Live Server loves your multi-root workspace
Live Server for server side pages like PHP. Check Here
[For 'command not found error' #78]
vscode marketplace v5.6.1 downloads 6.9M rating 4.5/5 (222)
travis branch passing appveyor branch failing license MIT
Launch a local development server with live reload feature for static & dynamic pages.
Acme Web Design
HOME ABOUT SERVICES
I Am Live
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Phasellus nisi diam, dapibus in risus a, maximus
suscipit mauris.


File Edit Selection View Go Debug Terminal Help
index.html - Javascriptlearning - Visual Studio Co...
index.html
index.html > html > script
5 </head>
6
7 <body>
8
9 </body>
10
11 <script type=text/javascript>
12
13 //single comment
14
15 /*
16 document.write("<h1>This is my testing</h1>");
17 document.write("<h2>This is my testing</h2>");
18 */
19
20 document.write("<h2>This is my testing</h2>");
21 document.write("<h2>This is my testing</h2>");
22 console.log("This is my Test");
23 console.log("This is my Test");
24 console.log(3+2);
25 document.write(3+2);
26 </script>
27 </html>
Ln 16, Col 3 (108 selected) Spaces: 4 UTF-8 CRLF HTML Port : 5501
127.0.0.1:5501/index.html
This is my testing
This is my testing
This is my testing
This is my testing
This is my testing
This is my Test
This is my Test
5
Loading extension
index.html:53
index.html:54
index.html:55
script.js:43

Part 4- JavaScript Variables & Data Types

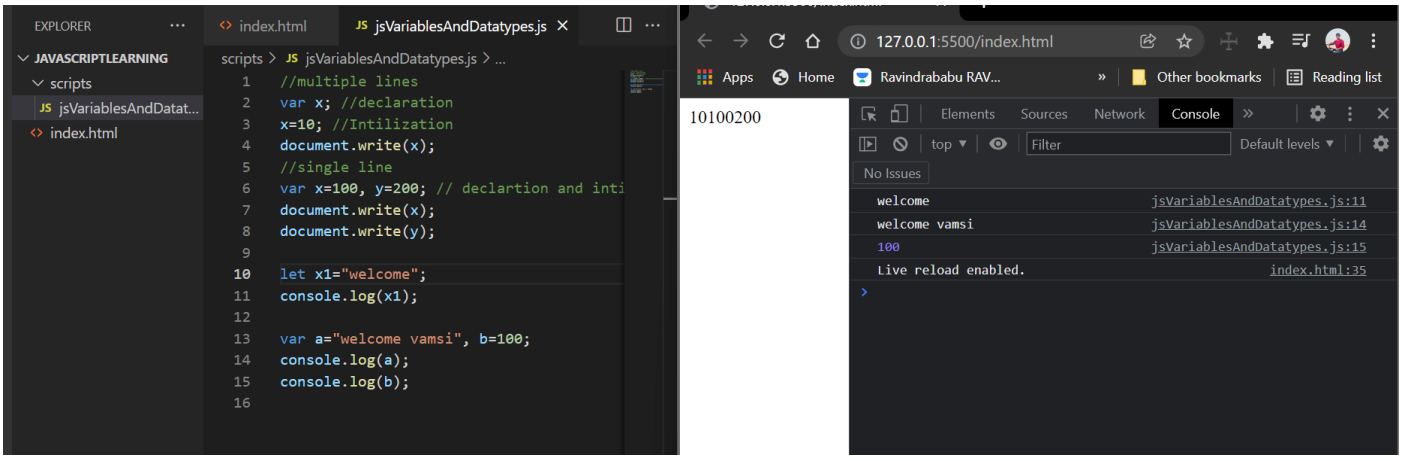
Agenda

- Variables
- Data Types



Variables

- Variables are containers for storing data values.
- We can create variables using **let**, **var** & **const** keywords



Difference between var and let in JavaScript

- var and let are both used for variable declaration in javascript.
- But the difference between them is that **var** is function scoped and **let** is block scoped.

Undefined

```
Input:
console.log(x);
var x=5;
console.log(x);
Output:
undefined
5
```

Error

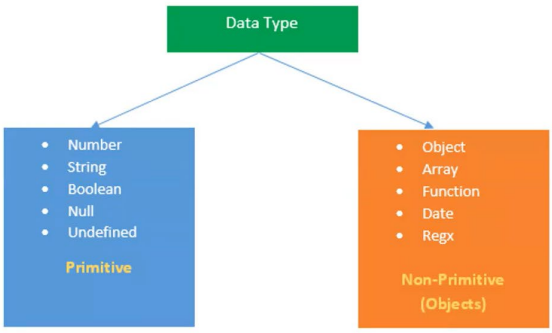
```
Input:
console.log(x);
let x=5;
console.log(x);
Output:
Error
```

const keyword

```
*/
const z=200;
console.log(z);
z=201;
console.log(z);
```

```
200                                          jsVariablesAndDatatypes.js:27
▶ Uncaught TypeError: Assignment to          jsVariablesAndDatatypes.js:28
constant variable.
    at jsVariablesAndDatatypes.js:28:2
>
```

Data Types in JavaScript




```
/* Data Types */
var price=10.5;
console.log(price);
console.log("price is:"+typeof(price));
var name="vamsi";
console.log(name);
console.log(typeof(name));
var flag=true;
console.log(flag);
console.log(typeof(flag));
var status=null;
console.log(status);
console.log(typeof(status));
var nothing;
console.log(nothing);
console.log(typeof(nothing));
```

```
No Issues
10.5                                         jsVariablesAndDatatypes.js:34
price is:number                              jsVariablesAndDatatypes.js:35
vamsi                                        jsVariablesAndDatatypes.js:37
string                                       jsVariablesAndDatatypes.js:38
true                                         jsVariablesAndDatatypes.js:40
boolean                                      jsVariablesAndDatatypes.js:41
null                                         jsVariablesAndDatatypes.js:43
string                                       jsVariablesAndDatatypes.js:44
undefined                                    jsVariablesAndDatatypes.js:46
undefined                                    jsVariablesAndDatatypes.js:47
>
```

Part 5- JavaScript Operators

Agenda

- Operators in JavaScript

What is an Operator?

- Operator is a symbol which will perform certain operation.
- For example, if we take a simple expression,

4 + 5=9.

Here 4 and 5 are called operands and '+' is called the operator.

JavaScript consists of different types of operators that are used to perform different operations.

Types of JavaScript Operators

- Arithmetic Operators
- Assignment Operators
- Relational/Comparison Operators
- Logical Operators

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Assignment Operators

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

```
//Arithmetic Operators +, -, *, /, %, **, ++, --
var x=10, y=20;
console.log(x+y);
console.log(x-y);
console.log(x*y);
console.log(x/y);
console.log(x%y);
console.log(5**2);
console.log(1/10);
console.log(1%10);

x=x+1;
x++; //Post Increment
console.log(x);
--x; //Pre Increment
console.log(x);

--x;
console.log(x);
x--;
console.log(x);
```

```
30
-10
200
0.5
10
25
0.1
1
12
11
10
9
```

scripts > JS Operators.js

```
22  */
23
24  //Assignment Operators
25  // +=, -=, *=, /=, %=, =
26  x=100;
27  y=200;
28  x+=y;
29  console.log(x);
30  x-=y;
31  console.log(x);
32  x*=y;
33  console.log(x);
34  x/=y;
35  console.log(x);
36  x%=y;
37  console.log(x);
38
```

```
300
100
20000
100
100
```

```
//Comparison/Relational operators
//returns boolean values true/false
// ==, !=, >, >=, <, <=, ===, !==
x=10;
y=20;
console.log(x==y);
console.log(x!=y);
console.log(x>y);
console.log(x>=y);
console.log(x<y);
console.log(x<=y);
console.log(x===y);
console.log(x!==y);

//Ternary Operator
// ?
var result=x>y?x:y;
console.log(result);
```

```
false
true
false
false
true
true
false
true
20
```

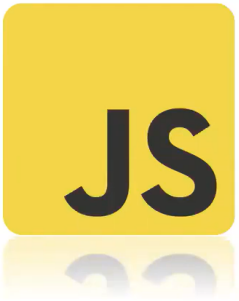
```
//Logical Operators
// &&, ||, !
let x1=false, y1=false;
console.log(x1 && y1);
console.log(x1 || y1);
console.log("------------");
x1=false, y1=true;
console.log(x1 && y1);
console.log(x1 || y1);
console.log("------------");
x1=true, y1=false;
console.log(x1 && y1);
console.log(x1 || y1);
console.log("------------");
x1=true, y1=true;
console.log(x1 && y1);
console.log(x1 || y1);
console.log("------------");
console.log(!x1);
```

```
false
false
------------
false
true
------------
false
true
------------
true
true
------------
false
```

Part 6- Conditional Statements in JavaScript | If | If else | switch case

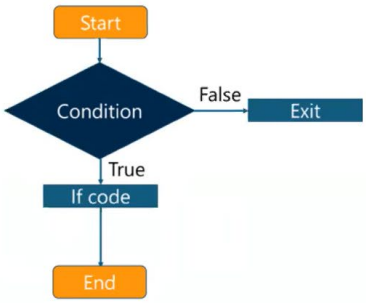
Agenda

- Control Statements
  - Conditional statements
  - Looping statements
  - Jumping statements

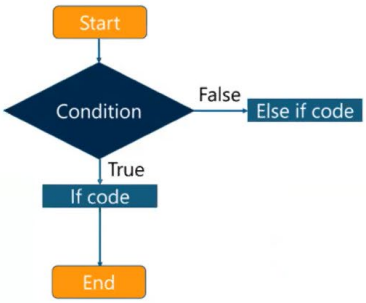


Conditional statements

- if
- else if
- Switch case



if

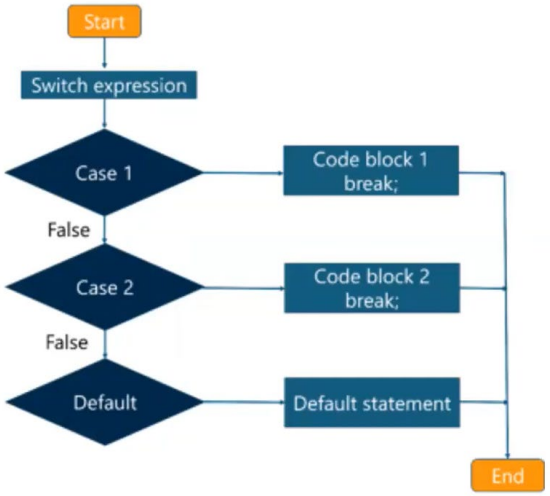


If..else

```
let a=10, b=20, c=30;
if( a> b && a > c)
    console.log("a is larger");
else if(b > a && b > c)
    console.log("b is larger");
else
    console.log("c is larger");

let person_age=18;
if(person_age>=18)
    console.log("Eligible for vote");
else
    console.log("Not eligible for vote");
console.log("Program Exited")

//Even or not
let num=4;
if(num%2==0)
    console.log("Even");
else
    console.log("Odd");
```

```
c is larger
Eligible for vote
Program Exited
Even
```

Switch case



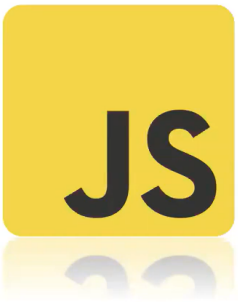

```
let char='a';
switch(char){
    case "a":console.log("a");break;
    case "b":console.log("b");break;
    default:console.log("Fuck You !!!");
}

let day="holiday";
switch(day){
    case "notholiday":console.log("Not holiday");break;
    case "holiday":console.log("holiday");break;
    default:console.log("Fuck You !!!");
}

//Print Week Numbers
let weekNo=8;
switch(weekNo){
    case 1:console.log("Sunday");break;
    case 2:console.log("Monday");break;
    case 3:console.log("Tuesday");break;
    case 4:console.log("Wednesday");break;
    case 5:console.log("Thursday");break;
    case 6:console.log("Friday");break;
    case 7:console.log("Saturday");break;
    default:console.log("Invalid Number");
}
```

```
a
holiday
Invalid Number
```

Part 7- Loops & Jumping statements in JavaScript | while | do while | for

Agenda

- Control Statements
  1. Conditional statements
  2. Looping statements
  3. Jumping statements



Loops

1. while loop
2. do while loop
3. for loop
4. for in loop
5. for of loop

```
for(initilization;condition;inc/dec)
{
statements;
}
```

```
//Print numbers in descending order
let i=10;
while(i>=1){
    console.log(i--);
}

//Print even numbers between 1 and 10
let i=1;
while(i<=10){
    if(i%2==0)
        console.log(i);
    i++;
}

//1...10
let i=1;
while(i<=10){
    console.log(i++);
}
```

```
let i=10;
do{
    console.log(i--);
}while(i>0);

let i=1;
do{
    console.log(i++);
}while(i<=10);
```

```
for(let i=2;i<=10;i++){
    if(i%2==0)
        console.log(i);
}
for(let i=1;i<=10;i++)
    console.log(i);
```

Jumping statements

1. Break
2. continue

```
//break, continue
for(let i=1;i<=5;i++){
    if(i==1 || i==3 || i==5)
        continue;
    console.log(i);
}
console.log("-----------------");
for(let i=1;i<=5;i++){
    if(i==5)
        continue;
    console.log(i);
}
console.log("-----------------");
for(let i=1;i<=5;i++){
    if(i==5)
        break;
    console.log(i);
}
```

```
2
4
-----------------
1
2
3
4
-----------------
1
2
3
4
```

Part 8- Functions in JavaScript

Agenda

- Java Script Functions

JavaScript Function

- A JavaScript function is a **block of statements** which will perform a **specific task**.
- A JavaScript function is executed when "something" invokes it (calls it).

```
function name(parameter1, parameter2, parameter3)
{
  // statements to be executed
}
```

Why Functions?

- You can reuse code: Define the code once, and use it many times.
- You can use the same code many times with different arguments, to produce different results.

Function Invocation

- The code inside the function will execute when "something" **invokes** (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

```
//Example 4
function greetings(){
    return("Hello Sucker !!!");
}
console.log(greetings());

//Examplle 3
function greetings(){
    console.log("Hello Fucker !!!");
}
greetings();

//Example 2
function add(a, b){
    return a+b;
}
let result=add(1, 2);
console.log(result);

//Example1
function add(a, b){
    console.log(a+b);
}
add(100, 200);
```

Example:

C:\Users\vamsi\OneDrive\Desktop\Learning\SKILL_SET_2022\Javascript\js_youtube\JavascriptLearning\calculator.html

JavaScript Objects

Agenda

- JavaScript Objects , Properties & Methods
- Accessing Object Properties
- Accessing Object Methods
- for/in loop with object

JavaScript Objects , Properties & Methods

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

- All cars have the same **properties**, but the property **values** differ from car to car.
- All cars have the same **methods**, but the methods are performed **at different times**.

```
let person={
    firstName:"vamsi",
    lastName:"pallapolu",
    height:160,
    weight:60
};

//Accessing the objects
console.log(person["firstName"]);
console.log(person.lastName);

//Adding property to the existing object
person["color"]="blonde";
console.log(person.color);

//Updating existing property value
person.weight=65;
console.log(person["weight"]);

//Deleting existing property
delete person["weight"];
console.log(person.weight); //undefined
```

```
vamsi
pallapolu
blonde
65
undefined
```

```
//for in loop
for(let x in person){
    console.log(x+":"+person.x);
}
for(let x in person){
    console.log(x+":"+person[x]);
}
```

```
firstName:undefined
lastName:undefined
height:undefined
color:undefined
firstName:vamsi
lastName:pallapolu
height:160
color:blonde
```

```
let employee={
    empid:4046,
    empname:"Vamsi Krishna Pallapolu",
    job:"Software Engineer",
    salary:100000,
    bonus:function(){
        return((this.salary*10)/100);
    }
};

console.log(employee.bonus());
```

```
10000
```

Part 10- Arrays in JavaScript

JavaScript Arrays

- JavaScript arrays are used to store multiple values in a single variable.
- An array is a special variable, which can hold more than one value at a time.
- Let cars = ["Saab", "Volvo", "BMW"];

```
// Variables of same type in same array
let cars=["bmw","honda","suzuki"];
console.log(cars);
let bikes=new Array("duke","pulzar","ducati");
console.log(bikes);
// Variables of different types in same array
let data=["vamsi", 'a', 1, true, null];
console.log(data);

// Array of objects
let person1={
    name:"vamsi",
    age:24
};
let person2={
    name:"krishna",
    age:25
};

let objectArray=[person1,person2];
console.log(objectArray);
console.log(objectArray[1]);

//Looping elements from array
let fruits=["Apple", "Banana", "Mango", "Orange"];
for(let i=0;i<fruits.length;i++){
    console.log(fruits[i]);
}
```

```
[ 'bmw', 'honda', 'suzuki' ]
[ 'duke', 'pulzar', 'ducati' ]
[ 'vamsi', 'a', 1, true, null ]
[ { name: 'vamsi', age: 24 }, { name: 'krishna', age: 25 } ]
{ name: 'krishna', age: 25 }
Apple
Banana
Mango
Orange
```

```
//Looping elements using "for of" loop
for(let fruit of fruits)
    console.log(fruit);

// Looping elements using "for/in" loop
for(let x in fruits)
    console.log(fruits[x]);

//Recognize an array
console.log(typeof fruits);

console.log(Array.isArray(fruits));
```

```
Apple
Banana
Mango
Orange
Apple
Banana
Mango
Orange
object
true
```

Part 11- Array Methods in JavaScript

JavaScript Array Methods

```
let fruits=["Apple", "Banana", "Mango", "Orange"];

// toString(), join()
console.log(fruits.toString());
console.log(fruits.join());
console.log(fruits.join('-'));

// pop() - It removes the last element from the array and prints it in console
console.log(fruits.pop());
console.log(fruits);

// push() - It adds a new element at the end of the array and returns the length of the updated array
let len=fruits.push("carrot");
console.log(len);
console.log(fruits);

// shift() - It removes and returns the first element and shifts the remaining elements into the lower index
fruits=["Apple", "Banana", "Mango", "Orange"];
console.log(fruits.shift());
console.log(fruits);

// unshift() - It adds new element in first postion and shifts the remaining elements into higher index
fruits=["Apple", "Banana", "Mango", "Orange"];
len=fruits.unshift("carrot");
console.log(len);
console.log(fruits);

// Deleting elements from an array
delete fruits[3];
console.log(fruits);
```

```
Apple,Banana,Mango,Orange
Apple,Banana,Mango,Orange
Apple-Banana-Mango-Orange
Orange
[ 'Apple', 'Banana', 'Mango' ]
4
[ 'Apple', 'Banana', 'Mango', 'carrot' ]
Apple
[ 'Banana', 'Mango', 'Orange' ]
5
[ 'carrot', 'Apple', 'Banana', 'Mango', 'Orange' ]
[ 'carrot', 'Apple', 'Banana', <1 empty item>, 'Orange' ]
```

```
// concat() - To join two arrays
let arr1=[1,2,3];
let arr2=["A", "B", "C"];
console.log(arr1.concat(arr2));

let arr3=['a', 'b'];
let res=arr1.concat(arr2, arr3);
console.log(res);

// slice() - slices the array from the index position and returns a new array
let arr=[1,2,3,4,5];
let res2=arr.slice(1);
console.log(res2);
console.log(arr);

// sort() - sorts the existing array and also returns a new array
arr3=[3,6,9,4,1];
res=arr3.sort();
console.log("res:"+res);
console.log(arr3);

// reverse() - reverse the existing array and also returns a new array
arr3=[3,6,9,4,1];
res=arr3.reverse();
console.log("res:"+res);
console.log(arr3);
```

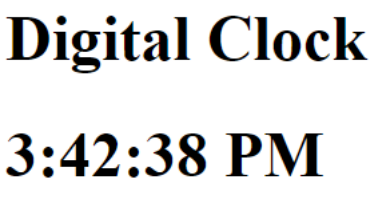
```
[ 1, 2, 3, 'A', 'B', 'C' ]
[
  1,   2,   3,   'A',
  'B', 'C', 'a', 'b'
]
[ 2, 3, 4, 5 ]
[ 1, 2, 3, 4, 5 ]
res:1,3,4,6,9
[ 1, 3, 4, 6, 9 ]
res:1,4,9,6,3
[ 1, 4, 9, 6, 3 ]
```

Part 12- Date Constructor in JavaScript | How To Create Digital Clock using JS

```
let d=new Date();
console.log(d);

// 1-31, 1-12
console.log(d.getDate()+":"+d.getMonth()+":"+d.getFullYear());

// 0-23, 0-59, 0-59
console.log(d.getHours()+":"+d.getMinutes()+":"+d.getSeconds());
```

```
2022-02-16T10:13:07.474Z
16:1:2022
15:43:7
```

```html
<html>
    <body>
        <h1>Digital Clock</h1>
        <h1 id="cl"></h1>
        <script>
            function showTime(){
                let date=new Date();
                let h=date.getHours();
                let m=date.getMinutes();
                let s=date.getSeconds();

                let session="AM";
                h=h<10?"0"+h:h;
                m=m<10?"0"+m:m;
                s=s<10?"0"+s:s;

                if(h>12){
                    h=h-12;
                    session=" PM";
                }

                let time=h+":"+m+":"+s+session;
                document.getElementById("cl").innerHTML=time;
                setTimeout(showTime, 1000);
            }
            showTime();
        </script>
    </body>

</html>
```



Part 13- Strings & Numbers in JavaScript

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Strings can be created as primitives, from string literals, or as objects, using the `String()` constructor:

String literals can be specified using single or double quotes, which are treated identically

JavaScript automatically converts primitives to `String` objects, so that it's possible to use `String` object methods for primitive strings.

In contexts where a method is to be invoked on a primitive string or a property lookup occurs, JavaScript will automatically wrap the string primitive and call the method or perform the property lookup.

```javascript
let s= "welcome";
s=new String("Welcome");

// charAt()
console.log(s.charAt(1));
// indexOf()
console.log(s.indexOf('e'));
console.log(s.lastIndexOf('e'));

// toLower() & toUpper()
console.log(s.toUpperCase());
console.log(s.toLowerCase());
```

```
e
1
6
WELCOME
welcome
```

```
// Following methods returns new Strings
console.log("Following methods returns new Strings");
// concat()
s="welcome";
e=" to java";
console.log(s.concat(e));
// substring()
console.log(s.substring(0));
console.log(s.substring(0, 3));
// replace()
console.log(s.replace('e','r'));
console.log(s.replaceAll('e','r'));
// slice()
s="pallapolu";
console.log("slice:"+s.slice(1));
console.log("slice:"+s.slice(0, 3));
// split()
s=" vamsi krishna pallapolu ";
let words=s.split(" ");
for(let word in words)
    console.log(words[word]);
for( let word of words)
    console.log(word);
// trim()
console.log(s);
let n=s.trim();
console.log(n);
console.log(n.length);
console.log(s.trimStart());
console.log(s.trimEnd());

num="100";
console.log(num.valueOf());
console.log(parseInt(num));
```

```
Following methods returns new Strings
welcome to java
welcome
wel
wrlcome
wrlcomr
slice:allapolu
slice:pal

vamsi
krishna
pallapolu


vamsi
krishna
pallapolu

 vamsi krishna pallapolu
vamsi krishna pallapolu
23
vamsi krishna pallapolu
 vamsi krishna pallapolu
100
100
```

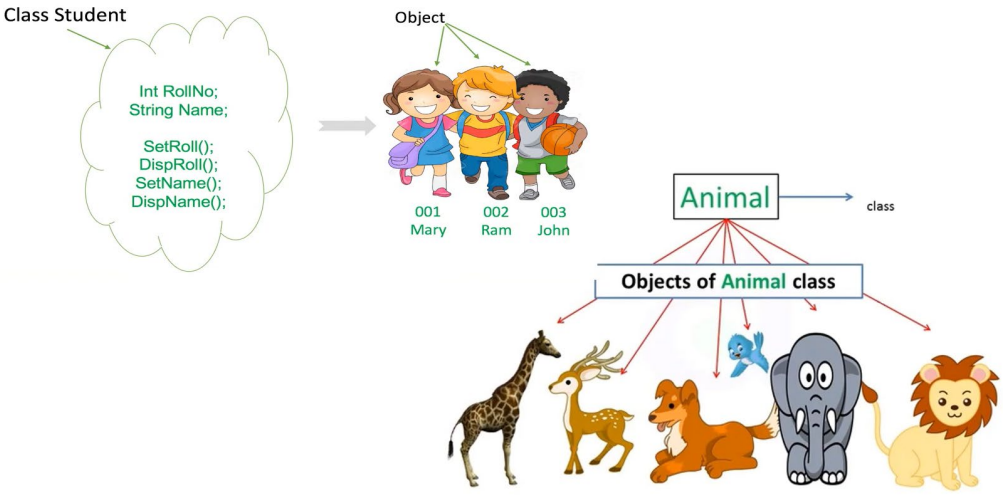
```
let n=123;
n=new Number();

let x=12;
let y=1.2;
let z="x";

// isInteger()
console.log(Number.isInteger(x));
console.log(Number.isInteger(y));
console.log(Number.isInteger(z));

// parseInt() - converts a string into Integer
console.log(parseInt("123"));

s="1234";
console.log(typeof(s));
console.log(typeof(parseInt(s)));

// parseFloat()
s="12.34";
console.log(typeof(s));
console.log(typeof(parseInt(s)));

n=123;
console.log(n.toString());
console.log(typeof(n.toString()));
```

```
true
false
false
123
string
number
string
number
123
string
```

Part 14- OOPS - Class, Object, Methods & Constructor in JavaScript

What is Class & Object

- Class is a Logical entity which contains variables & methods.
- Object is Physical entity & an instance of class.
- Method contains statements. Some times method can accept arguments.
- Class also contains constructors.
- Constructor will be called when you create object for the class.




```
class Student{

    constructor(sid, sname, grade){
        this.sid=sid;
        this.sname=sname;
        this.grade=grade;
    }
   /*setDetails(sid, sname, grade){
        this.sid=sid;
        this.sname=sname;
        this.grade=grade;
    }

    setDetails(){
        this.sid=1;
        this.sname="vamsi";
        this.grade="A";
    }*/

    display(){
        console.log(this.sid+":"+this.sname+":"+this.grade);
    }
}
```

```
let stu=new Student();
//stu.setDetails();
//stu.display()
//stu.setDetails(1, "krishna", "B");
//stu.display()
stu=new Student(1, "krishna", "B");
stu.display();
let stu2=new Student(2, "ram", "c");
stu2.display();
let stu3=new Student(3, "bob", "d");
stu3.display();
```

```
1:krishna:B
2:ram:c
3:bob:d
```

Part 15- OOPS - static keyword in JavaScript

OOP
static

```
class Test{
    static a=10;
    b=20;
    static m1(){
        console.log("Static method");
    }

    m2(){
        console.log("non-static method");
    }
}

// We can access static variables and methods through class name
// but we can't access object variables and methods through class name
console.log(Test.a);
Test.a=100;
console.log(Test.a);
console.log(Test.b); //undefined

Test.m1();
//Test.m2(); TypeError

// We can access non-static variables and methods through objects
// but we can't access static variables and methods through objects
let t=new Test();
console.log(t.a);
console.log(t.b);
//t.m1(); TypeError
t.m2();
```

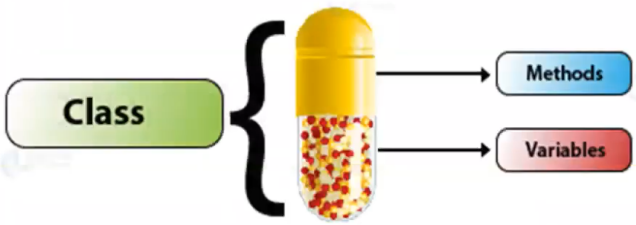
```
10
100
undefined
Static method
undefined
20
non-static method
```

Part 16- OOPS - Encapsulation in JavaScript

OOP
Encapsulation

What is Encapsulation?

- **Encapsulation** is defined as the wrapping up of data under a single unit.
- It is the mechanism that binds together code and the data.
- The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data.




```
// Encapsulation is the mechanism that binds code and data together
class Student{
    constructor(){
        let name, marks;
    }

    setName(name){
        this.name=name;
    }

    getName(){
        return this.name;
    }

    setMarks(marks){
        this.marks=marks;
    }

    getMarks(){
        return this.marks;
    }

}
let stu=new Student();
stu.setName("Vamsi");
stu.setMarks(10);
console.log(stu.getName()+":"+stu.getMarks());
```

```
Vamsi:10
```

Part 17- OOPS - Inheritance | Overriding | super Keyword in JavaScript

OOP

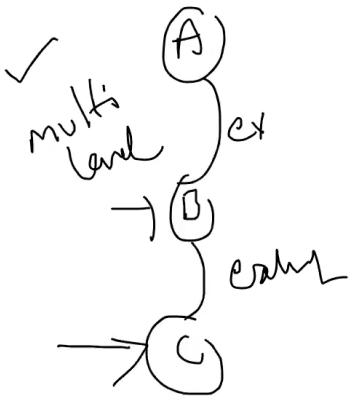
Inheritance

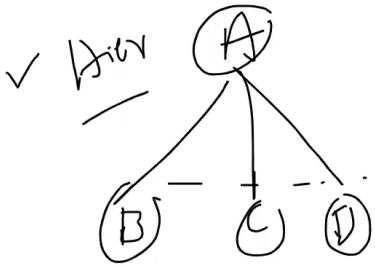
```
Class PersonalLoan
{
//50 Methods

}

Class HomeLoan
{
//50 Methods

}

Class VehicleLoan
{
//50 Methods
}
```

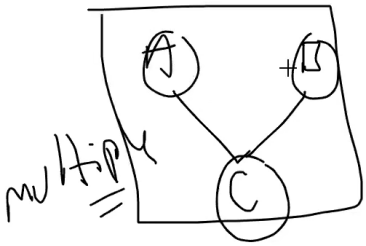
Total Methods: 150
Development Time: 150 Hours

```
Class Loan
{
//30 Methods / Which are common for all 3 Loan modules
}

Class PersonalLoan extends Loan
{
//20 Methods

}

Class HomeLoan  extends Loan
{
//20 Methods

}

Class VehicleLoan  extends Loan
{
//20 Methods
}
```

Total Methods: 90
Development Time: 90 Hours








```
class A{
    a="A";
    b="superB";
    display(){
        console.log(this.a);
    }
}
class B extends A{
    b="B";
    show(){
        console.log(super.a);
        console.log(super.b);
        console.log(this.a);
        console.log(this.b);
    }
}
ob=new B();
ob.display();
ob.show();
```

```
A
undefined
undefined
A
B
```

```
class Bank{
    roi(){
        return 10;
    }
}
class SBI extends Bank{
    roi(){
        return 12.5;
    }
}
ob=new Bank();
console.log(ob.roi());
ob=new SBI();
console.log(ob.roi());
```

```
10
12.5
```

```
class Animal{
    color;
    constructor(color){
        this.color=color;
    }
    printColor(){
        console.log(this.color);
    }
}
class Dog extends Animal{
    food;
    constructor(color, food){
        super(color);
        this.food=food;
    }
    eating(){
        console.log("Eating:"+this.food);
    }
    display(){
        this.printColor();
        this.eating();
    }
}
b=new Dog("Red", "Bread");
b.display();
```

```
Red
Eating:Bread
```

Part 18- OOPS - Prototype in JavaScript

OOP

Prototype

Prototype

- Prototype is an object that is associated with every function & Object by Default.
- If we want to add new properties at later stage to a function/class, we can take the help of prototype.

```
student.prototype.age=35;

s1=new student();
console.log(s1.name, s1.gender);
s1.age=24;
console.log(s1.name, s1.gender, s1.age);
s2=new student();
console.log(s2.name, s2.gender, s2.age);
s3=new student();
console.log(s3.name, s3.gender, s3.age);
```

```
vamsi male
vamsi male 24
vamsi male 35
vamsi male 35
```

```
class Employee{
    constructor(eid, ename){
        this.eid=eid;
        this.ename=ename;
    }
}

Employee.prototype.salary=2500;
Employee.prototype.display=function(){
    console.log(this.eid, this.ename, this.salary);
}
let e1=new Employee(101, "Ban");
console.log(e1.eid, e1.ename, e1.salary);
let e2=new Employee(102, "Sam");
console.log(e2.eid, e2.ename, e2.salary);

e1.display();
e2.display();
```

```
101 Ban 2500
102 Sam 2500
101 Ban 2500
102 Sam 2500
```

Part 19- OOPS - Polymorphism in JavaScript

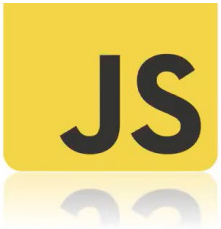
OOP
Polymorphism

- Poly = many
- Morph = forms
- Polymorphism = many forms
- Polymorphism is the ability to create a variable, a function, or an object that has more than one form.

```
class Shape{
    draw(){
        console.log("This is generic shape");
    }
}
class Square extends Shape{
    draw(){
        console.log("This is Square shape");
    }
}
class Circle extends Shape{
    draw(){
        console.log("This is Circle shape");
    }
}
let ob=new Shape();
ob.draw();
ob=new Square();
ob.draw();
ob=new Circle();
ob.draw();
```

```
This is generic shape
This is Square shape
This is Circle shape
```

Part 20- Converting a JSON Text to a JavaScript Object



Read JSON Data using JavaScript

```
let text='{"employees":[{"firstName":"vamsi", "lastName":"pallapolu"},{"firstName":"nani", "lastName":"pallapolu"}]}';
let obj=JSON.parse(text);
console.log(obj.employees[0].firstName+":"+obj.employees[0].lastName);
```

```
vamsi:pallapolu
```

```
data = '{"name": "mkyong","age": 30,"address": {"streetAddress": "88 8nd Street","city": "New York"},"phoneNumbe

let obj=JSON.parse(data)

console.log(obj["name"]);//mkyong
console.log(obj.name);//mkyong

console.log(obj["address"].streetAddress);
console.log(obj.address.streetAddress);
```