Rendering HTML page

Syntax: URL :- scheme :// host . domain.main/path/filename ? query string # anchor

↓

Http → www.google.com/search?q=what+is+javascript%20of&v1=k2&v3=k3 :

https

mail to

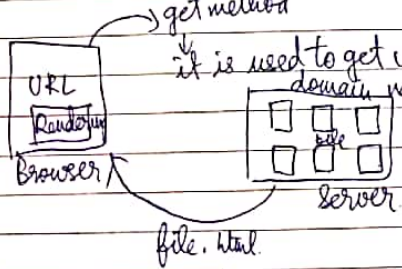
ftp

$ — % 24

Browser :- Anything that is able to display a file on web

Browsers will be at client side

within browser we will type URL



browser will parse the html file.

link external CSS with html page we use link tag

&

In JS:- ~~using this~~ it will render in HTML page and after rendering it will create DOM for javascript.

Usha Gold Page

Java Script

Currently we are using ES-6.

Java Script is dynamic type

⇓

var a = 10

a = "ise" ⇒ here by default 'a' will be changed to character

a = true ↳ boolean

Types of literals

010 - octal decimal

0x10 - hexa

Numeric :- 0, 1, 2.

Floating - -, + values

Boolean - true, false

String literal - it will store characters it can be stored in " " or ' '

Array - [ ]

eg:- a = [1 2 3 4]

a = ["a" "e" "i"]

Regular Expression - it is used to search

Semicolon is optional in JS
Here also there is
do
if
while

JS is a structured, dynamic type scripting language
It is used for interactive behaviour of web pages that is why it is called scripting language

It is platform independent as it can be run on any platform on any browser.
En JS there is no classes.
Here we create objects using constructor functions ∴ it is called Function based
There are lot of built in operations as well like string is an object.
Document object is default object in JS.

JS is prototype based as we use inheritance using functions and objects

Q) Write a webpage to display the info of your favourite politician

Q) Write an algo to check the no whatever user has typed in text box is the same that the browser has guessed

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <title> Favourite Politician </title>
<body>
  <h1> Yedurappa </h1>
  <p> - - - - - - - -
  - - - - - - - -
  - - - - - - - -
  - - - - - - - - </p>
<img src="Yedi.jpg">
 <h2> Vote for Yedurappa </h2>
   <form action
   <input type="radio" name="vote"
        value="vote" checked> Vote for Yedu
                                    </br>
  <input type="submit" value="Submit form">
  <input type="reset" value="Reset form">

</body>
</html>
</head>
```

CSS

Java Script

Difference b/w JS & ebma script

// → comment

```
//old version
var a = 10
console.log(a) → for printing

var a = "ayda"  ⇒ assuming same variable
console.log(a)
```

```
//ES6
    → later we can change value using let
let a = 10
console.log(a)
let a = "hgoke"  ⇒ we cannot redeclare
                   to avoid redeclare we use
                   let or const
```

```
const a = 10          using const we cannot
console.log(a)        change value later
a = "hello"
  ↓
error
```

```
let a = 10
console.log(a)
a = "hello"
console.log(a)

op :- 10
      hello
```

Scope of a variable

1) Global variable → it can be declared outside
2) Local variable → it is declared inside the

```
//old version
var a = 10

function printab()
{                                  ⇒ this will
    var b = a;                       display 9
    console.log(a,b)
}
printab()  → console.log(b) ⇒ this will give error
```

If we use let

```
for(var i = 1; i <= 5; i++)
{                                  o/p
    console.log(i)   ⇒              1
}                                  2
                                   3
                                   4
                                   5
console.log(i) ⇒ this will also
                 work because it is a advantage
                 this can be avoided using "let"
                 and "const"
```

with ES6 we have block scoping
earlier version has only local & global scoping

③ The way we want to print

//old version

var a=16
console.log("the value is"+a)

//ES6
console.log(`the value is ${a}`)
backtick symbol

⇓
here we can enter text with "abc" as well (double quotes)

Categories of objects

objects
- Built in objects (coding side)
  1) String
  2) Regular Expression
  3) Math
  4) Date
  5) Number
  6) Array
- custom → user defined object
- Browser side
  1) Window
  2) Navigator
  3) Document
  4) History
  5) Screen
  6) Location

Properties of string object

1) constructor :- Returns the fn. which creates
2) length

Methods of string object

Arrays

var arr=[1,2,3,4]

var arr2=new Array(1,2,3,4) => default constructor

var arr2=new Array(2c)
↳ single value will declare 2 slot of array

2) We have text box it can i/p any the number, char whatever we write it shd appear in array.
submit button
↳ once we click submit it shd be stored in array.

```
var myValue;
function submit() {
  myValue = document.getElementById
            ("myTextArea").value;
  console.log(myValue);
}

var array = Array();
function add_element_to_array()
{
  array[x] = document.getElementById
             ("text1").value;
  alert("Element:" + array[x] + "Added"
        + x);
  x++;
  document.getElementById("text1").value
        = "";
}
```

Q) To write JS in order to find whether given no is armstrong number

Q) in order to generate the fibonacci series

JavaScript Events

To create event handler

onEvent = "code to handle the event"

onEvent refers to name of an event

↓

user defined

or

built in (we can add)

Form Events

Event	Use
onsubmit	Trigger on submitting form ↳ we can validate particular form.

onselect
oninvalid
oninput
onformchange
onforminput
oncontextmenu
onchange
onblur

Mouse Events

onclick
ondblclick
ondrag
ondragend
ondragenter
ondragleave
ondragover
ondragstart

new options in HTML5

onmousedown
onmousemove
onmouseout
onmouseover
onmouseup
onmousewheel
onscroll

Keyboard Events

onkeydown
onkeypress
onkeyup

Media Events

onabort
oncanplay
oncanplaythrough
ondurationchange
onemptied
onended
onerror
onloadeddata

Events for Media Elements

onloadstart
onpause
onplay
:

Browser Events

onafterprint
onbeforeprint
onbeforeunload
onerror
onfocus
onhashchange
onload
onmessage
ononline

Main

onclick
onload
onsubmit
onmouseover
Image.src

Timer functions

setTimeout(function, delayTime)
↳ this will run only once

setInterval(function, IntervalTime)
↳ after every 5 sec this will be called it will run repeatedly

JS Document Object

~~The doc object is formed when~~
Provides access to all HTML elements

Collections

document.image

Document Object Properties

cookie

Document Object Methods

open
close
write
writeln
getElementById
getElementsByTagName
getElementsByClassName

Properties of a node

Generate Random story

dom - get element by name
↓
use text field

For the story use 3 arrays
3 arrays - for 3 values

on click for random story

we shld use let as there are changes in the words

<

⇒ Design a simple calculator

Emit abbreviations

emmet

shift ! then press enter → html basic format

div enter → div tag

div + a + p → div, anchor & para.

for class
p > class name . class name

for id
p > # id

to generate automatic class names
ul > li > class . item $ * 5

p p[a=e b=d] → user defined tag

a {click here} → anchor tag

Table which has 4 rows & within each row 5 columns
th * 5 > th * 4.

within division tag, p tag within this ul within this li

div > (p > ul > li * 2) + div > p

4) display paragraphs we have to create one event.

5) Display string functions and properties

ex 2

```
function add ()
{
  a = 5
  b = 7
  console.log (a+b)
}
add ()
```

Passing parameters

```
function add (a, b)
{
  console.log (a+b)
}
add (4,5) → if we give add (4)
                ↓
            it will give error
```

```
function add (a, b)
{
  a = a || 0
  b = b || 0
  console.log (a+b)
}
add (b)
```

o/p : 0

add (4)

o/p : 4

add(4, '9') → string

e/p :- concatenate
9 → in white colour it will display

To check data type

parseInt , parseFloat
(or)
is NaN

other method

var add = function (a, b)
{
a = a || 0
b = b || 0
}
add(4, 9)
console.log(add)

return

var add = function (a, b)
{
a = a || 0
b = b || 0
return (a + b)
}
sum = add(4, 9)
console.log(sum)

log of undefined

In JS whenever it is not type it does not perform type checking and it does not specify type of parameter

If there are more no of parameters also it will take whatever we have passed

//ES6

we use arrows here.

not required

const mult = (c, d) =>
{
console.log(c * d)
}
mult(6, 7)

this acts as fn

(c=1, d=1)

Default value can also be pass

const mult = (c=1, d=1) => console.log(c*d)
mult(6, 8)
this also works.

If we have more than we have to execute in a block

const mult = (c=1, d=1) => { return (c*d) }
const result = add(8, 6) + mult(4, 3) / 7
console.log(result)

whenever we specify user defined fun we arrow f

Diff b/w == and ===

```
if (5 == '5')        // this will not check datatype of element
{
  console.log ("I don't check type")
}
o/p                  == it will only compare if it is same

if (5 === '5')
{
  console.log ("I don't check the type")
}
```

o/p: ____ Blank :. this will check type as 5 is int & '5' is string

Array fun

```
let colors = ["red", "black", "orange", "pink"]
colors.forEach((product) => { console.log(product) })
```

o/p: red
black
orange
pink

forEach is used to loop in the array.

3rd property

```
colors.forEach((product, index) => { console.log(
  "At index " + index + " is " + product)
})
```

o/p - At index 0 is red
At index 1 is black

```
colors.forEach((product, index) => {
  console.log("At index " + index + " is " + product)
})
```

↓

To represent in array

Foreach is used to loop through the array to modify element in array

Map fun => used to copy & modify

```
let colors = ["red", "black", "orange", "pink"]
const newColors = colors.map((item) => {
  return item
})
console.log(newColors)
```

o/p: ["red", "black", "orange", "pink"]

const newColors = colors.map(item => { return item + "new" })

console.log(newColors)

o/p : ['redNew', 'blackNew', 'orangeNew', 'pinkNew']

filter

const newColors = colors.map(item => { return item + "new" })

const filterList = colors.filter(item => item !== "black")

console.log(filterList)

o/p :- ['red', 'orange', 'pink']

If we want only black

~~const~~ const filterList = colors.filter(item => item === "black")

o/p : ['black']

let age = [3, 5, 89, 78, 45, 27, 14]

const filterList = age.filter(item => item > 18)

o/p : [89, 78, 45, 27]

let ar = ["tiger", 89, "lion", 90]

const filterList = ar.filter(item => isNaN(item))

console.log(filterList)

o/p ['tiger', 'lion']

const - - - (item => parseInt(item) == item)

o/p : [89, 90]

(item => parseInt(item) = item)

o/p ['tiger', 'lion']

Number object

let num = new Number(value)
or
const

vs code let n = 5
n.
Number.

Math object

Date object

Q) Demonstrate all 3 objects.

Regular Expressions

[ ]
text box
This should accept only characters how to write (A-Z)
26 characters

ex [a-z] → find any char b/w the brackets
only a b c ⇒ [abc] | [a-c]
[0-9]

4 to 8 [4-8]

for . * $ & % 0 123 except abc

[^a-z]
they don't accept the range

for only . * $ 2 % we use
[^a-z] [^0-9] (^ → not symbol)
↓
Find any char not b/w the brackets.

[^a-z] | [^A-Z] | [^0-9]

Meta Characters

\n → new line character

To find single character we use .

+ - one or more characters

? - zero or one

* - zero or more characters

\w - find a word [a-z][A-Z]

\W - Non-word - not [a-z] | [^A-Z]

\d - digit [0-9]

\D - Non digit [^0-9]

\s - white space character

\S - Non-white space character

\0 - NULL character

\t - tab character

\xxx - octal no

\xdd - hexa decimal no

Quantifiers

n+ → Matches atleast one n in the string

n* → Matches zero or more occurances of n

n? → Matches any string that contains zero or one occurance of n

n{x} →

n{5}
nnnnn

n{\d}
n 9 or n 4 any times

n{x,y}

n{2,5}
range

n{x,}

eg: n{5,}
any no of times

n$ → Matches any string with n at end of string

eg: a$ ——— a

^n → It matches any string with n at beginning

^A eg: A_____ , B_____ , A_____

start with A followed by any no of characters ⇒ ^A\w eg: A\w{5}

?=n → matches any string followed by string n

?≠n → matches any string i.e not followed by specific string n

eg:- USN
1NT16ISC01 → alpha numeric
10 digits

1 number 2 alpha 2 nos
2 alpha 2 nos

Ans \d\w{2}\d{2}\w{2}\d{3}

ES6 → let, const

const Regular Expression

using RE literal
const RegularExpression = /pattern/flag

using RE constructor fn

const RegularExpression = new RegExp
("pattern", "flag")

flags
if i
globally - g
insensitive - i
search

I, i

It will return I

iii - g → i
Iii → i → it will return I

gi means both

m → multiline matching } flag

Regular Exp for mail ID

start → no x
spl char x
alphabet ✓
underscore x
dollar x

In between → . ✓
nos ✓

\w{1,10}

\d{10}

Event handling

how you shd respond for a particular action
Event handlers are also called event listeners

There are many types of event handling properties
on click
on mouse etc

(Add Event Listener)
3 attributes ~~event handler~~ will take
1) boolean values – true, false
2) event name
3) Event handler function

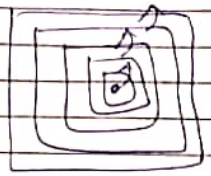
Query selector → will highlight first occurance of particular word

on blur
on mouseover
on mouse out.

Phase

Event Capturing Phase
Bubbling Phase
Target Phase

Bubbling phase



Event Delegation

Instead of attaching all event listeners

Demonstrate query selector all
on mouse over
on click

Event Capturing

User defined objects

// object creation → this can be done using literals

3 ways

1) object literal – also called object initializer

eg:- let emp = { } → property or method can be typed

syntax:- let objName = { p1 : v1 , p2 : v2 - - - }

v1 = value

v1 = function { } → method

let emp = { name : 'abc', id : 123, sal : 23000 } → advantage here is we can include many properties for single variable

console.log (emp)

o/p. { name : 'abc', id : 123, sal : 23000 }

Property is a normal variable associated with an object

For one particular property :- console.log(emp.name)

o/p :- { name : 'abc' }

2) // using object instance

let person = new Object () → default constructor

person.name = 'xyz'

person.age = 56

console.log (person).

3) // constructor function

function car (year, model, name)

{

this.year = year; → year for this set the value what user has given

this.model = model;

this.name = name;

}

let myCar = new car (2019, 'ABC1', 'tesla')

console.log (myCar)

o/p : car { year : 2019 , model : 'abc1', name : 'tesla' }

let emp = {

name : 'abc',

id : 123,

sal : 23000

}

emp ['year of join'] = 2000 → adding new object using box bracket

console.log (emp)

In JS objects are also called associative arrays

Object within object → parent object is car and engine is child object

```
let car = { color : 'red',
            wheels : 4
            purchase year : 2020,
            engine : { cylinders : 4
                       size : 2.2
                       model : 'K2'
                     }
          }
console.log (car.engine.model)
```

o/p :- K2

Using function

```
function car (color, wheels, year, engine) {
        this.color = color;
        this.wheels = wheels;
        this.year = year;
        this.engine = engine;
function engine (cylinders, size, model) {
        this.cylinders = cylinders;
        this.size = size;
        this.model = model;
    }
car myCar (
engine )
```

Delete a property

literal method

```
let car = {
        year : 2019,
        model : 'high'
    }
console.log (car)
delete car.model ;
console.log (car)
```

Object Methods

① // object literal

```
let car = {
    color : 'red'
    year : 2020,
    display : function ()
{
    console.log (`the car is ${this.color} is color and purchased in the year ${this.year}`)
    }
}
car.display ()
```

(this refers to car object)

2) Object instance

```
let person = new Object();
person.age = 45
person.name = 'abc'
person.display = function()
  {
    console.log(`My name is ${this.name}`)
  }
person.display()
```

3) Constructor function

```
function emp(fname, lname, sal)
  {
    this.fname = fname;
    this.lname = lname;
    this.sal = sal;
    this.display = function(){
      console.log(`Name is ${this.fname}
                   ${this.lname}`)
    }
  }
let e = new emp('Raj', 'Ram', 900)
e.display()
```

this.display = display (not this also)

Don't use arrow to specify fn here.

// comparing of objects

```
let fruit = {
    name: 'apple'
}

let f = {
    name: 'apple'
}

console.log(fruit == f)
console.log(fruit === f)
```

} both will give false we cannot say both objects are equal

only if we assign the objects we get it as equal

```
let f = fruit;
console
```

// this keyword
// old JS

```
let person = {
    name : 'abc',
    age : 29,
    sayDetails : function ()
    {
        console.log ('Name is' + this.name +
                     'age is' + this.age)
    }
}
person.sayDetails()
```

same program

```
{ console.log (          )
    sayAgain = function ()
    {
        console.log ( that.name )
    }
    sayAgain()
}
```

that = this

if we use this it will work

⇒ this will give o/p as undefined

// ES6

Refer phone

// getter → get and set method
setter

```
var obj = {
    a: 7,

    get b()
    {
        return this.a
    }
    set c(x)
    {
        this.a = x/2
    }
}
console.log(obj.a, obj.b)
obj.c(50) = 50
console.log(obj.a, obj.b)
```

```
function Emp(name,dept) {
    this.name = name || '';
    this.dept = dept || 'general'
}
let e = new Emp();
console.log(e);

o/p:- { name:'', dept:'general' }

let e = new Emp('emp1', 'testing');
o/p:- { name:'emp1', dept:'general' }
```

```
function Emp(name,dept) {
    this.name = name || '';
    this.dept = dept || 'general'
}
function Pemp(name,dept,projs)
{
    this.base = Emp
    this.projs = projs || [];
}

let pe = new Pemp('emp2', 'dept1', ['proj1', 'proj2']
console.log(pe);
```

Alternate method

```
function Pemp {
    Emp.call(this.name, dept)
    this.projs = projs || [];
}

function emp
```

```
function Car(name, model) {
    this.name = name || '';
    this.model = model || 'standard';
}

function Pcar(name, model, price) {
    Car.call(this, name, model)
    this.price = price || [];
}

function Cars(name, price, type) {
    Pcar.call(this, name, model)
    this.type = type || '';
}

let ch = new Cars('Benz', '1000', 'm1')
console.log(ch)
```

Map Set

// map

let capitals = new Map() → this is used to store (K,V) pairs
↳ it is a class object
it inherits from parent.
+ 2 like unique value

let capitals = new Map([1,2], [3,4])

let capitals = new Map() // empty map
capitals.set('bangalore', 'KA');
// many others K V

capitals.set('Mumbai', 'MH'); K's cannot be duplicate.

console.log(capitals.size) O/P :- 2

~~capitals~~ console.log(capitals.get('bangalore')) O/P: KA

console.log(capitals.has('Mumbai')) O/P :- true
↓
as value is present

capitals.clear ⇒ all (K,V) pairs will be deleted
capitals.delete ⇒ delete particular value

capitals.delete('bangalore')
(console.log(capitals) O/P :- Mumbai
↳ console.log(capitals.has('bangalore'))
we can use this also

capitals.clear()
console.log(capitals)

console.log(capitals.entries()) ⇒ to display all entries

let a = capitals.values() ⇒ only values will be displayed
console.log(a)

for (let [key, value] of capitals)
{
console.log(key + ' is capital of ' + value)
}

o/p:- karnataka is capital of Mumbai
tamilnadu ... Mt

let obj = new Object()
console.log(obj instanceof Map) } inheritance
let m = new Map()
console.log(m instanceof object) }

o/p:- true

Set.js

//set

① let myset = new Set() //Empty set

② let set1 = new Set([1,2,3,4]) //entering value

In array we can store duplicate values but in set we cannot store duplicates

console.log(set1.entries())

o/p:-
[1,1]
[2,2]
[3,3]

let set1 = new Set([1,2,3,4,5,'string',{"objectkey":"objvalue"}])

set1.has(2) ⇒ to check if key is present or not

console.log(set1.has('string')) o/p:- true

if (set1.has('string')) {
set1.delete('string')
}
console.log(set1.entries()) } deleting

```
if (set1.has('string')) {
    set1.delete('string')
}
else {
    set1.clear() //to delete all entries
}

set1.add(78)
console.log(set1.entries())  -> o/p: [1,1]
                                     [2,2]
console.log(set1.size)

console.log(set1.value())  o/p:- only one no [1,1]

for (let i of set1.values()) {
    console.log(i)
}                          //just display values not string

set1.forEach(
```

// Converting set into an array

```
let set2 = new Set([3,56,'a string'])
console.log(set2)          o/p:- {3, 56, 'a string'}

let set2 = new Set([3,56,'a string'])
console.log(set2)
let ar1 = [...set2]  //destructuring
console.log(ar1)           o/p: [3, 56, 'a string']
```

// Converting array into set

```
let ar2 = [45,78,34]
let setOne = [...new Set(ar2)]
console.log(setOne)
```

o/p:- [45, 78, 34]

=> // Remove duplicate elements from array

```
let ar3 = [2,2,3,3,4,55,5,6,6,7,1,2,3]
```

// 2 ways

① Use in called array.from

```
let ar3Set = Array.from(new Set(ar3))
console.log(ar3Set)
```

o/p: [no duplicates]

// 2nd way

```
let s3 = new Set(ar3)
console.log(s3)
let ar4 = [...s3]
console.log(ar4)
```

// Performance efficiency

```
let a=[], s = new Set()
let b = [1,2,3,4,5,6,7,8,9]

for(let i=0; i<b.length ; i++)
{ console.time('Array')
  a.push(b[i])          // copy b to a
  console.timeEnd('Array')
  s.add(b[i])           console.time('Set')
} console.timeEnd('Set')

// Access
let result   console.time('Array')
result = a.indexOf(2)!==-1;
console.timeEnd('Array')
result = s.has(2)        console.time('Set')
console.timeEnd('Set')
```

conclusion:- Set is more efficient

⇒ Demonstrate the Map and Set Data Structure use it for objects also (using various values)
Compare performance b/w object & Map.

Add objects to Set

```
let ob1 = {id:1, name:'abc'}
let ob2 = {id:2, name:'x y z'}
s.add(ob1)
s.add(ob2)
console.log(s)
```

// Map

```
let m = new Map()
m.set
```

Performance Analysis for Map

Convert object to Map

How do you convert object to Map

Perform any one set (union, intersection, difference)

Day Run-time create object and store in array

Login Page

to create as user login first time he will enter the details

Obtain login info from user create object and store in array

```
let k = 'user'
let i = 1

let m = k+i

m= new Object();

outside loop <- let ar =
        ar[i] = m.

m.name = 'abc'
m.age = 12
m.phnum = 13345-1
console.log(ar)
```

Urban Guild

Promise JS

It is used mainly for asynchronous operations.

Synchronous → Event handling - button - alert, <p>

JS single thread

Asyn → fetching particular file/server video streaming

fetch(file), display

↓

possibility there may be error : server crash busy etc

// Promise - real time promises
// Scenario - writing exam - pass - partly fail - tumple

// in terms of JS
// writing exam - Promise
// waiting for result - asyn operations in JS
// Pass / Fail - Promise return value
// Pass - Promise is fullfilled
// Fail - Promise is rejected
// Having Party - success callback
// Tumple - failure callback

Creation of promise

let a = 2+3

let p = new Promise(function(resolve, reject) {
  if (a == 5) {
    resolve('Success')
  } else {
    reject('Failure')
  }
})

console.log(p)

⇒ we have to write failure callback otherwise we'll get warning.

How to write callback

Promise chaining for success

p.then((message) => {
  console.log(message + 'then is executed')
})

for failure

p.catch((result) => {
  console.log(result + 'catch is executed')
})

// Failure callback

don't use p1, p2 individually

for 2 error callback

console.log(p)

p.then((message) => {
  console.log(message + 'then is executed')
}).then((msg) => {
  console.log('multiple then can be executed')
})

const p1 = new Promise((resolve, reject) => {
  resolve('Video 1 is recorded')
})

const p2 = ...
  resolve('Video 2 is recorded')
})

const p3 = ...
  resolve('Video 3 is recorded')
})

// all promises together

Promise.all([p, p1, p2, p3]).then(
  (message) => {
    console.log(message)
  }
).catch(
  (message) => {console.log(message)}
)

Promise.allSettled

{

}

O/p: whether failure or success one promise is fulfilled it will display all S & F

Promise.race([

not utilizing external things video or anything

promise to be completed that will be displayed

]}. ✓

Asynchronous way

Async JS

```
const one = () => {
    console.log("This is one")
}

const two = () => {
    console.log("This is two")
}

const three = () => {
    console.log("This is three")
}

two()
one()
three
```

Data structures using JS

```
Stack (-) => Array
{
    push()
    pop()
    disp()
    is_empty()
    is_full()
    top()
}
```

s = new Stack()
s.push[10]

create a new stack user defined and perform all operations

Different types of scopes in JS

1. Global scope
2. Local - this existence is confined only to the fn.

Closure Property

1) Functions will be bundled together
2) Accessing outer fn through inner fn provided inner fn is within the scope of outer fn

similar to inheritance

If inner fn is present within scope of outer fn then we can access variables of outer fn

We have 3 types of scopes in JS
1) Local scope
2) Global scope
3) outer fn scope

Different Access specifiers
1) Public
2) Private
3) Protected

Static - we can invoke fns and variables of fns without creating objects

Main is not returning anything

~~We are passing arguments of~~

String args() → we are not returning anything here.

In JS while using closure we might want to restrict some of the variables this flexibility ~~will be~~ is also available in JS by declaring as private

oops using JS

```
const person = {
    property : 'hello'
    fun : function() {
        console.log("I am a Person")
    }
}
const me = new
const me = object.create(person)
me.property = 'byee byee'
me.name = 'alex'
console.log(me, person)
me.hasOwnProperty
```

we use : only in object literal

do both

replace with

```
console.log(me)
console.log(person)
me.hasOwnProperty ('it is a property')
console.log(me.hasOwnProperty('property'))
                    ('name'))
console.log(person.hasOwnProperty('name'))
```

constructor fn

constructor

```
function c() {
    c.meth1 = function() {
        console.log("This is C")
    }
}
```

c.prototype.meth = function()

Here can use this. method call

```
let c1 = new C()
c1.meth1()
```

constructor

Create Class

```
class C {
    constructor() { // setup the object
    // instance properties -> unique for each object
    console.log("I'm in class C")
}

// instance methods
meth1() {

}

let c1 = new C()
let d1 = new D()

let c1 = new C()
let c2 = new C()
console.log(c1)
console.log(c2)
```

// whenever obj is created constructor will be invoked

Create user database give inputs (generic database)

- make login
- make concepts :- array concept
  number object

Node.js

```
class Animal {
  constructor(name) {
    this.name = name
  }
  makeSound() {
    console.log('generic animal sound')
  }
}

class Dog extends Animal {
  constructor(name) {
    super(name)
  }
  makeSound() {
    super.makeSound()
    console.log('woof woof')
  }
}

const a1 = new Animal()
a1.makeSound()

const a2 = new Dog('Bruno')
a2.makeSound()
```

Node.js