Css specific terminologies:

Attribute selectors
Css colors
Rounded corners
Box shadow
Text shadow
Multiple background images

Null is object type
0==“0”. Then. True
0===“0” then False

Typeof operator to check the datatype
typeof(undefined) is undefined.

Push—at the end of the array adds an item
Unshift—start of the array
Pop—last item will be removed
Accessing values from keys in object:
. Notation :objectname.keyname
[] notation:objectame[keyname]
Object.key.function is method

A function which is accessed through key is method.

Constants means literals in java script
Which can’t be changed.Eg :1,sowji etc

Variables are the containers which holds the data

Var name = prompt(“enter your name”);
Console.log(name);

Asks the username.

Lexical scope

Es6:let and const

Let is var

Let name="jjj"//can be changed

Const name="jj"//which can't be changed

Hoisting nature of javascript:

Var means local or global can be accessed
Let means local or global matters

Es5:

```
Var person={
        firstName="sow",
        lastName="Bojja",
        Get : function(){
                console.log("Outer"+this.firstName+this.lastName);
                        var print =function(role){
                                console.log("inner"+this.firstName+role);
}.bind(this,"developer")
print():
}}
```

ES6:

```
Let person={
        firstName="sow",
        lastName="Bojja",
        Get : function(){
                console.log("Outer"+this.firstName+this.lastName);
                        let print =()=>{
                                console.log("inner"+this.firstName);
}
print():
}}
```

Person.get()

For outer function(Get : function()) we can not use arrow function

Es5:

```
Function print(a  ,  b){
Return (a+" "+b);
}
print(s,f)
```

Es6:

```
Const print=(a , b) =>{

        Return (a+" "+b);
}
print(s,f)
```

For no arguments:

```
Const print=() =>{

        Return (a+" "+b);
}
```

Map:

```
a=[1,2,3]
a.map(function(item){

Return item*5
})
```

O/p:[5,10,15]

Filter:

```
a.filter(function(item){
    true
}
)
```

O/p:returns everything

```
a.filter(function(item){
    return item==1
}
)o/p:[1]
```

Reduce:

```
A.reduce(function(x,y){
Console.log(x , y)
Return x*y
```

})
O/p:

First element (1),second element(2)
1*2=2
2,Third element(3)
3*2=6

Es5:
In ES5 we combine strings or literals with + symbol
Eg:
My name is +" " +firstname+" "+lastname
In ES6:
We can use $ symbol to replace dynamic literals:
Eg:
My name is ${firstname} ${last name}
Multiline string can be executed without the use of \n operator.

```
Function greet(name){
        var greeting = "HI"+name;
        return function(){
                console.log(greeting);
                        }
sayhello=greet(sow);

sayhello()
```

Closure:

```
Function welcome(name)
{
var greeting = "HI"+name;
        var message= function(){
                console.log(greeting);
                        }
Return message;
}
Var sayhello=welcome(sow);
```

Hoisting:
Variable can be define outside the scope
Arguments can be passed as array with apply and accepted as individual parameters where as cal is used to pass as individual parameters

Call and apply will be used as arguments inside one function and called for

seperate object.

Spread operator:
When you wish to transform array as individual elements.
Gets=function(a , b , c){
Return a+b+c;
}
arr=[1,2,3]
Gets.apply(null,arr) gives result as 6
We can use spread operator to simply this.
Gets(...arr)
We can use spread operator for array concatenation also.

[...a ,...b] where a=[1,2,3] and b=[4,5,6]

We can use spread operator to copy an array
a1=(...a) where a is array
We can copy object also.
Rest parameter:
Opposite of spread operator.
We pass individual parameters which will be used as array in function.
Rest parameter must be the last formal parameter for any function.

For arrow function arguments keyword is not defined it will be solved using rest parameters

Const f=(...args)=>{
Console.log(args[0])
}

destructuring:
Restructuring the array into individual elements.
Const mobile=['91','4455','4555']
const[x,y,z]=mobile
So x=91
y=4455
z=4555
const[x, ,z]=mobile
x=91
z=4555
Similarly object also destructed.

Const customer{
fn="def'

ln=“ivf”
}
Const {fn,ln}=customer
We can give aliases also
Const{fn:f,ln:l}

Javascript is synchronous programming language that means tasks are performed step by step.
To achieve asynchronous in js we will use settimeout .
Syntax:
Settimeout (()=>{
Console.log...
},3000);
Other methods are:cleartimeout,clear interval,setinterval

These are used in methods in event listener, methods which are used as API calls for the server, geolocations, file extensions etc.

Callback:A fn passed as an argument to another fn.
It can be used when certain task is finished callback will be invoked.
Example:
Document.getElementId(“mybtn”).addEvenListener(“click”,
()=>{alert(“button clicked”);});highlighted one is(alert fn)is call back fn.

Promise:
A javascript object which provides linkage between producer and consumer.

Let promisobj = new Promise ((resolve, reject) = >{
//producing code
});
Producing code is executed automatically as soon as the promising object is created.

Internal properties after creating promise object.
Promise status: pending
Promise value: undefined.

Producer result can be success(resolve callback fn) or failure(reject call back fn)

For resolve(val):

Stutus:resolved
Value:val
For reject(error)
Status:rejected
Value:err

Consumers:
Then method and catch method

Then method will call two arguments for resolve and reject callback methods which in turn accepts the arguments s value and error.
promiseObj.then(successcallback,errorcallback)
promiseObj.then((val)=>{//code to be excecuted when promise is resolved}
PromiseObj.then((err)=>{//code to be executes when it is rejected})
Example:

```
Let promiseObj= new Promise((resolve, reject)=>{
Console.log(“getting name from db..);
setTimeout(()=>{
resolve(‘sow’);
},3000);
});
//consumer-then method:
then(val =>
{console.log(‘name = ${val}’);},
(Err) => {console.log(‘error’ = $(err)’);
});
```

```
promiseObj.catch(errorcallback);
promiseObj.catch(err = > {//code to be executed when the promise is rejected.
});
//promise object with the parameter
Let promiseObj= new Promise((resolve, reject)=>{
Console.log(“getting name from db..);
setTimeout(()=>{
Reject(new Error(“could not get name from DB”));
},3000);
});
//consumer-catch method:
promiseObj.catch(err) =>
{console.log(‘Error occurred = $(err)’);});
```

There are two methods to define the consumer for producer code:

Method1:for both resolve and reject then method is used with the parameters val and err
Method2:then method for resolve with parameter val and catch for reject with parameter error.

Multiple callbacks leads to the ppm of callbackhell this will be resolved using promises

If we keep async keyword infant of any arrow fn it returns promise fn.

Eg:
Let fn = async() = > “sow”;
Fn.then(alert);is same as
Let fn=async()=>Promise.resolve(“sow”);
Fn.then(val => alert(val))

Async and await easy than promise with try and catch
Try will use value with await and catch takes the error

React Js Advantages:
React helps to build User interactive interfaces.
You can use latest features introduced in the JS language.
React does not demand to rewrite or ship your existing code in it.
React follows the concept of virtual DOM , which makes DOM manipulation super fast and easy.
Bundling:combining all css files and all script files
Magnification:removing unnecessary code
Webpack:used for bundling.
Lint:corrects the code
Jsx:js+xml+php

Cntrl+A and alt+shift+f —— for formatting
ctrl+/ for commenting

Jsx vs html:
className instead of class
Label htmlFor instead of label for
defaultValue instead of value
Jsx is case sensitive because js is case sensitive
Converting jsx into js:

Using React.createElement(element_name,element_properties(optional),infinite arguments(optional))
Jsx:input id="name"type="text"
Js:React.createElement(input,(id:name),type="text")
Jsx:

React

Js:
React.createElement(div,{id="module"},
  React.createElement(p, null, "React"))

Components:
Dynamic,independent and reusable
Types:
functional(stateless):pass data to component and returns react element
class(state):additional features but need to be handled otherwise goes into inconsistent state

Components should start with capital letter to differentiate from html files

State:inside only class because updating will render the method(class components)

Prop:external and maintained by whatever component that is called.(functional components)

Components life cycle:
Mounting:
Instance of component created and inserted into DOM
Updating:
Components updated by props or state
unmounting:
Component removed from DOM
First constructor is called
Then render is called
Then componentdidmount is called

Nodejs:

Node.js is js runtime built on chromes's v8 js engine
Server side javascript which does not require browser unlike client.

DOM is not available.All features that support client side will not support on server side and vice versa
REPL:
Read input from user
Evaluate input and return
Print or display result
Loop previous commands until termination
Every node.js follows the steps :Code—compile—execute
REPL bypass the compile step