51)
```
function (){
    for( i=1; i< 3; i++){
        setTimeout( function(){
            console.log(i);
        }, i*1000);
    }
}
```

This will print 3 after 0 seconds, 1 second, 2 second. Because when the setTimeout executes, for loop has already ran and i becomes 3. Now console inside setTimeout searches for a global copy of i which holds 3. Now if we want to print 1, 2, 3, we need to use let:

```
function (){
    for( let i=1; i<3; i++) { ..... } }
```

Now distinct i will be there for each iteration.

or

Use sdf function which will allow each iteration to have new scope over which timeout function can close and when we pass variable i to sdf executing function then each iteration's timeout function will have its own right value to access.

```
( function (i){
    setTimeout() .....
})(i);
```

// Passing i is mandatory else it will take global copy

Why it works with let? :- Because let's say three scope of timeout function will be created for each for iteration. When we used var, each scope is pointing to same GLOBAL variable but let is a BLOCK scope, so each ~~it~~ scope of timeout will be pointing to different variable (i.e. 0, 1, 2).

52) function aa(){
return {
test : 1
}
}

console.log(typeof(aa)) → // undefined

No, it will not be object because JS expects a semicolon at end of every line and { is on the next line, so it will add ; which becomes return; means return nothing and hence undefined.

return {
test : 1 // object as { is on same line.
}

53) f bar(){
return foo;
foo=100;
f foo(){}
var foo = '11'
}

what is value of bar() type?
i.e. typeOf(bar());

In this case, hoisting comes into action. Variable declaration for line 3 and 5 and function hoisting of line 4 will be done. Since variable initialization is not hoisted, 3 and 5 gives undefined and 4 will make foo = function. So, typeOf(bar()) will be 'function'.

<u>Note</u>:- Functions declaration have priority over variable declaration but not variable assignment.

① var foo;
f foo(){};
type of foo (function)

② var foo=10;
f foo(){}
type of foo (number).

Here priority means that variable with same name will be assigned with high priority value and not that there sequence. (~~Seq~~ all hoisting is done parallely).

for 2nd example:-
var foo; (undefined) → but function have high priority. so, → f foo(){} (function) → but now, we set foo=10; which overwrites function with number.

1) Semant
Elemen
browser

2) Async
Defer s
parsing
finish

Async
and
before

If rec
and
until

Javascript.

1) alert ([] + [] + 'foo'.split)

'foo'.split → ['f','o','o'] + [] + []
(object).

empty array coerces to empty string. So,

output = 'f,o,o' (string)

2)
```
arr = [1,2,3]
arr.length = 0
arr.push(4)
alert(arr[0]) → 4
```

arr = [1,2,4]
arr.foo = "property"
arr.length = 3 (remains 3)
We can read this property in for...in and not in for.

3)
```
f foo(a,b){
  arguments[1] = 2;
  alert(b)
}
foo(1);
```

Output = Undefined.

foo needs 2 arguments. If we don't pass all args, JS automatically set it to undefined. ~~sets b undefined~~.
Now argument[1] = undefined, which you cannot now set to Number. If we passed any initial value, then arg[1] = 2 would have worked.
argument[1] = undefined does not mean b = undef.
It means argument 2 is undefined. That is why we can't replace it.

4) An object consists of following property.

```
object.getOwnPropertyDescriptor(myObject, "a");
↓
{
  value: 2,
  writable: true | false,
  configurable: true | false,
  enumerable: true | false
}
```

```
var myObject = {
    a: 2
}
```

(By making writable and configurable as false, it becomes an object constant).

5) How to prevent an object from having new properties?

```
Object.preventExtensions(myObject);
myObject.b = 3;
myObject.b   // Undefined.
```

Here, you can reconfigure or delete existing property.
To prevent that as well:-

```
Object.seal(....)   // You can modify value of the property.
```

To stop writable as well, use:-

```
Object.freeze(...)
```

6) Every() and some()?

* forEach() iterates over all values but every() and some() are different.
* Every() executes till end or until callback returns a falsy value. It checks whether all element satisfy any condition.
* Some() executes till end or until callback returns true. It checks whether atleast one element satisfies given condition.

```
arr = [2, 4, 1, 6, 8];
var odd = arr.some(isOdd);   // iterates till 1 is true
```

Output of odd = true;

(isOdd is function which returns (num % 2 == 0))

```
var odd = arr.every(isEven);   // iterates till 1 is false
```

(isEven is function which returns (num % 2 == 0))

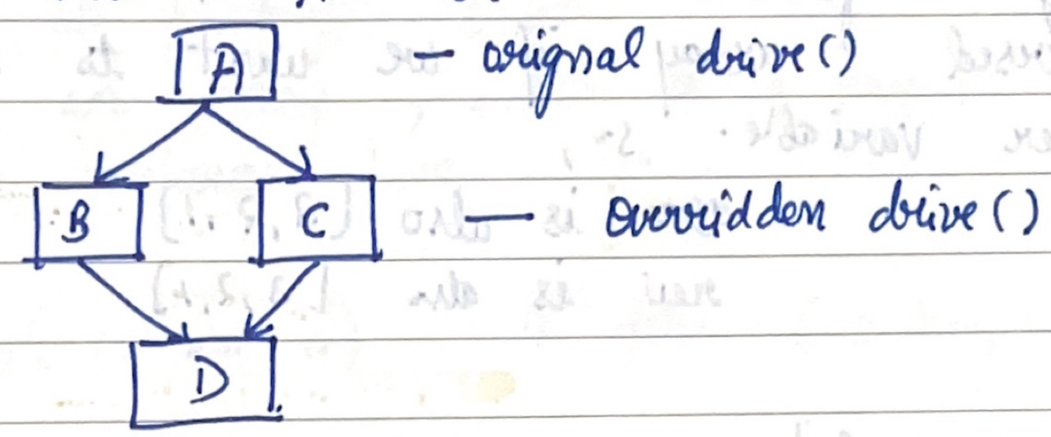
7)
```
f(a,b){ ... }
var value = f(1,2,3,4)
```

What is value of arguments.length inside function?

Ans: It will be 4. Arguments.length reads all arguments passed while calling.

8) What is diamond problem of inheritance?

When during multiple inheritance, D inherits from 2 parent class B and C and B and C inherits from 1 parent A where B and C override a same method drive() from A then which drive() should D call — A's or B's or C's.



<u>Note</u>:- JS does not thus provide Multiple inheritance.

9) What does push, pop, shift, unshift returns.

```
arr = [1,2,3];
console.log( arr.push(8) )   // returns 4
                             (length of array)
console.log( arr.pop() )     // returns 3 (last element)
console.log( arr.unshift(4)) // returns 4
                             (new length of array)
console.log( arr.shift() )   // returns 1
                             (first element of array)
```

10)
```
arr = [1,2,3]
rev = arr.reverse()
result = rev === arr
console.log(result)   // true
```

True, Because ~~arr~~ reverse function reverses the original ~~function~~ array ~~array~~ as well and returns a reversed array if we want to store it in any other variable. So,

arr is also [3,2,1]
rev is also [3,2,1]

11) What is eval?

eval function evaluates JS code represented as a string and returns its final value.

```
eval("4+4");      // 8
eval('2+2') === eval('4')   // true
eval(new String("2+4"))
// [String: '2+4']
new String(eval("2+4"))
// [String: '6']
eval("a*b")       // 16   (a=4, b=4)
```

Basically, if argument of eval is an expression, evaluate the expression. If argument is one or more JS statements, execute the statement.

12) Va
log (
IN opera
value) is
or maybe
So, for
2 in

13) What

The exp
evaluates

14) ~ (bit
Bitwise NOT
removes al

~
~
~

It does not

12) Var a = [10, 20, 30]

log (20 in a) // true

IN operator returns true if specified property (not value) is in specified object (array is an object too) or maybe its prototype chain.

So, for an array, index is a property. Thus 2 in a returns true

log(10 in a) // false (as length is 3).

13) What is 3 ** 3?

= 27

The exponentation operator ** (assignment operator) evaluates to :-

(first operand)^second operand

$(3)^3 = 27$.

14) ~ (bitwise NOT)

Bitwise NOT inverts all bits and returns a whole number, removes all decimal value if any.

~ ~ (2.52) ⇒ 2 ~~2.812~~
~ (2.52) ⇒ -3
~ (2.42) ⇒ -3

You must think how? Because,

It does not work in integer, it works in bits. So,

a = 5 // 0000000 101
~5 // 1111111 .... 010 ⇒ -6
a = -3 // 11111111 01
~a // 000000010 ⇒ 2

So, ~5 does not mean we are negating value 5. It means, we are negating bit value of 5.

Operand → converted to 32 bit signed integer → Numbers with more than 32 bits get their significant bits discarded → Invert bits and return number.

15) Expressions with && and || (logical and / or).

AND :- Returns expr1 if it can be converted to false otherwise returns expr2. For Boolean values, returns true if both true, else false.

console(true && 300) - 300
(200 && f()) - f()
(0 && 10) - 0
(false && 10) - false
('a' && 'b') - 'b'

OR :- Returns expr1 if it can be converted to true else expr2

console (true || 300) - true
(false || 300) - 300
('a' || 'b') - 'a'
(0 || 10) - 0

16) fun = (x = 262) => { }
fun (null)
x = ?
x = null
Initial value works if no value or undefined is passed (Not NULL).

17) "good" > "bad"
True, because strings are compared lexicographically in JS and g > b.

18) Number. MIN_VALUE < 0
false, because MIN_VALUE is the smallest positive numeric value in JS. (approx 5e-324) i.e. as close as you can get to 0 but not 0.

Overall smallest value is Number. NEGATIVE_INFINITY (-∞) but that's not really numeric in strict sense.

19) Object.is (NaN, NaN), — True
Although NaN === NaN, but Object.is does strict match.

20) Number.parseInt ("1995Hello"); → 1995
It parses string argument into integer. If first letter cannot be converted to number, then it returns NaN. So,
Number.parseInt ("Hello"); → NaN.

21) Array Join with argument and without argument.

[1,2,3] · join () → 1,2,3
[1,2,3] · join ('') → 123
[1,2,3] · join (' ') → 1 2 3
[1,2,3] · join ('d') → 1d2d3

So, if no argument provided, comma is used by default.

22)
```
class User {
    function def() { ... }
}
```
type of User → function

Class declaration creates a new class using prototype based inheritance) just like "function User() {} ; User.prototype.def = function() {}." type of returns "function" string

Note:-
```
function get() {
    return "name"
}
```
typeof get – function, type of get() – string.

23) Comma operator.

The comma operator evaluates all values from left to right and returns the last operand. The precedance is lowest of all operator.

return (1, 2) ⇒ returns 2.
result = (2, 4, 6) ⇒ result = 6.
x = (y=1, z=4) ⇒ x = 4.

24) Sets in javascript.

Set object lets you store unique value of any type, whether primitive values or object references. A value in set may only occur.

```
Var set = new Set([1,2,1,3,2,2,4])
    set = [1,2,3,4]
Var set = new Set("foooood")
    set = ['f','o','d']
```

add() → add specified value
delete() → delete specified element
clear() → removes all elements

25) isFinite("2009") — true
This function determines if provided value is finite or not. So, if value is NaN, positive or negative infinity, it returns false else true.

26)
```
var arr = [1,2,3]
var dup = [1,2,3]
console.log (arr === dup)   // false, because when
```
any object or array is created, it creates a new instance with same or different internals.
```
console.log (arr === arr)   // true
```
Because, here we are pointing to exact same array or same memory location.

Now,

→ a
is
we
→
poin

27) S
Set
cha

We
actions
whole

Note:-
al
set,
entries

of any type,
nces. A value

2, 4)]

d value is finite
, positive or negative
rue.

lse, because when
creates a new
internals.
rue
exact same array

Var a = new car()
Var b = new car()
Var c = a;

→ console.log(a === b) // false, because new object is created every time we call the constructor even if we call same constructor.

→ console.log(a === c) // true, because here c is pointing to same reference of car() as that of a.

27) Set() ~~and~~ ~~filter()~~.

Set method is used to create a set of unique characters of the string/array.

new Set("Kavya") :- {'K', 'a', 'v', Y'}
new Set([1,2,3,1,2]) :- {1, 2, 3}

We can use multiple set of methods for performing actions on set :- add(), has(), delete(), clear() {clears whole set}, forEach(), etc.

<u>Note</u>:- There is a function called values() which returns all values of set. And since, there are no keys in set, keys() method also returns values. Similarly, entries() method returns [value, value] instead of [key, value]

28) What are maps().

Maps are similar to object having key-value pairs but maps can have any datatype as a key, even an object can be an key for map.

```
const map = new Map()
map.set("1", "value")

typeOf map ⇒ object
map instance of Object ⇒ true
obj instance of Map ⇒ false

map.set(1, "a") ⇒ {1 ⇒ 'a'}
obj[1] = 'a' ⇒ {'1' ⇒ 'a'}
```

Because obj supports string key or symbols.

29) Cookies:

Cookies are small data sent from website and stored on computer by a web browser which can be used to remember information for later use.

```
document.cookie = "key1 = value; expires = date";
```

Document.cookie string keeps a list of name-value pairs seperated by semicolons, where name is the name of cookie. By default, the cookies are deleted on browser close but we can persist it for long with a expiry date.

y - value pairs
as a key,
e map.

e

a'}
⇒ 'a'}
symbols.

and stored
n be used
se.

pires = date";

- value pairs
name of cookie.
owser close
ith a expiry

Cookies

Session Cookies

These are destroyed on browser close.

Persistent Cookies

These ~~are~~ ~~st~~ survive the browser close and can be used by website when you re-open.
Eg:- Cart Items.

document.cookie = "first = Kavya; last = Jain; expires = ...; path = /";

To read cookie :-

```
var cookies = document.cookie.split(';');
```

We mention path because some browser won't allow the deletion of cookies unless there is a clear path that tells which cookie to delete.

20) Arrow function.

→ It allows implicit return i.e. omission of curly braces and creating one-line return operations in methods.

→ Arrow fn. use lexical this which means for any function having this, if its an arrow function then this becomes lexical which means it will point to its parent object whereas without arrow fn., this will be a global (window) object.

```
x = {
  name: Kavya
  show: function(){
    setTimeout (consle(this.name),100)
  }
}
// Undefined
this = window {}
```

```
x = {
  name: Kavya
  show: f(){
    setTimeout (()=> {
      console(this.name)}
  }
}
// Kavya.
```

31)
```
const x=10;
function f(){
    console.log(x)
    const x=20;
}
f();          // x is not initialized.
```

Although, we can access global variables inside function but if we create an inner variable with same name, then outer variable scope is lost.

One thing, if inner variable is const or let then output will be (not initialized) because these don't hoist. Else it will be undefined.

32) Join() → converts array of element into string seperated by a comma or a specified seperator
[1,2,3] → 1,2,3
['abc','xyz'] → abc,xyz
['a','b','c','x'] → a,b,c,x.

33)
```
var x=1;
function f(){
    x=2;
    return x;
}
console.log(x+f());      // 1+2=3
```

34)

35)

This num but refere So this also

lized.

bles inside function
riable with same
lost.

const or let then
because these don't
.

ment into string
eperated by a comma
or a specified seperator

3

34) let set = [1, 2, 3, 4, 5, 6];
let [, , 4] = set;
console.log(4) // 3

9 refers to index value of set. [, , 4] = set[2].
[, , , , 4] = set[4]

35) const x = [1, 2, 3];
const y = x;
x.push(4);
console.log(y) // [1, 2, 3, 4]

This happens because in JS the primitive datatypes like number, string are assigned using value by copy but objects or arrays (non-primitive) are passed by reference.

So, x is pointing to a memory location where this object is created. And when we use y = x, y also points to same memory location.

So, change in x = change in y.

var ~~const~~ x = [a, b, c] #A2306
y = x; x.push(d) #2306
x = 10;
console.log(y, x) #2306 and new address.
↓
[a, b, c, d] 10

36) What is $ (dollar identifier) in JS?

$ in JS simply means that it identifies an object in same way that a name or variable does. Any variable, function, objects, events can be identified using $. In JS, it is not treated as a special character.

```
var $ = 10;
function $(x) { ... }
```

37)
```
let data = 'javascript';
data[0] = 'n';
console.log(data)        // javascript.
```

JS strings are immutable. We cannot directly change value using indexing.
We can use substr() to manually change the values.

```
a = substr(0,1)     // j
b = substr(2)       // ava
return a + replacedChar + b;
```

Or we can use splice() in same way.

38) Diff. b/w splice() and substr().
The main functionality of both are same but if we write ~~smaller~~ greater value as first param and smaller as second, then substr() swaps the arguments and slice() returns empty string. If both values are NaN or negative, substr() treats both of them as 0 but slice() treats first as 0 and counts ~~from~~ backwards indexes from end of string.

text=
subst
substr
slice
slice (

39) what

Note:- W
To

This is

We can

ypes an object
t does. Any
identified using
special character.

directly change

change the values.

way.

same but
param and
swaps the
string. If
() treats both
as 0 and counts
of string.

text = "Mozilla";

substr(-5, 2)); // Mo

substr(-5, -2); //

slice(-5, 2); //

slice(-5, -2); // zil

39) what is type of variable formed using spread operator?

const string = "abcdef";

const [a, b, ...c] = string;

a → a

b → b

c → ['c', 'd', 'e', 'f']; → object.

(typeof([])) - object

Note:- We can use spread operator on string or array.
To iterate over an object, we write:-

const object = {'a':'a', 'b':'b', 'c':'c', 'd':'d'}

const {first, second, ...rest} = object

↓ ✓ ↓

undefined {'a':'a', 'b':'b' .... }

This is destructuring of object.

const {a, b, ...rest} = object

↓ ↓ ↓

a b {'c':'c', 'd':'d'}

We can use , to skip values.

const [, , rest] = 'bcde' → rest ⇒ 'de'

40)
```
const obj = {
    msg : 'Hello' ;
    logMsg () {
        console.log (this.msg);
    }
}
setTimeout ( obj.logMsg , 1000);    // Undefined
```

By default, normal function refers to object scope variable. Which means 'Hello' should be printed. But, we can see undefined. Because, when setTimeout calls obj.logMsg as a callback, it invokes logMsg as a function and not as a method. And for a regular function, this equals global Object which will be window. And msg variable for window is not defined.

41) Diff b/w function and method.

A method is set of instructions associated with an object. A function is set of instruction or procedures which are independent of any object.

```
obj.f()    // Method        f()    // function
```

42)
```
const object = {
    msg: 'Hello World'
}
function getMsg () {
    console.log (this.msg)
}
```

How to

for thi

for thi

43)

This will
regular
global o

Note:-

February 2018
Mon 5 12 19 26
Tue 6 13 20 27
Wed 7 14 21 28
Thu 1 8 15 22
Fri 2 9 16 23
Sat 3 10 17 24
Sun 4 11 18 25
March 2018
5 12 19 26 Mon
6 13 20 27 Tue
7 14 21 28 Wed
1 8 15 22 29 Thu
2 9 16 23 30 Fri
3 10 17 24 31 Sat
4 11 18 25 Sun

How to call getMsg() that it prints 'Hello World'.

for this, we need to bind object with getMsg().
for this purpose, we use call, apply, bind.

```
getMsg.call(object)
getMsg.apply(object)
const log = getMsg.bind(object)
log()
getMsg.bind(object)()
```

43)

```
var a=4;
function get(){
    console.log(this.a)
}
const obj = {
    var a = 5;
    method (callback){
        get()
    }
}
obj.method (callback, 1, 2);
```

This will print 4 (not 5), because we are making regular function call for get() which represents global object which is 4.

Note:- Global this ⇒ arrow and regular function call
Block this ⇒ method call

// Undefined

object scope variable.
inted. But, we
setTimeout calls
okes logMsg as
d. And for a
lobal Object which
variable for window

associated with an
unction or procedures
ject.

// function

44) const obj = {
msg : 'Hello world'
get : function() { console... }
}
let output = obj.get(); // Undefined.
output();

Because obj.get is method call but when we assign it in output, it and call output(), it becomes global call. So, undefined

45) foo(); let foo = function() {..} // Reference error

foo(); var foo = function() {...} // Type error.

first thing, function expressions are not hoisted. So, none of them works. Now, for let, hoisting is not done. So, foo is not even declared when it is accessed. Thus, reference error. for var, hoisting is done. So, foo is declared but nothing is initialized. So, we can't access it as a function. Thus, Type error.

46) Async and await.
Async functions runs asynchronously using an implicit Promise to return its result. It implies that a Promise will be returned, if not manually, JS automatically wraps the result in resolved state. So,

ndefined.

t when we assign
output(), it becomes

3 // Reference error

3 // Type error.

are not hoisted.
ow, for let,
is not even
us, reference error.
foo is declared
, we can't access
error.

usly using an
result. It
be returned, if
wraps the result

```
const test = async() => {
    const value = 2+2;
    return value;
}
test();  →  Promise {4}
```

Now, if we want to make our async function to wait until the promise is resolved then we use await.

```
const test = async() => {
    const value = await 2+2;
    return value;
}
test();  →  Promise {<pending>}
//Pending until resolved or catch error.

test.then(response => ...)   // 4
```

Note:- Here, we will not see much difference but when we have any API calls, etc. then value will be undefined until API calls is completed. And in that case, we want our control to wait on that line of code until it is resolved internally else value will be returned as undefined.