Javascript Interview Prep

1) Type of Null :- object
Undefined :- Undefined
[] :- object

2) Ways to create object in JS.

```
new object();
Object.create(null);
var obj = {};
var obj = new ClassName();
```

3) first class and first order function.

first class fnctn means when functions are treated like a variable. fnctn can be passed as an argument or can be returned by function and can be assigned as a value to any variable.

```
const handler = () => { }
document.addEvent('click', handler);
```

first order a fnctn that doesn't accept another function as argument nor returns a function as return value.

```
const firstOrder = () => console.log();
```

4) What is unary function?
A function that takes exactly one argument.

5) Meaning of ; after for statement.

```
for ( i=0; i< 5; i++) ; {
        console.log(i)
}
```

Here, console will be ignored as for runs and iterates 4 times over a null statement.
; indicates a null statement.

6) Diff b/w const and freeze.

const applies to variables, creates immutable bindings.

```
const p = {
    name : "Kavya"
}
p = something else // error.
```

freeze works on values. It makes object immutable.

```
p.name = something else //error.
```

7) Undefined and not defined.
A variable declared but no value defined is undefined.
A variable neither declared nor defined is not defined.

gcoment.

urns and
ement.

utable bindings.

ict immutable.

error.

e defined

period is

8) Diff. b/w forEach and map.

forEach iterates over a list, applies some operation and does not return anything.

map iterates over a list, transforms members of list and returns a new array accordingly.

```
a = [ .... ]
b = a.forEach()
console.log(b)      // undefined
c = a.map()
console.log(c)      // a new array.
```

9) Javascript is a pass by value lang. objects are values and member function of objects are values.

10) Diff b/w map and filter.
filter takes all objects and formats new list that contains elements which satisfy specific condition.

Map returns an array with exact same size as input after performing some action.

P.T.O.

```
var arr1 = [1, 2, 3, 4, 5];

var arr2 = arr1.map((item) => {
                return item*2
            });

var arr3 = arr1.filter((item) => {
                return item>3
            });
```

11) Reduce function :-

Returns a value from calculation of preceeding element. The final result is a single value of all elements.

```
var val = arr1.reduce((acc, store) => {
                acc + store
            }, initial Value);
```

12) find function :-

Searches for a true value and immediately stops. If no match then returns undefined.

```
console.log(arr1.find(e => e%2 ==0));
```

13) for..in vs for..of.

for..in is used to iterate over objects, array, string to return their properties. Applies to enumerable properties.

for array:- returns index (same for string).
object:- property name

for..of is used for iterating over iterable collection. These are objects that have [symbol.iterator] property. Applies to iterable collection.

for array:- returns elements
object:- Doesn't work as they are not iterable.
String:- Returns characters.

14) How to stop forEach loop.
You cannot exit from forEach loop other than by throwing an exception.

15) find missing number in an array.

```
var arr = [1 - - - 100]
for (var i=1; i<100; i++){
    if( a.indexOf(i) == -1){
        console.log(i);   // Missing item.
    }
```

16) Promise.all and Promise.race-

Race is settled as soon as any of the promise is settled, either resolved or rejected. It does not wait for all promises to settle. Eg:- Stockholder site. Only one will be enough to fulfil your order.
All is settled when all are resolved or exits early when any 1 is rejected.
Eg. User logs in fb, etc., all promises are vital for user homepage. If any one is rejected, it redirects to login page.

Note:- for promise.all, if all are resolved then the output array will be in same order as order of input promises irrespective of time.

```
setTimeout( resolve, 444, 'a');
            "        333, 'b');
            "        111, 'c');
            "        222, 'd');
```

output:- [a, b, c, d];

17).

18)

19)

17). arr.sum() → create a function.

```
Var arr = {
    num1 : 1,
    num2 : 2,
    sum : function(){
        return this.num1 + this.num2;
    }
}
console.log(arr.sum()) → 3.
```

18) setTimeout(fn, 0);

This will execute as soon as the stack is empty, not immediately.

19)

```
var obj = {
    title : 'test'
    func : function(){
        setTimeout( function(){
            console.log( this.title);
        })
```

This will give undefined bcz setTimeout is a fnctn of window object. So, it is called with global object which is not defined. So, returns undefined.
To point it to test, use :-

setTimeout({}.bind(this), 500);

20) Cookies.

Cookies are text files with small amount of data. They are basically used to identify users and improve browsing.
Server creates a cookie with an ID, when cookies are exchanged b/w computer and server, server reads the ID and knows information to show you.

21) Cache Vs cookies

Cache is used to store online page resources for better loading on long run whereas cookies stores user choices to trace user preferences. Cookies are read on server-side whereas other storages (local, session) are read on client side.

22) DRY Principle

Don't Repeat Yourself
Every piece of knowledge must have a single, unambigious representation within a system. Create small reusable units.

23) Javascript principle

SOLID. principle.

24) Hoisting

console

S :-
O :-
L :-
I :-
D :-

let

var

const

Cannot a

Var decl

Undefined

func

func

// f

let ~~var~~

fun

The fun2

Variable.

before

ount of data-
sers and

When cookies
server, server
nation to

resources
whereas
trace user
er-side
sion) are

gle,
tem. Create

S :- Single Responsibility principle
O :- open-closed principle
L :- Liskov Substitution principle
I :- Interface segregation principle
D :- Dependancy inversion principle.

24) Hoisting.

console.log(x) ~~// Undefined~~ // Reference Error
(y) ~~// 20~~ // Undefined
(z) // Reference Error.

let x = 10;
var y = 20;
const z = 30;

Cannot access let and const before initialization/declar.
Var declaration hoists but not initialization so Undefined.

func1(); → works fine
func2(); → Reference error.
// func1 → normal function
let ~~var~~ func2 = function() { }
func2(); → Works fine
The fun2 is a function expression being assigned to variable. This cannot work as a function ~~until~~ before declaration.

25) Arrow function this.

This inside an arrow function will refer to global scope variable and not inside function.

```
var x = 5;
let obj = {
    x: 2
    getX: function (){
        console(x) (this.x)
    }
    ~~getX: function~~
    arrowX: () => {
        console(x) (this.x)
    }
}
obj.getX()          // 2
obj.arrowX()        // 5
obj.call({x:10})    // 10. for getX
                    5 for arrowX.
```

Also, when we have to use setTimeout or any other global methods, then by default, they point to global object. Using arrow, we can make them point to current object using lexical scoping of arrow fn.

lexical scope means scope is taken from where function is called.

```
f(){
setTimeout(){
    f(){ console(x)
}
}
```

↓

gives undefined

↓

```
setTimeout(){
    () => { console(x)
}
}
```

↓

gives 2.

Note:- Here x inside object is block scope but what if we have one x inside getX function.

```
getX: function (){
    const x = 20;
    console();      // 20 or 2 ?
```

Here, again ~~20~~ 2 will be printed because this refers to the instance of invocation which is obj. And x in obj is 2.

26) Call in

Collectio

list ~~sum~~

Va

Va

function

fu

3

ret

3

// this

27) What is

Loadash provides objects an use dir

Some eg:-

Array:- index of

- slice ,

Collection:- - f

Date:- - now

Others are:-

will refer
not inside

we have to
eout or any other
thods, then by
, they point to
otal object. Using
rrow, we can
them point to
nt object using
lexical scoping
of arrow fn.
scope means scope
n from where
is called.

getX
arrowX.

but what
unction.

0 or 2 ?
use this
ich is obj.

26) Call inner function from outside.

Collection list is a fnctn that returns a fnctn list ~~sum~~ that is accessible from outside.

```
Var list = collectionlist();   // returns fnctn list
Var sum = list(n);             // returns some sum.
```

```
function collectionlist() {
    function sum(n) {
        return n;
    }
    return sum;
}
// this will show error.
```

```
function collectionlist() {
    Var obj = {
        list: function(n) {
            return n;
        }
    }
    return obj.list;
```

27) What is loadash?

Loadash is a JS library that works on of _. It provides various in-built functions for collections, arrays, objects and other utility methods that we can use directly instead of writing from scratch.

Some eg:-

Array:- _indexOf, _fill, _join, _concat, _pull, _nth, _reverse, _slice, etc.

Collection:- _filter, _countBy, _every, _find, _forEach, _includes, etc

Date:- _now

Others are:- Math, function, Number, object, String ..etc.

28) How to get max and min value in ES6. for integers.

Number. MAX_SAFE_INTEGER → 9007199254740991

Number. MIN_SAFE_INTEGER → -9007199254740991

for floating point:-

Number. MAX_VALUE → 1.798e+308

Number. MIN_VALUE → 5e-324.

29) Is 24.00 integer ?

Because of precision limit of ECMAscript floating point no. encoding.

Number. isInteger(24.00) → True.

30) Null is a special keyword and undefined is an identifier. So, in non-strict mode, we can create a variable named undefined, although its bad idea.

31) what does void a returns?

void when used with any expression returns undefined.

void a = void function() = undefined.

It is similar to:-

```
function a() {
    return;
}
```

32) How d

So,

Note:-

This hac

Numl

33) Different

1)

2)

3)

(-0 a

a

But

32) How do you test NaN?

NaN is never equal to itself. (only value in JS).

NaN !== NaN.

So, to test any variable as NaN,

a === NaN // false

isNaN(a) // true.

Note :- If a = 2/"foo" which is NaN,

typeOf(a) // Number.

This had some flaws which got fixed in ES6 as Number.isNaN(). Always use this.

33) Different special types of Number :-

1) NaN
2) Infinite (- and +)
3) Zero (- and +)

(-0 are in some case represented as 0.)

a = -0

a.toString() // "0"

a + "" // "0"

JSON.stringfy(a) // "0"

But console reveals true form,

console.log(a) // -0.

a=0, b=0/-3

a==b // true

0 == -0 // true

a===b // true

0===-0 // true

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

March 2018

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

April 2018

34) So, if everything is deceptive, how to test if 0 is negative without console?

```
function isNegative(n)
    n = Number(n)
    return (n===0) && (1/n===-Infinity)

isNegative(-0)   // true
           (0/-3) // true
           (0)    // false
```

To deal with such deceptions, ES6 has new polyfill Object.is().

```
Object.is(0,-0)   // false
          (a,0)   // false  (a=-0)
          (a,NaN) // true   (a=3/"and").
```

35) Pass by value & reference in JS.

In JS, there no such thing as pointer. So, there is no reference from one variable to other. Infact, type of value defines if it will be assigned by value or reference.

Value ⇒ simple values like string, no, bool, null, undefined, symbol (scalar primitive)

Reference ⇒ object, arrays, functions (shared value)

35) func

for

x = [1,

Because, w
a reference
x becom
So, when
value is

o to test if 0 is

(1/n ===-Infinity)

has new polyfill

(a=-0)
(a=3 | "and").

4. So, there is
. Infact,
assigned by

, no, bool,
(scalar primitive)
s (shared value)

So,

```
a=2
b=a
a++
a   //3
b   // still gives 2. (value)

a= [1,2]
b= a
a.push(3)
a   // [1,2,3]
b   // [1,2,3]  (reference).
```

35)

```
function foo(x){
    x.push(4)
    print(x)}
foo  a=[1,2,3]
     foo(a)
     print(a).
```

x = [1,2,3,4] and a = [1,2,3,4]

Because, when we pass a as argument, a is a reference to compound value (shared value). So, x becomes another reference to that same value. So, when x modifies that value, the shared value is being modified.

Now, x = [4,5,6] (changed reference).
a still is [1,2,3,4].

To, make a as [4,5,6]

x.length = 0
x.push (4,5,6) } reference is not changed and a and x both still points to same shared value.

So, Now:-

x // [4,5,6]
a // [4,5,6].

36) How can we pass compound values without reference?
If at any case, we want to pass compound val as an argument but without creating shared reference, we can do that by creating a manual copy of that value which can be done using slice.

a = [1,2,3]
foo(a.slice()) — This does not pass reference of a and instead new shallow copy.

foo (x){
x.push (4)
}

a // [1,2,3]
x // [1,2,3,4].

37) How to pass scalar primitives with reference and not value.
Create a compound object of that value.
b = 2 // scalar primitive
Instead :- var obj = {
b = 2;
}

Note :-

So,

Although

So, for

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

Note:- We may think to create compound value using Number object but that does not work.

```
a = 2
b = new Number(a)        // [Number: 2]
foo(b)                   // passes shared reference of b
                         instead of value.

function foo(x){
    x = x+1              // removes the reference and
                         extracts scalar primitive.
x    // 3
b    // 2
```

So,

Although, we can add new properties.

```
function foo(x){
    x.c = 3
}
x    // {[Number: 2] c: 3}
b    // "
```

So, for simple values to pass as reference, use simple objects.

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

38) Type Casting and coercion.

when one datatype is converted to other explicitly, it is type casting and when implicitly, its coercion.
Coercion always result in simple values.

Type casting — statically typed — compile time
Type Coercion — dynamically typed — run time.

Although in JS, we have:-

explicit Coercion (Intentional type conversion)	Implicit Coercion (as obvious side effects)
b = String(a)	b = a + "";,

39) Unary operators for coercion.

String to number :- a = "1"
b = +a; // 1
c = 5+ +a // 6

Date to number :- d = new Date(----)
+d // 140------ (number).

Earlier, it was used to get timestamp.
var timestamp = +new Date();
(Now we use date.now() or date.getTime()).

40) 0 | x, where x = -0, NaN, ∞ results in 0 because | (OR) is a bitwise operator.
And a bitwise only deals with 32-bit numbers and (0) all values x are not 32-bit.

41) Is fa

Yes,

Before ES
decidi
So,

Why?
Because
for
base.

So,
par

~ (tilt bitwise) = −(x+1)

44) Is parse and cource different?

Yes,

a = "42px"

parseInt(a) = 42 (stops at invalid p)

Number(a) = NaN (fails for unacceptable numeric value).

Before ES5, parsInt used a second argument for deciding the base of parsing. If no

So,

parseInt(1/0, 19) → 18 (not error or ∞)

Why?

Because, 1/0 = ∞ and JS chooses "Infinity". for parsing it takes second arg, 19 as base.

Silly base-19 of I = 18
of n = Invalid

So, as for 42 px (fails at invalid p).
parseInt(1/0, 19) fails at invalid n and returns 18.

42) $\boxed{a = "2"}$ (string)

b = a - 0 = 2 (Number)

b = a * 0 = 0 "

b = a / 0 = Infinity "

b = a + 0 = 20 (String)

$\boxed{a = [2]}$ (Object)

b = a - 0 = 2 (Number)

$\boxed{a = ["2"]}$ (Object)

b = a - 0 = 2 (Number)

$\boxed{a = true}$ (Boolean)

b = a - 0 (1-0) = 1 (Number)

b = a + 0 (1+0) = 1 "

$\boxed{a = "2"}$ (String)

b = !!a (True) (Boolean)

(!a gives !(True) - So, to get correct value, we do !!)

$\boxed{a = null}$

b = !!a (false)

$\boxed{a = true}$ (Boolean)

b = a + "" = true (String)

b = a - "" = 1 (Number)

(1 - 0) = 1

a = "ta" (String)

b = a - "" = NaN (Number)

43) Is

44)

This

declare

If:-

If we

in top-

Note:- If

done

43) Is bool of [] and {} true or false?
It is true
Boolean([]/{}) — True
Boolean(-0/NaN/0) — false.

44)
```
function foo(){
    a=1;          (not declared formally)
}
foo();
a;                (auto global scope Variable)
```

This will not throw error but we should declare variable to belong to individual blocks.

eg:-
```
a;                // a not defined error.
foo();
```

If we set a variable before it been declared, ~~an~~ automatic you'll end up creating a variable in top-level.

Note:- If we use strict mode, this will not be done and show reference error.

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

45) Tokens ⇒ Meaningful chunks of a string

Var a=2;

Var, a, =, 2, ;

This is called tokenizing.

Now if we put tokenizer to invoke stateful parsing rules to figure out whether a should be a distinct token or part of another is called lexing.

46) What is ~~ast~~ abstract syntax tree?

AST is a tree of nested elements which collectively represents the grammatical structure of program.

for Var a=2; we have a top-node called Variable declaration

↓

a child node of Identifier(a) and another child node of AssignmentExpression

↓

a child of Assignment called NumericLiteral(2)

47) How many steps compiler have?

1) Tokenizing / Lexing
2) Parsing – (Parsing stream of token to AST)
3) Code Generation

After above 2 are done, code generation comes to picture. Turning AST to executable code. for AST of Var a=2; , turn it into set of machine instruction to create a variable called (a)

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

including reserve
of 2 into a.

Note:- These are
JS engine
3 steps. But
in JS and
languages, so
can just say
but every
execution.

48) Output of

for
var foo = funct
...
}

Now, function
expression. S
no value
not referenc
Now, we
undefined v
or

And bar(1);,
name identifi

stateful
a should
nother is

ch
ructure of

-node called

and another
xpression

literal (2)

AST)

to
for
of
alled (a)

including reserved memory, etc and then store value of 2 into a.

Note:- These are basic compiling steps for languages. JS engine is more complex than just these 3 steps. But all the steps happen in mere seconds in JS and not in a build step like other languages, so that is beyond our discussion. We can just say, JS is dynamic interpreted lang but every code is JS is also compiled before execution.

48) Output of :-

```
foo();   bar();
var foo = function bar(){
    ....
}
```

Now, function declaration is hoisted but not function expression. So, var foo will go to top with no value returning foo as undefined and not reference error.
Now, we are attempting to call foo i.e. an undefined value. So,
output will be Type Error.
And bar(), will give reference Error because the name identifier is not available in enclosing scope.

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

March 2018

49) Output of :-

```
console.log(a)
var a=2
```

Variable declaration is hoisted but assignment is not. So, a is declared but not assigned.
So, it becomes :-

```
var a ;
console.log(a)   // Undefined.
a=2
```

for :-

```
a=2 ;
console.log(a)
var a ;
```

It becomes :-

```
var a;
a=2;
console.log(a)   // 2.
```

50) What are generator functions.

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

April 2018

51) fun

3

This wi
Because
ran
searches
Now if
need

Now di

Use
have
close
function
have

// Pa