1. How many types can you create OBJECT.

**a. Using the Object() constructor:**

var d = new Object();

This is the simplest way to create an empty object. I believe it is now discouraged.

**b. Using Object.create() method:** Object.create(null); sets the prototype of a as null where as

var a = Object.create(null);

This method creates a new object extending the prototype object passed as a parameter.

c. var b = {}; sets the prototype of a as the Object object

URL: <https://coderwall.com/p/p5cf5w/different-ways-of-creating-an-object-in-javascript>

2. Difference between object.property and object[‘property’]…

URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects>

3. var sumObj = {

a: 1,

b: '2',

c: [3,4,5],

d: 4,

e: 'name',

}. Output: Sum = 5

var sum=0;

for(var key in sumObj){

if(typeof sumObj[key] == "number"){

sum+= sumObj[key];

}

}

4. Difference between fetch and axios call.

Both fetch and axios used for HTTP Requests.

If you use .fetch() there is a two-step process when handing JSON data. The first is to make the actual request and then the second is to call the .json() method on the response.

const url = '[https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF'](https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF%27)

fetch(url).then(response => response.json()).then(data => console.log(data));

Axios: So by using axios you can cut out the middle step of passing the results of the http request to the .json() method. Axios just returns the data object you would expect.

const url = '[https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF'](https://api.spotify.com/v1/artists/0OdUWJ0sBjDrqHygGUXeCF%27)

axios.get(url).then(response => console.log(response));

Fetch: if given url is wrong too, the ‘.then’ block will be executed and then the catch block get the bad request error.

Axios: it directly catches the error in catch block.

Fetch: is not supported in IE and Safari.

Axios: is supported in all browsers.

URL: <https://medium.com/@thejasonfile/fetch-vs-axios-js-for-making-http-requests-2b261cdd3af5>

<https://www.javascriptstuff.com/ajax-libraries/>

­­­­

5. How to iterate through the object?

var p = {

"p1": "value1",

"p2": "value2",

"p3": "value3"

};

for (var key in p) {

if (p.hasOwnProperty(key)) {

console.log(key + " -> " + p[key]);

}

}

each object in javascript (actually a key-value pair) has a property called \_\_proto\_\_ or prototype. This property has a reference to its parent object. An object automatically inherits property from its parent. This is the reason of using hasOwnProperty, which signifies that we're interested in objects own property and not its parent ones.

6. How to convert object into array?

Object.entries() returns an array whose elements are arrays corresponding to the enumerable property [key, value] pairs found directly upon object.

const obj = { foo: 'bar', baz: 42 };

console.log(Object.entries(obj)); // [ ['foo', 'bar'], ['baz', 42] ]

// array like object with random key ordering

const anObj = { 100: 'a', 2: 'b', 7: 'c' };

console.log(Object.entries(anObj)); // [ ['2', 'b'], ['7', 'c'], ['100', 'a'] ]

const object1 = { foo: 'bar', baz: [42,3,4] };

console.log(Object.entries(object1)); // [["foo", "bar"], ["baz", [42, 3, 4]]]

// iterate through key-value gracefully

const obj = { a: 5, b: 7, c: 9 };

for (const [key, value] of Object.entries(obj)) {

console.log(`${key} ${value}`); // "a 5", "b 7", "c 9"

}

// Or, using array extras

Object.entries(obj).forEach(([key, value]) => {

console.log(`${key} ${value}`); // "a 5", "b 7", "c 9"

});

7. const object1 = { foo: 'bar', baz: 42 };

output: [‘bar’, 42];

var values = [];

for(var key in object1){

value.push(object1[key]); output: [‘bar’, 42];

}

8. valueOf() : The **valueOf()** method returns the primitive value of the specified object.

function MyNumberType(n) {

this.number = n;

}

MyNumberType.prototype.valueOf = function() {

return this.number;

};

const object1 = new MyNumberType(4);

console.log(object1 + 3);

// expected output: 7

// console.log(object1 == 4); true

9. sum(2)(3). Output: 5

Closure concept: function sumAll(x){

var innerAdd = function(y){

return x+y;

}

return innerAdd;

}

var aa = sumAll(2);

console.log(aa(4));

10. var aa ={

a: {

b: {

c: 3

}

}

}

Var bb = aa;

If I change ‘c’ value it should not effect aa object.

Ans: you cannot use object.assign() here. You should do deepClone.

var bb = JSON.parse(JSON.stringify(aa)); // it’s a tricky code for deep clone.

bb.a.b.c = 5;

console.log(bb.a.b.c);

console.log(aa.a.b.c);

URL: explains about the advantage and disadvantage of […spread, object.assign(), and JSON.parse(JSON.stringify)].

<https://scotch.io/bar-talk/copying-objects-in-javascript>

ANS 2:

function deepClone (obj) {

var clone = {}

for (var key in obj) {

if (obj.hasOwnProperty(key)) {

var value = obj[key]

switch (Object.prototype.toString.call(value)) {

case '[object Object]': clone[key] = deepClone(obj[key]); break

case '[object Array]' : clone[key] = value.slice(0); break

default : clone[key] = value

}

}

}

return clone

}

let obj = {

a: 'a',

b: {

c: 'c',

d: 'd',

e: {

d: 'd',

f: 'f'

}

},

}

let objB= deepClone(obj);

console.log(objB);

objB.b.c ='test';

objB.b.e.d ='test';

console.log(objB.b.e.d);

console.log(obj.b.e.d);

11. var arr1 = [1,3,4];

Var arr2 = [5,6,7];

Output: arr3 = [1,3,4,5,6,7];

Ans: var arr3 = […arr1, …arr2];

12. var arr1 = [1,3,4];

Var arr2 = [5,6,7];

Output: sumAll = [6,9,11];

Ans: 1. var sumAll = arr1.map((num, index) =>{

return num + arr2[index]];

})

1. Array.prototype.SumArray = function(arr) =>{

Var sum = this.map((num, index) =>{.

/\* here the ‘this’ context holds for the arr1

\* hence we can map through the through this as it contains array values in it

\*/

Return num + arr2[index]];

})

Return sum;

}

Var arr3 = arr1. SumArray(arr2);

1. /\* To support the different length var a= [1,2], b=[3,4,5,7] \*/

. function sumArray (a, b){

Let Cc = [];

. for(let i =0; i< Math.Max( a.length, b.length); i++) {

Cc.push( (a[i] || 0) + (b[i] || 0) );

}

Return cc;

}

Var arr3 = sumArray(arr1, arr2);

13. Can you name some HTML5 features/APIs?

Answer:- Here listed some of the HTML 5 features/API  
• Media API  
• Drag and Drop Feature  
• Offline Cache  
• Full screen API  
• Link Prefetching  
• New Input types like : date, datetime, email, number, tel etc…  
• New Form elements like : datalist, keygen , output  
• New Form Attributes like : autocomplete, min and max, placeholder, required etc..

14.  What is a DOCTYPE!?  
The declaration instructi the web browser about what version of HTML the page is written in.  
Before in HTML4 and prior, declaration refers to a specific DTD.  
Whereas HTML 5 does not require a reference to a DTD so simply declared as

15.  What is the difference between alt, title and longdesc attributes of the image tag?

Alt tag specifies an alternate text for an image.  
Title tag specifies offers to display information about the image when hovered.  
Longdesc tag specifies the URL to a document that contains a long description of an image.

16. What are some possible values for the CSS position declaration?  
STATIC, FIXED, RELATIVE, ABSOLUTE, INHERIT

17. What are some possible values for the CSS display declaration?  
INLINE, BLOCK, NONE

18. What is the difference between display: none and visibility:  
The display property specifies if/how an element is displayed, and the visibility property specifies if an element should be visible or hidden.  
display: none hides an element and it will not take up any space. Whereas visibility: hidden hides an element, but it will still take up the same space as before.

19. What are some of the CSS3 pseudo classes/properties you have used?  
Some ex: :nth-child(N) , :last-child, :enabled, :disabled, :only-child etc…

20. what is **Destructuring assignment**

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

var a, b, rest;

[a, b] = [10, 20];

console.log(a); // expected output: 10

console.log(b); // expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);

// expected output: [30,40,50]

**Can also Give Default values:**

A variable can be assigned a default, in the case that the value unpacked from the array is undefined.

var a, b;

[a=5, b=7] = [1];

console.log(a); // 1

console.log(b); // 7

**Example on objects:**

var a, b;

({a, b} = {a: 1, b: 2});

**LINK:** [**https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring\_assignment**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

21. What is Rest Parameters ?

The **rest parameter** syntax allows us to represent an indefinite number of arguments as an array.

function sum(...theArgs) {

return theArgs.reduce((previous, current) => {

return previous + current;

});

}

console.log(sum(1, 2, 3)); // expected output: 6

console.log(sum(1, 2, 3, 4)); // expected output: 10

**LINK:** <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters>

2. function fun1(...theArgs) {

console.log(theArgs.length);

}

fun1(); // 0

fun1(5); // 1

fun1(5, 6, 7); // 3

22. Difference between Arguments and Rest?

There are three main differences between rest parameters and the [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments) object:

* rest parameters are only the ones that haven't been given a separate name (i.e. formally defined in function expression), while the arguments object contains all arguments passed to the function;
* the arguments object is not a real array, while rest parameters are [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array" \o "Array)instances, meaning methods like [sort](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort), [map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map), [forEach](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach" \o "Array forEach method) or [pop](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop) can be applied on it directly;
* the arguments object has additional functionality specific to itself (like the calleeproperty).

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters>

23. add(2,3) //5

add(2)(3) // 5

function add(a, b){

return a && b ? a+b : function(c){return a+c;}

}

function add(a,b){

var ddd = function (b){return a+b;};

if(typeof b =='undefined'){

return ddd;

}else{

return ddd(b);

}

}