How would you check if a number is an integer?

num % 1 === 0;

True : integer

False : float

What will the following code output?

function isInt(num) {

return num % 1 === 0;

}

console.log(isInt(4)); // true

console.log(isInt(12.2)); // false

console.log(isInt(0.3)); // false

Write a function that would allow you to do this.

multiply(5)(6);

function multiply(a) {

return function(b) {

return a \* b;

}

}

multiply(5)(6);

When would you use the bind function?

A good use of the bind function is when you have a particular function that you want to call with a specific this value. You can then use bind to pass a specific object to a function that uses a this reference.

function fullName() {

return "Hello, this is " + this.first + " " + this.last;

}

console.log(fullName()); // => Hello this is undefined undefined

// create a person object and pass its values to the fullName function

var person = {first: "Foo", last: "Bar"};

console.log(fullName.bind(person)()); // => Hello this is Foo Bar

What does "use strict" do?

The "use strict" literal is entered at the top of a JavaScript program or at the top of a function and it helps you write safer JavaScript code by throwing an error if a global variable is created by mistake. For example, the following program will throw an error:

function doSomething(val) {

"use strict";

x = val + 10;

}

It will throw an error because x was not defined and it is being set to some value in the global scope, which isn't allowed with "use strict" The small change below fixes the error being thrown:

function doSomething(val) {

"use strict";

var x = val + 10;

}

Can m and n can accesse out of the function ? no

function doSomething(val) {

var m,n = val;

}

Can m and n can accesse out of the function ? m cannot be access, but n can be access

function doSomething(val) {

var m=n = val;

}

What is the difference between == and ===?

The difference is that == performs implicit type conversion to check if values are equal to each other. So it will convert the values to true or false boolean values and then compare them. Below are some weird results of this implicit type conversion using the double equal operator:

"1" == 1 // true

true == 1 // true

[] == false // true

"" == false // true

undefined == null // true

This can really cause some problems when you're writing JavaScript, so the safe thing to do is use the triple equal operator instead, which checks that the two values are of the same type and does not perform type conversion.

"1" === 1 // false

true === 1 // false

[] === false // false

"" === false // false

undefined === null // false

How would you create a private variable in JavaScript?

To create a private variable in JavaScript that cannot be changed you need to create it as a local variable within a function. Even if the function is executed the variable cannot be accessed outside of the function. For example:

function func() {

var priv = "secret code";

}

console.log(priv); // throws error

To access the variable, a helper function would need to be created that returns the private variable.

function func() {

var priv = "secret code";

return function() {

return priv;

}

}

var getPriv = func();

console.log(getPriv()); // => secret code

What will the following code output?

0.1+0.2 === 0.3

1. **Answer**
2. This will surprisingly output false because of floating point errors in internally representing certain numbers. 0.1 + 0.2 does not nicely come out to 0.3 but instead the result is actually 0.30000000000000004 because the computer cannot internally represent the correct number. One solution to get around this problem is to round the results when doing arithmetic with decimal numbers.

Explain what a callback function is and provide a simple example.

A callback function is a function that is passed to another function as an argument and is executed after some operation has been completed. Below is an example of a simple callback function that logs to the console *after* some operations have been completed.

function modifyArray(arr, callback) {

// do something to arr here

arr.push(100);

// then execute the callback function that was passed

callback();

}

var arr = [1, 2, 3, 4, 5];

modifyArray(arr, function() {

console.log("array has been modified", arr);

});

1. **How many types of functions JS support?**
   * A function in JavaScript can be either named or anonymous.

Example: anonymous

alpha=()=>{ console.log('hello'); }

()=>{ console.log('hello'); }

Named function:

function test(){ console.log('hi'); }

1. **How to define an anonymous function?**
   * An anonymous function can be defined in a similar way as a normal function but it would not have any name.

**Which built-in method calls a function for each element in the array?**

* forEach method calls a function for each element in the array.

Without index: a.forEach(t=> console.log(t))

With index : a.forEach((t,i)=> console.log(i))

**Difference between “var” and “let” Keywords?**

* **Var** was there from the beginning but the let was introduced in ES2015/ES6.  
  **Let** has block scope and “Var” has function scope

**Difference between “var” and “let” Keywords?**

* **Var** was there from the beginning but the let was introduced in ES2015/ES6.  
  **Let** has block scope and “Var” has function scope

1. **Which built-in method sorts the elements of an array?**
   * Sort method sorts the elements of an array.

[11,22,333,1,2,3,1]

[“hi”,’a’,’beta’,’test’,’aman’]

[{id:101,name:’raman’,gender:’male},{id:1,name:’aman’,gender:’male},{id:10,name:’ridhi’,gender:’female}]

1. **Which built-in method reverses the order of the elements of an array?**
   * Reverse method reverses the order of the elements of an array ?? the first becomes the last, and the last becomes the first.
2. **What is SetTimeout()?**
   * When you setTimeout it becomes asynchronous and it has to wait on the stack to get everything got finished
3. **Output of below statements**

<script>

document.write({});

</script>

[object Object]

Apply vs. Call vs. Bind

#### Function.prototype.call()

The method Call invokes the function and allows you to pass in arguments one by one using commas.

let customer1 = { name: 'Leo', email: 'leo@gmail.com' };  
let customer2 = { name: 'Nat', email: 'nat@hotmail.com' };  
  
function greeting(text) {  
 console.log(`${text} ${this.name}`);  
}  
  
greeting.call(customer1, 'Hello'); // Hello Leo  
greeting.call(customer2, 'Hello'); // Hello Nat

#### Function.prototype.apply()

The method Apply invokes the function and allows you to pass in arguments as an array.

let customer1 = { name: 'Leo', email: 'leo@gmail.com' };  
let customer2 = { name: 'Nat', email: 'nat@hotmail.com' };

function greeting(text, text2) {  
 console.log(`${text} ${this.name}, ${text2}`);  
}

greeting.apply(customer1, ['Hello', 'How are you?']); // output Hello Leo, How are you?  
greeting.apply(customer2, ['Hello', 'How are you?']); // output Hello Natm How are you?

#### Function.prototype.bind()

The Bind method returns a new function, allowing you to pass in a this array and any number of arguments. Use it when you want that function to later be called with a certain context like events.

let customer1 = { name: 'Leo', email: 'leo@gmail.com' };  
let customer2 = { name: 'Nat', email: 'nat@hotmail.com' };

function greeting(text) {  
 console.log(`${text} ${this.name}`);  
}

let helloLeo = greeting.bind(customer1);  
let helloNat = greeting.bind(customer2);

helloLeo('Hello'); // Hello Leo  
helloNat('Hello'); // Hello Nat

The Bind implementation would be like this:

Function.prototype.bind = function(context) {  
 var fn = this;  
 return function() {  
 fn.apply(context, arguments);  
 };  
};

**Call** and **Apply** are interchangeable. You can decide whether it’s easier to send in an array or a comma separated list of arguments. **Bind** is different. It always returns a new function.

We can use **Bind**to curry functions like in the example. We can take a simple hello function and turn it into a helloJon or helloKelly. You can use it for events where we don’t know when they’ll be fired but know what context is.

Arrow functions vs bind

**Arrow functions** in class properties are much slower than bound **functions**, and both are much slower than usual **function**. You should only **bind** with .**bind**() or **arrow function** a method if you're going to pass it around

# How to include js file in another js file?

var array1 = ['a', 'b', 'c'];

array1.forEach(function(element) {

console.log(element);

});

you can only include a script file in an HTML page, not in another script file. That said, you can write JavaScript which loads your "included" script into the same page:

var imported = document.createElement('script');

imported.src = '/path/to/imported/script';

document.head.appendChild(imported);

There's a good chance your code depends on your "included" script, however, in which case it may fail because the browser will load the "imported" script asynchronously. Your best bet will be to simply use a third-party library like jQuery or YUI, which solves this problem for you.

// jQuery

$.getScript('/path/to/imported/script.js', function()

{

// script is now loaded and executed.

// put your dependent JS here.

});