**1.**

What will be logged and **why**?

console.log(false == '0'); console.log(false === '0');

Falsy conditions.

First one it does not compare types and therefore is true (false is equal to 0, but in second one do care about the types so it is false.

**2.**

What will be logged and **why**?

console.log(1 + '2' + '2');

console.log(1 + +'2' + '2');

console.log(1 + -'1' + '2');

console.log(+'1' + '1' + '2');

console.log('A' - 'B' + '2');

console.log('A' - 'B' + 2);

* "122"
* "32"
* "02"
* "112"
* "NaN2"
* NaN

**3.**

What will be logged and **why**?

(function (x) { return (function (y) { console.log(x);}) (2);}) (1);

1 – first function that is returns – returns undefined.

**4.**

What's the difference between .call and .apply?

Function a(num1, num2, num3){

Return this+1+num1 +num2 +num3

Var b=2

Var result=a.call(b,10,20,30);

Console.log(result);

.call (uses the piece you want to have the value of ‘this’ as first parameter and arguments after that. Call takes a list of arguments.

**.apply is used the same way but for arrays.**

Var c=[10,20,30]

Var result=a.apply(b,c);

If .apply called with null as first parameter it is because we don’t care about value of this in the code that it is being passed to.

**5.**

Explain the difference between the two:

function Person(){} var person = Person();

function Person(){} var person = new Person();

First one – function call – returns undefined.

Second one – creates a new object – returns an object. Best not to use new ot create new object this way but to use

Var person = Object.create(Person.prototype).

It’s the prototype instance that you can take a copy of.

**6.**

Write the following function:

duplicate([1,2,3,4,5]); // [1,2,3,4,5,1,2,3,4,5]

**function duplicate(array1){**

**return (array1.concat(array1));**

**}**

**console.log(duplicate([1,2,3,4,5]));**

**7.**

Write the following function:

console.log('hello'.repeatify(3)); // hellohellohello

console.log('hello'.repeat(3));

**8.**

What will be logged and **why**?

function test()

{ console.log(a);

console.log(getNumber());

var a = 1;

function getNumber()

{ return 2; }

}

test();

it returns ‘undefined’ followed by 2, as at first console.log it knows about a – but it has not yet been assigned a value, and then it gets number 2 returned from function and placed in console.log.

**9.**

Give an example of a Revealing Module Pattern?

**10.**

var message = { text: 'Hello' }; var welcome = message; welcome.text = message = { text: 'Welcome' };