**1.**

What will be logged and **why**?

console.log(console.log(console.log('console.log')));

Console log

Undefined

Undefined

(as Console.log returns undefined to the first two functions.)

**2.**

Explain:

* Why do we need lexical scopes? Identifier collisions
* Why do we need closures? So you can access identifiers inside other lexical scopes (that can still be hidden in those scopes). Allows values to be remembered. Eg. Timeouts and for loops.
* Why do we need this? So identifier can be referred to at author-time even though its value won’t be known until execution time.
* Why do we need bahaviour delegation? So that not every object needs to have each property defined separately, just refer to prototype chain and get it. It’s a link.

**3.**

What's the potential problem with this check?

typeof car === 'object' Null has typeof object as does array so this check won’t be enough. Can fix this by adding ‘&& n’ on to the check.

If n is null it will operate as a falsey and not execute, if n is [] then will execute.

If looking for array – use Array.isArray(n)

**4.**

What will be logged and **why**?

(function () { var message = info = 'Introduction'; })(); console.log(typeof message !== 'undefined'); // ???

console.log(typeof info !== 'undefined'); // ???

console.log(art) => referenceError (as it has not been defined)

console.log(Typeof(art)) => undefined.

So above,

Var message=info=’Introduction’; is a valid line.

First line is false (as message is undefined in global scope), and undefined !== undefined is false.

Second line is true.

**5.**

What is IIFE good for?

(function () {})(); Getting functions to run straightaway.

Good for calling anonymous functions as they can not be referenced.

**6.**

What will be logged and **why**?

var car = { make: 'Tesla',

model: 'Model X',

whatAreYou: function () {

var that = this;

console.log(that.make);

console.log(this.make);

(function () {

console.log(that.model);

console.log(this.model); })

(); } };

car.whatAreYou();

Last console.log is undefined as the model isnot defined in global scope, therefore returns undefined. There is no binding back to the car object for this to work.

**7.**

What will be logged and **why**?

function getData(data, callback)

{ setTimeout(function () { callback(data); }, 1000); }

getData(10, function (firstNumber)

{ var x = 1 + firstNumber;

getData(30, function (secondNumber)

{ var y = 1 + secondNumber;

getData('Meaning of life: ' + (x + y), function (answer) { console.log(answer); }); }); });

**Meaning of life:42 but why? So that it is asynchronous – so that the first step is not executed until the previous task is finished. Good for if you need something returned from a server before you do the next step.**

**Nesting like this protects that sequence.**

**8.**

Explain the difference between this:

function modelS () { return { make: 'Tesla' }; }

And that:

function modelS () { return { make: 'Tesla' }; }

When JS sees a New Line character – it will put a semi-colon in for you! So the example above with a Line return after the return statement will not go any further.

**9.**

Find 2 ways to make sure that 2 will be logged?

console.log('1' + 1);

console.log(1 + 1);

console.log(‘1’ - -1);

or

console.log(+’1’+1) as when + is a unary operator it converts the string to a number.

Console.log(1 ++1);

Or use parseInt(‘1’ + 1);

**10.**

Write function sum that returns the same answer in both cases:

console.log(sum(5, 10)); // 15

console.log(sum(5)(10)); // 15

function sum(x,y) {

return(x+y)}

function sum(x) {

return sum(