Basics(oops, functional programming)

1) What is the difference between Object Oriented Programming and Functional Programming?
    a) https://www.educba.com/functional-programming-vs-oop/
2) What is the difference between let, const and var keywords?
    a) https://dev.to/sarah_chima/var-let-and-const--whats-the-difference-69e
3) What is hoisting?
    a) Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. Remember that JavaScript only hoists declarations, not initialisation. Let's take a simple example of variable hoisting,
    b) `console.log(message); //output : undefined`
    c) `var message = 'The variable has been hoisted';`
4) What is the difference between slice and splice keywords?
    a) Some of the major difference in a tabular form

Slice	Splice
Doesn't modify the original array(immutable)	Modifies the original array(mutable)
Returns the subset of original array	Returns the deleted elements as array
Used to pick the elements from array	Used to insert or delete elements to/from array

5) What is the difference between Call, Apply and Bind?

The difference between Call, Apply and Bind can be explained with the below examples, Call: The call() method invokes a function with a given this value and arguments provided one by one

```
var employee1 = {firstName: 'John', lastName: 'Rodson'};
var employee2 = {firstName: 'Jimmy', lastName: 'Baily'};
```

```
function invite(greeting1, greeting2) {
  console.log(greeting1 + ' ' + this.firstName + ' ' + this.lastName+ ', '+
greeting2);
}

invite.call(employee1, 'Hello', 'How are you?'); // Hello John Rodson,
How are you?
invite.call(employee2, 'Hello', 'How are you?'); // Hello Jimmy Baily, How
are you?
```

Apply: Invokes the function and allows you to pass in arguments as an array

```
var employee1 = {firstName: 'John', lastName: 'Rodson'};
var employee2 = {firstName: 'Jimmy', lastName: 'Baily'};

function invite(greeting1, greeting2) {
  console.log(greeting1 + ' ' + this.firstName + ' ' + this.lastName+ ', '+
greeting2);
}

invite.apply(employee1, ['Hello', 'How are you?']); // Hello John Rodson,
How are you?
invite.apply(employee2, ['Hello', 'How are you?']); // Hello Jimmy Baily,
How are you?
```

bind: returns a new function, allowing you to pass in an array and any number of arguments

```
var employee1 = {firstName: 'John', lastName: 'Rodson'};
var employee2 = {firstName: 'Jimmy', lastName: 'Baily'};

function invite(greeting1, greeting2) {
  console.log(greeting1 + ' ' + this.firstName + ' ' + this.lastName+ ', '+
greeting2);
}

var inviteEmployee1 = invite.bind(employee1);
var inviteEmployee2 = invite.bind(employee2);
```

```
inviteEmployee1('Hello', 'How are you?'); // Hello John Rodson, How are you?
inviteEmployee2('Hello', 'How are you?'); // Hello Jimmy Baily, How are you?
```

Call and apply are pretty interchangeable. Both execute the current function immediately. You need to decide whether it's easier to send in an array or a comma separated list of arguments. You can remember by treating Call is for comma (separated list) and Apply is for Array. Whereas Bind creates a new function that will have this set to the first parameter passed to bind().

6) Which function would you use to create an array of elements which satisfy a condition from an existing array?

a) filter()
b) some()
c) every()
d) splice()

Ans- filter()

7) What is a strict mode in javascript? How do you declare it ?

a) Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context. This way it prevents certain actions from being taken and throws more exceptions. The literal expression "use strict"; instructs the browser to use the javascript code in the Strict mode.

```
"use strict";
x = 3.14; // This will cause an error because x is not declared
```

8) What is the difference between == and === operators?

a) https://www.c-sharpcorner.com/article/difference-between-and-in-javascript2/

9) Explain the difference between Object.freeze() vs const
https://alligator.io/js/const-vs-obj-freeze/

10) What will the code below output to the console 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);
```

Ans - The above code will output the following to the console:

"122"
"32"
"02"
"112"
"NaN2"
NaN
Here's why…

The fundamental issue here is that JavaScript (ECMAScript) is a loosely typed language and it performs automatic type conversion on values to accommodate the operation being performed. Let's see how this plays out with each of the above examples.

Example 1: 1 + "2" + "2" Output: "122" Explanation: The first operation to be performed in 1 + "2". Since one of the operands ("2") is a string, JavaScript assumes it needs to perform string concatenation and therefore converts the type of 1 to "1", 1 + "2" yields "12". Then, "12" + "2" yields "122".

Example 2: 1 + +"2" + "2" Output: "32" Explanation: Based on order of operations, the first operation to be performed is +"2" (the extra + before the first "2" is treated as a unary operator). Thus, JavaScript converts the type of "2" to numeric and then applies the unary + sign to it (i.e., treats it as a positive number). As a result, the next operation is now 1 + 2 which of course yields 3. But then, we have an operation between a number and a string (i.e., 3 and "2"), so once again JavaScript converts the type of the numeric value to a string and performs string concatenation, yielding "32".

Example 3: 1 + -"1" + "2" Outputs: "02" Explanation: The explanation here is identical to the prior example, except the unary operator is - rather than +. So "1" becomes 1, which then becomes -1 when the - is applied, which is then added to 1 yielding 0, which is then converted to a string and concatenated with the final "2" operand, yielding "02".

Example 4: +"1" + "1" + "2" Outputs: "112" Explanation: Although the first "1" operand is typecast to a numeric value based on the unary + operator that precedes it, it is then immediately converted back to a string when it is concatenated with the second "1" operand, which is then concatenated with the final "2" operand, yielding the string "112".

Example 5: "A" - "B" + "2" Outputs: "NaN2" Explanation: Since the - operator can not be applied to strings, and since neither "A" nor "B" can be converted to numeric values, "A" - "B" yields NaN which is then concatenated with the string "2" to yield “NaN2”.

Example 6: "A" - "B" + 2 Outputs: NaN Explanation: As explained in the previous example, "A" - "B" yields NaN. But any operator applied to NaN with any other numeric operand will still yield NaN.

11) What will be the output of this code?

```
var x = 21;
var girl = function () {
    console.log(x);
    var x = 20;
};
girl ();
```

Ans - Neither 21, nor 20, the result is undefined

It’s because JavaScript initialization is not hoisted.

(Why doesn’t it show the global value of 21? The reason is that when the function is executed, it checks that there’s a local x variable present but doesn’t yet declare it, so it won’t look for global one.)

12) What is the output of the following code? Explain your answer.

```
var a={},
    b={key:'b'},
    c={key:'c'};
a[b]=123;
a[c]=456;
console.log(a[b]);
```

Ans - The output of this code will be 456 (not 123).

The reason for this is as follows: When setting an object property, JavaScript will implicitly stringify the parameter value. In this case, since b and c are both objects, they will both be

converted to "[object Object]". As a result, a[b] anda[c] are both equivalent to a["[object Object]"] and can be used interchangeably. Therefore, setting or referencing a[c] is precisely the same as setting or referencing a[b].

13) What do the following lines output, and why?

```
console.log(1 < 2 < 3);
console.log(3 > 2 > 1);
```

Ans - The first statement returns true which is as expected.

The second returns false because of how the engine works regarding operator associativity for < and >. It compares left to right, so 3 > 2 > 1 JavaScript translates to true > 1. true has value 1, so it then compares 1 > 1, which is false.

14) Write a sum method which will work properly when invoked using either syntax below.

```
console.log(sum(2,3));   // Outputs 5
console.log(sum(2)(3));  // Outputs 5
```

Ans -

```
function sum(x, y) {
  if (y !== undefined) {
    return x + y;
  } else {
    return function(y) { return x + y; };
  }
}
```

Design Patterns in js

15) What is a first order function?

First-order function is a function that doesn't accept other functions as an argument and doesn't return a function as its return value.

```
const firstOrder = () => console.log ('I am a first order function!');
```

16) What is a pure function?

a) https://blog.bitsrc.io/understanding-javascript-mutation-and-pure-functions-7231cc2180d3

17) What is memoization?

Memoization is a programming technique which attempts to increase a function's performance by caching its previously computed results. Each time a memoized function is called, its parameters are used to index the cache. If the data is present, then it can be returned, without executing the entire function. Otherwise the function is executed and then the result is added to the cache. Let's take an example of adding function with memoization,

```
const memoizAddition = () => {
  let cache = {};
 return (value) => {
  if (value in cache) {
   console.log('Fetching from cache');
   return cache[value]; // Here, cache.value cannot be used as
property name starts with the number which is not valid JavaScript
identifier. Hence, can only be accessed using the square bracket
notation.
  }
  else {
   console.log('Calculating result');
   let result = value + 20;
   cache[value] = result;
   return result;
  }
 }
}
// returned function from memoizAddition
const addition = memoizAddition();
console.log(addition(20)); //output: 40 calculated
console.log(addition(20)); //output: 40 cached
```

18) What is the Temporal Dead Zone in ES6?

19) What is CQRS Design Pattern? How is it different from CRUD?

a) https://blog.risingstack.com/cqrs-explained-node-js-at-scale/

20) What is Saga Pattern?

a) https://blog.couchbase.com/saga-pattern-implement-business-transactions-using-microservices-part/

Nodejs Environment (npm and node modules basics)

21) Is Node Js single threaded or multithreaded?

a) Single threaded

22) What is npm? What is the main functionality of npm?

npm is the world's largest repository in software industry.
https://docs.npmjs.com/about-npm/

23) What is package.json file?

package. json is a plain JSON(Java Script Object Notation) text file which contains all metadata information about Node JS Project or application. Every Node JS Package or Module should have this file at root directory to describe its metadata in plain JSON Object format.
https://www.journaldev.com/7553/importance-of-package-json-in-node-js-applications

Nodejs Framework (Expressjs)

24) What is ExpressJS and why is it used? Name some other NodeJS frameworks.

a) https://stackoverflow.com/questions/12616153/what-is-express-js

b) Hapi, Fastify, Meteor

Async (Event Loop) - Promises , callbacks , async await

25) What is a callback hell? How would you solve it?

a) Callback Hell is an anti-pattern with multiple nested callbacks which makes code hard to read and debug when dealing with asynchronous logic. The callback hell looks like below,

```
async1(function(){
   async2(function(){
      async3(function(){
         async4(function(){
            ....
```

```
        });
      });
    });
  });
```

b) Async Await / Promises

26) What is the use of the promise.all(...) function?

a) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

27)
```
let num = 0;
async function increment() {
  num += await 2;
  console.log(num);
}
increment();
num += 1;
console.log(num);
/****
 * What is the resulting output?
 *  2, 3
 *  1, 3
 *  1, 2
 *  2, 1
 */
```

Ans- 1, 2

Compilation, swagger docs (addon)

28) What is Swagger Documentation and why is it used?

a) https://swagger.io/docs/specification/about/

Closures and Scoping

29) What are closures?

A closure is the combination of a function and the lexical environment within which that function was declared. i.e, It is an inner function that has access to the outer or enclosing function's variables. The closure has three scope chains

i. Own scope where variables defined between its curly brackets
ii. Outer function's variables
iii. Global variables Let's take an example of closure concept,

```
a) function Welcome(name){
b)   var greetingInfo = function(message){
c)    console.log(message+' '+name);
d)   }
e) return greetingInfo;
f) }
g) var myFunction = Welcome('John');
h) myFunction('Welcome '); //Output: Welcome John
myFunction('Hello Mr.'); //output: Hello Mr.John
```

As per the above code, the inner function(greetingInfo) has access to the variables in the outer function scope(Welcome) even after outer function has returned.

30) What will the following code output?

```
const arr = [10, 12, 15, 21];
for (var i = 0; i < arr.length; i++) {
  setTimeout(function() {
    console.log('Index: ' + i + ', element: ' + arr[i]);
  }, 3000);
}
```

Ans - https://medium.com/coderbyte/a-tricky-javascript-interview-question-asked-by-google-and-amazon-48d212890703

JS with Ethereum

31) What does the front end use in order to connect to the Ethereum backend (Smart Contracts)?

a) Web3.js

32) How to perform batch transactions in web3js?

a) https://web3js.readthedocs.io/en/v1.2.0/web3-eth.html#batchrequest

33) Write a code to derive the Ethereum address from an Ethereum Public key.

a) https://ethereum.stackexchange.com/questions/3542/how-are-ethereum-addresses-generated

34) What is the ABI used for?

Ans: ABI is a description of the public interface of a contract, which is used by DApps for invoking the contract.

35) Let's say you have a Test.sol file. Deploy this contract and save the contract address in a variable without using truffle.

```
myContract.deploy({
    data: '0x12345...',
    arguments: [123, 'My String']
})
.send({
    from: '0x1234567890123456789012345678901234567891',
    gas: 1500000,
    gasPrice: '30000000000000'
}, function(error, transactionHash){ ... })
.on('error', function(error){ ... })
.on('transactionHash', function(transactionHash){ ... })
.on('receipt', function(receipt){
   console.log(receipt.contractAddress) // contains the new contract address
})
.on('confirmation', function(confirmationNumber, receipt){ ... })
.then(function(newContractInstance){
    console.log(newContractInstance.options.address) // instance with the new
contract address
});
```

Here data is the Bytecode of the contract which is generated by compiling the Test.sol file with solc.

```
var solc = require('solc');

var input = {
  language: 'Solidity',
  sources: {
    'Test.sol': {
```

```
    content: 'contract Test { function f() public { } }'
  }
 },
 settings: {
  outputSelection: {
   '*': {
    '*': ['*']
   }
  }
 }
};

var output = JSON.parse(solc.compile(JSON.stringify(input)));

// `output` here contains the JSON output as specified in the documentation
for (var contractName in output.contracts['test.sol']) {
 console.log(
  contractName +
   ': ' +
   output.contracts['Test.sol'][contractName].evm.bytecode.object
 );
}
```

36) Create an Ethereum wallet address in javascript without using truffle.

a) web3.eth.accounts.create();

37) Is sending one ether like this ".send({ value: 1 })" okay?

a) No, you send 1 wei. Transactions always work with wei.

Persistence/Message bus

38) What is the difference between SQL and noSQL databases? Explain Pros and Cons of each.

a) https://www.guru99.com/sql-vs-nosql.html

39) What is an ORM? Name one SQL ORM and one noSQL ODM in Javascript?

a) https://searchwindevelopment.techtarget.com/definition/object-relational-mapping

b) SQL- Sequelize, NoSQL- Mongoose

40) What do you mean by inner joins in SQL?

a) https://www.w3resource.com/sql/joins/perform-an-inner-join.php

41) What do you mean by outer joins in SQL?

a) https://www.w3resource.com/sql/joins/perform-a-full-outer-join.php

42) Database Locking: What it is, Why it Matters and What to do About it?
https://www.methodsandtools.com/archive/archive.php?id=83

Architecture

43) Difference between Monolith vs Microservices architecture. Which one is better for building a production level application with a lot of process intensive operations?
   a) https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59?gi=907b7c821dd3
   b) Microservices

44) How do microservices communicate to each other?
   a) https://blog.logrocket.com/methods-for-microservice-communication/

45) What is KAFKA?
   a) https://dzone.com/articles/microservices-and-kafka-part-one

46) What is load balancing?
   a) Load balancing is defined as the methodical and efficient distribution of network or application traffic across multiple servers in a server farm. Each load balancer sits between client devices and backend servers, receiving and then distributing incoming requests to any available server capable of fulfilling them.

47) How do you implement load balancers in Nodejs?
   a) Load Balancing Node JS - Techintoo

Event Sourcing

48) What is an Event Sourcing Design Pattern? Explain.
   a) Event sourcing, CQRS, stream processing and Apache Kafka: What's the connection?

DevOps

49) What is Docker used for?
   a) What is Docker?

50) What is Continuous Delivery, Continuous Deployment and Continuous Integration?
   a) Continuous Delivery, Deployment & Integration: 20 Key Differences

GIT basics

51) What is the difference between git pull and git fetch?

a) Git pull command pulls new changes or commits from a particular branch from your central repository and updates your target branch in your local repository.

b) Git fetch is also used for the same purpose but it works in a slightly different way. When you perform a git fetch, it pulls all new commits from the desired branch and stores it in a new branch in your local repository. If you want to reflect these changes in your target branch, git fetch must be followed with a git merge. Your target branch will only be updated after merging the target branch and fetched branch. Just to make it easy for you, remember the equation below:

c) Git pull = git fetch + git merge

52) What is git stash?

a) Often, when you've been working on part of your project, things are in a messy state and you want to switch branches for some time to work on something else. The problem is, you don't want to do a commit of half-done work just so you can get back to this point later. The answer to this issue is Git stash.

b) Stashing takes your working directory, that is, your modified tracked files and staged changes and saves it on a stack of unfinished changes that you can reapply at any time.