FRONT END – PART 2
ASYNCHRONOUS JAVASCRIPT

(Promises)

1. What State will the variable x be in when it gets logged?

```
var x = new Promise(() => {})
console.log(x)
```

a) resolved b) rejected c) pending

(Callback disadvantages)

2. Why do we use promises instead of callbacks?

a) Because it is faster than callback b) Because it is more readable

c) To escape call back hell d) Because it makes the code more maintainable

(setTimeout())

3. Is setTimeout() part of javascript?

a) Yes b) No

(Event loop)

4. What is the main function of event loop?

a) It executes the code of javascript

b) It creates separate threads for asynchronous tasks to run

c) It checks for the call stack and pushes callbacks from the callback queue

d) It sets timer for setTimeout()

(Queues)

5. Which queue are the callbacks of promises are pushed into?

a) Task queue b) Microtask queue

(Async try catch)

6. Consider the following function:

```
async function x() {
  try{
    await Promise.reject("Rejected!!")
  } catch(e) {
    console.log(e)
  }
}
```

What will happen when it gets called?

a) It will give an uncaught error b) “Rejected!!” will get printed

c) It will give an error for using await inside try catch block

ASSIGNMENT

(Promise.all())

7. What will be the result of the following code?

```
const promise1 = new Promise((resolve,reject) => { setTimeout(() =>
{ resolve('a') }) })

const promise2 = new Promise((resolve,reject) => { resolve('b') })

const promise3 = new Promise((resolve,reject) => { setTimeout(() =>
{ resolve('c') },10) })

Promise.all([promise1,promise2,promise3]) .then((msg) =>
{ console.log(msg) })
```

a) a b) b c) c d) ['a','b','c']

(promise vs SetTimeout)

8. What will be logged in the output?

```
setTimeout(() => {
  console.log('a')
})

new Promise((resolve,reject) => {
  setTimeout(() => {
    console.log('b')
  })
})
```

a) a b b) b a

(async await)

9. What will happen when the following code runs?

```
async function x() {
  try{
    await Promise.reject("Rejected!!")
  } catch(e) {
    console.log(e)
  }
}

x()
.catch((msg) => {
  console.log(msg)
})
```

a) It will give an uncaught error

b) “Rejected!!” will get printed once

c) “Rejected!!” will get printed twice

(callbacks)

10. What will be the output of the following code?

```
x((a,b,c) => {
  return a+b+c
},2,3,6)

function x(callback,a,b,c) {
  let str = ""
  setTimeout(() => {
    str += "x"
  });

  str += callback(a,b,c)
  console.log(str)
}
```

(True or false)

11. Javascript is a multithreaded language. True or False

a) True

b) False

(Queue priority)

12. Which queue gets priority from the event loop while pushing the callbacks to the call stack?

a) Task Queue

b) Microtask queue

(Rejection in await)

13. What will happen if the following code runs?

```
function x() {
  Promise.reject('x')
  return Promise.resolve('y')
}

async function log() {
  const val =  await x()
  console.log(val)
}

log()
```

a) It will give a promise rejection error

b) It will print y

c) It will print y and give a promise rejection error

(promise vs SetTimeout 2)

14. What will get logged in the console?

```
const promise = new Promise((resolve,reject) => {
  resolve('a')
```

```
})

setTimeout(() => {
  console.log('b')
})

promise.then((msg) => {
  console.log(msg)
})
```

a) a b) b c) b a d) a b

(Promise.race())

15. Write the output of the following code.

```
const promise1 = new Promise((resolve,reject) => {
  setTimeout(() => {
    resolve('a')
  })
})

const promise2 = new Promise((resolve,reject) => {
  resolve('b')
})

const promise3 = new Promise((resolve,reject) => {
  setTimeout(() => {
    resolve('c')
  },10)
})


Promise.race([promise1,promise2,promise3])
.then((msg) => {
  console.log(msg)
})
```

****