Name: Akash Gujarathi
B-Number: B00765802
Course: CS-544-01

1]
Example where this behavior is useful,

- If we have an array_1 = [1,2,3,4,5,6,7] and want to return an array which multiplies all the element by ‘x’ then,
- Having called array_1.map() creates a new array with the results of calling a provided function on every element in the calling array
- map provides callback function once for each element in an array, in order, and constructs a new array from the results
  console.log(array_1.map(x => x * 2)); will return me an array = [2,4,6,8,10,12,14]

  Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

2]

a. let r = RegExp('/0[0-7]*/')

b. let r = RegExp(/[+-]?0*('10|11|100)?/ ')

c.

d. Regex for equal number of dashes and dots is not possible
   Just for morse code not equal number of dashes and dot

   Reference: https://stackoverflow.com/questions/17197887/java-regexp-match-morse-code

   let r = RegExp('[+.|+\s|+-]')
   Tested examples
   console.log(r.test('-.. --- --.')) //true
   console.log(r.test('... ..-.-. ...- .. ...- .- .-..')) //true
   console.log(r.test('...---...')) //true
   console.log(r.test('-. . . -..')) //true\

e. let r = RegExp('^((?:[0-9A-Fa-f]{1,4}))((?::[0-9A-Fa-f]{1,4}))*::((?:[0-9A-Fa-f]{1,4}))((?::[0-9A-Fa-f]{1,4}))*|((?:[0-9A-Fa-f]{1,4}))((?::[0-9A-Fa-f]{1,4})){7}$')

   Tested examples
   console.log(r.test('FE80:0000:0000:0000:0202:B3FF:FE1E:8329')) //true
   console.log(r.test('2001:0db8:85a3:0000:0000:8a2e:0370:7334')) //true
   console.log(r.test('2001:0db8:3c4d:0015:0000:0000:1a2f:1a2b')) //true
   console.log(r.test('2340:1111:AAAA:0001:1234:5678:9ABC:1234')) //true

3]

```
/*text will be a input string*/
function extractEmails ( text ){
  return text.match(/([a-zA-Z0-9._-]+(@|at|\sat|at\s|\sat|\sat\s)[a-zA-Z0-9._-]+(\.|dot|\sdot|dot\s|\sdot\s)[a-zA-Z0-9_-]+)/gi);
  }
```

Example tested

```
console.log(extractEmails("abc@b@inghamton.edu abc abcatbinghamton.com afgeufe f dshfief hwh wef abc at g dot com"))
```

Output

```
[ 'b@inghamton.edu', 'abcatbinghamton.com', 'abc at g dot com' ]
=> undefined
```

4]

```
let Random = {
  "lastRand" : 0.5,
  "randNum" : 0.5,
  "rand" : function(){
    while(this.randNum == this.lastRand){
      this.randNum = (new Date().getTime()*Math.PI*this.lastRand)%1;
    }
    return this.lastRand = this.randNum;
  }
}
console.log(Random.rand());
```

5]

A]

```
fs.readFile(pathOfFile, (err, data) => {
 if (err) throw err;
 console.log(data);
});
```

- The callback is passed two arguments (err, data),
- err are the errors and if there are any are thrown using the if statement
- Data is the contents of the file.

```
fs.writeFile('message to be written', (err, message) => {
 if (err) throw err;
 console.log(message);
});
```

- The callback is passed two arguments (err, message)
- err are the errors and if there are any are thrown using the if statement
- Message is the contents written to the file.

Reference: https://nodejs.org/dist/latest-v10.x/docs/api/fs.html#fs_fs_readfile_path_options_callback

B]

fsPromises.readFile((path[,options]).catch(() =>console.log(error))

- Function returns a <promise>
- Promise is resolved with content of the file
- If there are any errors .catch() will display them

fsPromises.writeFile((path), message)).catch() =>console.log(error).then()=>console.log(message))

- Function returns a <promise>
- Promise is resolved by writing content to the file
- If there are any errors .catch() will display them

Reference: https://nodejs.org/dist/latest-v10.x/docs/api/fs.html#fs_fspromises_readfile_path_options
Reference: https://dev.to/mrm8488/from-callbacks-to-fspromises-to-handle-the-file-system-in-nodejs-56p2

C]

```
const readFile = promisify(fs.readFile)
const read = async () => { const content = await readFile(path, fileFormat) return content }
read().then(content => console.log(content))
read().catch(err => console.log(err))
```

- We need to promisify in order to use async and await
- await needs to be used inside async
- .then will display the content of the file
- .catch returns the errors if any

```
const readFile = promisify(fs.writeFile)
const write = async () => { const content = await writeFile(message,path) return content }
write().then(content => console.log(content))
write().catch(err => console.log(err))
```

- We need to promisify in order to use async and await
- await needs to be used inside async
- .then will display the content written to the file
- .catch returns the errors if any

Reference: https://dev.to/damcosset/asynchronous-code-with-asyncawait-7cd

7]

Specification Bugs

```
09        for (var c in cutoffs) {
10                if (total < c) categories.name = cutoffs[c];
11        }
```

a)

In this loop the cutoff will be assigned the maximum value as it just checks if total is less that c
Example if the total is 5 then ideally cutoff should be fired, but as if condition just checks if total is less that c that cutoff value that will be assigned will be outstanding as 5 < 9

b)

categories.name = cutoffs[c];
categories.name is assigned cutoff value, whereas specification requires code to send [names]: 'cutoff'

Style Problems

02 var categories = {};
can be replaced by let categories = {};

03 var c = 0;
replaced to let c = 0;

04 sum = 0;
replaced to let sum = 0;

06 var total = scores.name.total;
replaced to let total = scores.name.total

Brittleness

05 **for** (var name in scores)

Object gets methods and properties passed by JavaScript itself. Those are methods that every object gets when its created.
.hasOwnProperty to find only the properties and methods you assigned to the object

```
for (var name in scores){
        If(scores.hasOwnProperty(name))
}
```

New fixed version

```
/*
        Assuming if there are same names then new value replaces old value
*/
function performanceReview(scores, cutoffs) {
        let categories = {};
        let total = 0;
        for(let names in scores){
                total += scores.name.total;
                categories[scores.name] = cutoffs[Math.floor(scores.name.total)]
        }
        return [total/(Object.keys(scores).length), categories]
}
```

8]

```
        async function cheapestFare(fromAirport, toAirport,
                                date, fareType)
           {
            let cheapest = Number.MAX_SAFE_INTEGER;
            for (const [airline, params] of
                Object.entries(AIRLINE_PARAMS))
            {
              const fare =
                await getAirFare(params, fromAirport, toAirport,
                            date, fareType);
              if (fare < cheapest) cheapest = fare;
            }
            return cheapest;
           }
```

- await makes program slower as it awaits for each response
- instead we can use array of promises

```
async function cheapestFare(fromAirport, toAirport,date, fareType)
{
 let cheapest = Number.MAX_SAFE_INTEGER;
 let promiseArray = [];
   for (const [airline, params] of Object.entries(AIRLINE_PARAMS))
   {
    promiseArray.push(getAirFare(params, fromAirport, toAirport,date, fareType));
   }
    const fare =  await Promise.all(promiseArray);
    if (fare < cheapest) cheapest = fare;
    return cheapest;
}
```

9]

A]

True, console.log((1e9-1)*2)
1999999998

B]

False, object as const can be changed

C]

True, F() === new F() is always false
tested example

```
function f(){
        return true
}

console.log(f()=== new f()) // false
```

D]

False, constructor doesn’t not require it’s name to start with upper case

E]

False, change is not seen immediately