# Programming for the Web (CS-544-01)
# Homework – 2

**Name – Chaitanya Kulkarni** **B-Number: B00814455**

**Ans 1 –**

The problem with the code is that array daysOfWeek is declared globally as var. Instead we can use const to declare it. Also, since it is declared globally, the array will get initialized with every function call. Hence to initialize it only once, we can declare it inside the closure of the returning function. Below function can be used instead of the given function

```
const getDayOfWeek = (function() {
    const daysOfWeek = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'];
    return function(n) {
        return daysOfWeek[n];
    };
})();
```

**Ans 2 –**

Implementation of the Counter constructor –

```
class Counter{
    constructor(){
        let counter = 0;
        this.inc = function inc() { counter++; return counter; };
        this.val = function val() { return counter; }
    }
}
```

**Ans 3 –**

The function defined here is an asynchronous callback. It will execute when file loading is completed. To make it run, we need to wrap our call in a function and pass in our own callbacks. Or rather just call the functions using async and await. Any of both the solutions should work.

```
const DIR = `${process.env.HOME}/my-stuff`;
function readFile(callback) {
    fs.readFile(`${DIR}/${filename}`, function (err, result) {
        if (err) return callback(err)
        callback(null, result)
    })
}
readFile(function (err, content) {
    console.log(content)
})
```

OR

```
const DIR = `${process.env.HOME}/my-stuff`;
async function readFile(filename) {
    await fs.readFile(`${DIR}/${filename}`,
        function(err, result) {
            //console.log(err + " Result = " + result);
            return result;
        });
}
const contents = readFile('input.txt');
```

**Ans 4 –**

The given code is slow because it calls the function getShipCost() using await serially many times for all shipping companies. Since these companies are independent, we can use promise.all() and make parallel calls to that function.

```
const SHIPPING_PARAMS = {
    usps: { ... },
    ...,
    ups: { ... },
    fedex: { ... },
};
async function cheapestShipping(fromZipCode, toZipCode, shipType) {
    const promises = Object.entries(SHIPPING_PARAMS)
                           .map(([shipper, params]) => {
                               getShipCost(params, fromZipCode, toZipCode, shipType)
                           });
                   return (await Promise.all(promises)).reduce((cheapest, cost) =>
                       cost < cheapest ? cost: cheapest, Number.MAX_SAFE_INTEGER)
}
```

**Ans 5 –**

Function for `curried_mul` given below –

```
function curried_mul3(a) {
    return function(b) {
        return (c) => mul3(a, b, c);
    }
}
```

**Ans 6 –**

Since both the functions return `this.x`, that means they return the value of x in their respective objects.
Therefore, `obj1.f()` returns 21
`obj2.f()` returns 31 and
`f()` will return the value of obj2.x since f is bound to obj2.
Therefore `f()` returns 31
Hence the output will be = 21 - 31 + 31 = 21

**Ans 7 –**

```
function sumCont(array, ret, index) {
    return index === array.length
        ? ret(0)
        : sumCont(array, x=> { ret(array[index-1] += x) }, ++index);
}

sumCont([1,2,3,4,5], x=>console.log(x), 0)
```

**Ans 8 –**

**Specification bugs -**

- The object that maps student names to respective inner object (assignments and total), what if two students name is same, it may raise a conflict. Instead we can assign their roll numbers as an id

**Coding bugs -**

- to access the total of students in object, we need to write `grades[name].total`

- Same for letters.name, instead it should be `letters[name]`
- There is missing comma (,) where c and sum are initialized
- There is no check to see if empty object passed, in that case the number of students will be 0. That means the function might return sum/0 as aggregate which can print NaN
- in CUTOFFS object, 100 is put in quotes ''
- if statement is not breaking after assigning the letter grade to the respective student. Resulting in assigning "A" grade to everyone. Hence `break` the if statement after the letter grade is assigned

**Style bugs -**

- Instead of returning the array of average as first element and list of students and their grades, we can write separate functions to get aggregate and list of students
- Same function is used to calculate both average and grades, this violates the basic rule of coding that each function should be dedicated to one proper calculation
- Code is poor in style since all the modules are calculated in the same function
- Object oriented features are not properly styled. Could use OOPS techniques in better way
- To iterate over the list, for-in loop is used, there are inbuilt javascript functions defined to iterate over the list

**Brittleness bugs -**

- No proper exception handling is done in the code. If any of the minor mistake happens in the code, it might raise an exception
- Example, while passing student object, it is passing only names property of students, instead what if roll number property was passed, it now has 2 properties, i.e. Names and Roll numbers. Here the for-in loop will not know the correct property. It has to be determined using `Object.hasOwnProperty()` function of the object
- Also, if empty array is passed, it might break due to divide by 0 exception in average

**Below is the improvised code –**

```
const CUTOFFS = { 60: 'F', 65: 'D', 75: 'C', 85: 'B', 100: 'A' };

const grade = {
    ['Ben Bitdiddle']: { hw1: 88, total: 78 },
    [ 'Alyssa P. Hacker']: { hw1: 100, total: 97 },
    [ 'Louis Reasoner']: { hw1: 66, total: 62 }
};

function getAggregate(grades){
    const totalStudents = Object.getOwnPropertyNames(grades)
    const sum = totalStudents.reduce((acc, index) => acc + grades[index].total,
0)
    return (totalStudents.length !== 0) ? sum/totalStudents.length : 0
}

function getLetterGrades(grades, cutoffs){
    const letterGrade = {}
    const totalStudents = Object.getOwnPropertyNames(grades)
    totalStudents.map(name=> {
        let total = grades[name].total;
        for (let marks in cutoffs){
            if(total < marks){
                letterGrade[name] = cutoffs[marks];
                break;
            }
        }
```

```
    })
    return letterGrade;
}
console.log(getAggregate(grade))
console.log(getLetterGrades(grade, CUTOFFS))
```

**Ans 9 –**

A. **True** - Since return type of String.match() function is array, if the match is found then this will return the match as an array. It will throw an error inside if statement since it requires Boolean Expression.

B. **False** - It is true that according to Javascript naming conventions, you should only capitalise the first character of the name of a function when you need to construct the object by "new" keyword. This is called "the Constructor Invocation Pattern" stated in the book "Javascript: the good parts". But, as long as we reference the constructors and functions with the same casing the code should Work. Hence it is not mandatory to Uppercase the first character of a function name.

C. **True** - It is impossible to wrap an asynchronous function within a synchronous function. To make the function synchronous, call that using async and await. Async function returns promise implicitly and resolved value of that promise will be the value that is returned from our function. Hence our function will be called using async keyword and it will contain await keyword inside function which will wait for a Promise to return. In this way we can make asynchronous function as synchronous function.

D. **True** – Yes, once a Promise is fulfilled or rejected, the respective handler function will be called asynchronously and will return a Promise object.

E. **False** – Since Promise.all() makes calls concurrently on multiple asynchronous tasks, and creates their promise for result, it waits for all the tasks being finished and then will return the single promise. Suppose there are 5 async tasks being called by Promise.all(), taking times 2, 5, 1, 2,4 seconds respectively. It will not wait for their total time(14 secs) to return the single promise. Instead it will wait for all tasks to finish i.e. 5 seconds and will return the promise.