

# Final Exam

|  |  |
| --- | --- |
| **Instructor:** | **ALBERT DANISON** |
| **Class:** | **MAD6135** |

|  |
| --- |
| * Read all questions carefully. * Partial marks can be awarded. * Time allowed: 120 minutes * Total marks: 80 |

|  |  |  |
| --- | --- | --- |
| Q1 |  | /10 |
| Q2 |  | /10 |
| Q3 |  | /10 |
| Q4 |  | /10 |
| Q5 |  | /10 |
| Q6 |  | /10 |
| Q7 |  | /10 |
| Q8 |  | /10 |
| Total |  | /80 |

|  |  |
| --- | --- |
| **Date:** | **01-03-2021** |
| **Student name:** | Sriram Elangovan |
| **Student number:** | C0769009 |

## How to check if an object is an array or not? Provide an example.(10 marks).

let sampleArray1 = ["A1", "A2", "A3", "A4"]

let sampleArray2 = {id: '1', name: "Sriram"}

//function will check whether parameter value is array or not

function verifyArray(tempArray) {

if(Array.isArray(tempArray)) {

console.log("The given input is an array")

} else{

console.log("Sorry. The given input is not an array.")

}

}

verifyArray(sampleArray1) // **OUTPUT**: The given input is an array

verifyArray(sampleArray2) // **OUTPUT**: Sorry. The given input is not an array.

1. Implement enqueue and dequeue in JavaScript using only two stacks. *Enqueue* means to add an element, *dequeue* to remove an element. (10 marks)

I have written program for multiple enqueues and dequeues consecutively.

let sampleArray1 = []

let sampleArray2 = []

function enqueue(value) {

**//if dequeued before then we are reversing and moving all the values to sampleArray1(stack1)**

if(sampleArray2.length > 0) {

while(sampleArray2.length > 0) {

sampleArray1.push(sampleArray2.pop())

}

}

sampleArray1.push(value)

**//stack 1 after value is added**

console.log(sampleArray1)

}

function dequeue() {

**//if stack2 (sampleArray2) is empty then we are reverse and moving values from sample array 1**

if(sampleArray2.length == 0){

if(sampleArray1.length > 0) {

while(sampleArray1.length > 0) {

sampleArray2.push(sampleArray1.pop())

// console.log(sampleArray1)

console.log(sampleArray2)

}

}

console.log("The dequeued value is "+ sampleArray2.pop())

} else if (sampleArray2.length > 0) { **// if stack2 already has value then dequeued directly**

console.log(sampleArray2.pop())

}

}

enqueue(1) // **output: [1]**

enqueue(2) // **output: [1,2]**

enqueue(3) // **output: [1,2,3]**

dequeue() // **output: The dequeued value is 1**

enqueue(4) //**output: [2, 3, 4]**

dequeue() // **output: The dequeued value is 2**

dequeue() //**output: The dequeued value is 3**

1. Write a function doBase(N)that would allow you to do this. (10 marks)

var add6 = doBase(6);

add6(10); // returns 16

add6(21); // returns 27

var add6 = (N) => {

console.log(N + 6)

}

add6(10) // **Output: 16**

add6(21) // **Output: 27**

add6(30) // **Output: 36**

1. Given two strings, return true if they are anagrams of one another and false if they are not. (10 marks)

For example: Mary is an anagram of Army.

function isAnagram(str1, str2) {

let words1 = str1.split("")

let words2 = str2.split("")

let isAnagramFlag = true

if (str1.length == str2.length) {

for (let i = 0; i < words1.length; i++) {

if (!(words1[i].indexOf(words2))) {

isAnagramFlag = false

break;

}

}

} else {

isAnagramFlag = false

}

let result = isAnagramFlag ? "The given strings are anagrams" : "The given strings are not anagrams"

console.log(result)

}

isAnagram("Mary", "Army") **// Output: The given strings are anagrams**

isAnagram("Listen", "SILENT") **// Output: The given strings are anagrams**

isAnagram("Something", "Nothing") **// Output: The given strings are not anagrams**

1. Create a function that takes two numbers and a mathematical operator **+ - \* /** ^ and will return the result based on the given numbers and chosen operation. (20 marks)

Examples:

calculator(2, "+", 2) ➞ 4

calculator(2, "\*", 2) ➞ 4

calculator(4, "/", 2) ➞ 2

calculator(4, "-", -5) ➞ -1

calculator(4, "^", 2) ➞ 16

Note:

If the input tries to divide by 0, return: "Can't divide by 0!"

function calculator(num1, num2, action) {

switch (action) {

case '+': {

console.log("The result is: " + (num1 + num2))

break;

}

case '\*': {

console.log("The result is: " + (num1 \* num2))

break;

}

case '/': {

if (num2 == 0) {

console.log("Can't divide by 0")

} else {

console.log("The result is: " + (num1 / num2))

}

break;

}

case '-': {

console.log("The result is: " + (num1 - num2))

break;

}

case '^': {

console.log("The result is: " + (Math.pow(num1, num2)))

break;

}

}

}

calculator(1, 2, '+') **// output: The result is: 3**

calculator(2, 3, '\*') **// output: The result is: 6**

calculator(4, 8, '/') **// output: The result is: 0.5**

calculator(4, 0, '/') **// output: Can't divide by 0**

calculator(7, 3, '-') **// output: The result is: 4**

calculator(5, 3, '^') **// output: The result is: 125**

1. How are Maps and Sets implemented in JavaScript? Provide a practical example of their use in JavaScript. (10 marks)

Map and Set are part of data structure concepts which will be very useful for programming. However, map and sets are not available before ES6, but those can be implemented with Array and Object. Although the Map and Sets are implemented in alternative ways before ES6, there are some potential drawbacks.

To overcome disadvantage and improve the javascript language, the Map and Set are released in ES6, which user to create and manipulate values easily.

**Set:**

User can create a Set easily with ‘new Set()’

e.g: let values = new Set()

Some significant method in Set are,

Add Value:

values.add(1)

values.add(2)

values.add(3)

Delete Value:

values.delete(2)

Search Value:

values.has(2) // returns true

values.has(4) // returns false

The other methods includes clear() to clear all values inside the Set, values.forEach() to iterate the values, size() to iterate the size of the sets.

Realtime Scenarios:

Set is really useful to hold unique values. For example, if we like to store student ids and do some operations then set is the best data structure to use.

**Map:**

Map is like an object with key and value pair. The map would have entries with key as point of reference and value would be of any type from primitive values like number, string, float to objects, array.

User can create a Map with new Map()

let students = new Map()

Add Value:

students.set(“Rahul’, {fatherName: “Raj”, class:”8”, marks: 80})

students.set(“Sri, {fatherName: “Kal”, class:”10”, marks: 75})

students.set(“Ram’, {fatherName: “Lars”, class:”7”, marks: 90})

To retrieve a value use get(key)

students.get(‘Sri’)

Delete Values:

students.delete(‘Sri’)

Other important methods are size() returns the map size, students.values() return an iterator to retrieve values, Similarly, students.key() return an iterator which returns only key.

Realtime Scenarios:

Map is useful any place which store object or group values upon a single reference(key) and do some operations on it.

1. What is the role of Iterators and Generators in JavaScript. Provide an example of each. (10 marks)
2. How is Asynchronous Programming implemented in JavaScript? Provide an example. (10 marks)

Asynchronous programming improves the web application performance and provides new user experience. Unlike old fashion the user had to wait until the required data is fetched from the server, the asynchronous programming let the user to render the non-dependent components on the User interface instead of waiting for the data.

The asynchronous programming is done in two ways,

Call Backs

Promises

Call Backs:

Call Backs are like the old fashion way of implementing the asynchronous programming,

function loadData(url, type, callback) {

let xhr = new XMLHttpRequest();

xhr.open('GET', url);

xhr.responseType = type;

xhr.onload = function() {

callback(xhr.response);

};

xhr.send();

}

function showValues(values) {

let data = document.createElement('p');

p.innerHtml = values

document.body.appendChild(data);

}

loadData (‘/getData', ‘text’, showValues);

In the above example the showValues is an callback function which only be executed after the data is retrieved submitted.