Examen 1

Pregunta 1:

- a)

  Voy a seleccionar el lenguaje de JavaScript, mi nombre es Jose y empieza con J.

  - 1a) Sobre las asociaciones presentes en JS (sobrenombre corto de JavaScript) tenemos a la asociación temprana, en el aspecto de que el programador tiene la libertad de declarar variables, funciones, procedimientos y módulos (de los que hablaremos luego) a medida que es escrito el código con el nombre y parámetros que sean deseados, además JS cuenta con una amplia gama de funciones predefinidas en el lenguaje que ayudan al programador a poder lograr el objetivo que este desea (manipulación de datos con arrays y objetos, operaciones matemáticas con la librería Math ,etc.) , ciclos/repeticiones y condicionales (incluido el operar ternario) . A su vez tenemos presente a la asociación tardía ya que JS es un lenguaje que es interpretado por lo que a medida que se va ejecutando/leyendo la secuencia de código es que se conoce el estado del programa y por lo tanto el valor de las variable, constantes etc.

    Lo positivo de este acercamiento hecho por el diseñador/diseñadores del lenguaje, es que JS es un lenguaje flexible y por lo tanto los programadores pueden realizar lo que necesiten con el, en particular , JS es el principal lenguaje del diseño Web ya que es el lenguaje que utilizan los navegadores (que tienen distintos motores de Interpretación de JS) aunque también es utilizado para otro tipo de aplicaciones (Apps de consola, APIs, etc.).

    El principal efecto negativo de la flexibilidad de JS al no ser un lenguaje fuertemente tipado es que, el programador se puede equivocar en la asignación en el tipo de dato que esta asignando a variables y funciones , por lo que pueden ocurrir errores al momento de correr los programas en JS por asignación y manipulación errónea de datos, es decir, pueden ocurrir errores o bugs terminales en ambientes de producción. Aunque este efecto negativo puede ser mitigado utilizando TypeScript que es un lenguaje construido (por Microsoft) sobre JS que es fuertemente tipado.

    El alcance de JS es estático, JS en particular tiene un scope muy bien definido para sus bloques de códigos y variables. Lo positivo de esto es que para el programador es relativamente fácil leer un código de JS y saber que es lo que se esta realizando y que variables o valores serán utilizados al momento de la ejecución.

- 2a) Sobre los módulos en JS, pasa algo muy curioso con ellos y es que no tienen un nombre, es decir, no tenemos declaraciones explicitas (al menos no tradicionalmente, pero si existe una sintaxis de import * as Module from ´folder´, que crea una instancia de un objeto Module el cual puede acceder a todo lo que tiene la etiqueta export del archivo importado) por lo general cuando queremos crear un modulo, generamos un nuevo archivo de JS con la lógica separada que deseamos exportar, esto pueden ser clases con sus respectivos métodos y constructores, como funciones que sean para realizar cierto tipo de operaciones y manipulaciones. Existen varias maneras de realizar importaciones y exportaciones, para exportar podemos crear default exports, una lista de funciones/clases que deseamos exportar o colocar la etiqueta export al mismo nivel de la declaración de lo que deseamos exportar. Y para importar utilizamos la palabra clave import { Nombres de variables, funciones o clases separadas por comas } from ‘Ruta del archivo’.
- 3a) Sobre alias en JS, tenemos a la desestructuración de objetos o arrays, por ejemplo, supongamos que tenemos :

```
let myObject = {
  x: 2,
  alo: "Como alo"
}

/*
  Si queremos acceder a los valores x o alo de myObject podemos hacer myObject.x
  o realizar una desestructuracion con un cambio de nombre a la variable.
*/

myObject.x = 50
// Imprime 50
console.log(myObject.x)

const {x : miAlias } = myObject
 miAlias = 0

// Imprime 0
console.log(myObject.x)
```

Respecto a la sobrecarga, en JS no podemos sobrecargar funciones, al menos no de manera directa, pero si podemos hacerlo mediante condicionales, mientras verificamos la cantidad de argumentos pasados a la firma de la misma por ejemplo:

```
 funtion hola() {
  switch (arguments.length) {
    case 0:
    console.log("Hola no has pasado argumentos para realizar acciones")
    break
```

```
    case 1:
    console.log(`Hola pasaste el argumento ${arguments[0]}`)

    default:
    console.log("Has pasado al menos dos elementos hagamos cosas!")
    let result = 0
    const size = arguments.length
    arguments.forEach((arg) => {
      result= result +(arg % 2 ) * 4
      })
    console.log(result)
    }
}

hola()
//Imprime el caso 0
hola(4)
//Imprime el caso 1
hola(1,2,3)
//Imprime el caso default
```

Finalmente en JS hay polimorfismos de la manera clásica de un lenguaje orientado a objetos y es mediante una clase que extiende (hereda de otra), por ejemplo :

```
 class Vehiculo {
  runRun() {
    console.log("RUNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN")
  }
}

class Carro extends Vehiculo {
}

let corsa = new Carro()
corsa.runRun()
// Imprime RUNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

- 4a) Como mencione anteriormente JS es el lenguaje que posee la supremacía del diseño Web, además de ser utilizado para apps de consola y backends. Por lo tanto tiene una amplia gama de herramientas para ser usados por los desarrolladores, por ejemplo hay varios frameworks tanto para desarrollo de frontend (Angular,React) como de backend(Express con nodeJS), también posee varias librerías de testing como jest, karma y jasmine para pruebas unitarias y cypress para pruebas end to end. Sobre debbugers JS puede ser debugueado tanto en buscadores como chrome como con Node. A su vez el mismo chrome posee un profiler para JS incluido en sus

herramientas de desarrollador, además cada navegador posee un motor de interpretación que le permite leer código en JS.