Examen de conocimientos
Este examen se realiza con la finalidad de conocer la forma lógica de pensar del candidato/candidata y comprobar que tiene el conocimiento solicitado, la primer parte se compone con una serie de preguntas las cuales se pueden resolver con pseudocodigo (serie de pasos) y la segunda con un ejercicio práctico.

Primer parte (tiempo máximo 1 hora):

1. Diseñar un algoritmo para resolver la problemática de turnos en el departamento de urgencias de un hospital, las prioridades son las siguientes:
    - Por color, hay tres colores: rojo, naranja y verde, donde el rojo tiene más prioridad que el naranja y el naranja más que el verde.
    - Por tiempo de llegada, el primero que llega es el primero que atienden, pero respetando la prioridad por color, es decir, si un naranja llegó antes que un rojo, se atiende primero al rojo.

2. Diseñar un algoritmo que invierta el orden de un arreglo dado con longitud x, tratar de hacerlo con los menos pasos posibles.
    - Ejemplo de entrada [0,2,4,3,1,5,6,7,8] tendría de salida [8,7,6,5,1,3,4,2,0]

3. JS: Explicar brevemente las diferencias entre callback, promesas sin usar el async/await, promesas usando el async/await y observadores.

Segunda parte (tiempo máximo 3 horas):
1. Crear con **nodejs**, **mongodb** y **angular** 2+ un **singin/signup** utilizando tecnologías como **mongoose**, **express**, **jwt**, las características deben ser las siguientes:
    - Protección de rutas, los usuarios firmados sólo podrán acceder al “home”, mientras que los no firmados sólo podrán acceder al signin y signup.
    - Almacenamiento del token por lado del cliente.
    - Si te sientes familiarizado con otra librería puedes usarla, por ejemplo en vez de express usar sails, hapi o cualquier otra.
    - La entrega del proyecto será por medio de GitHub o similar.

No importa si el examen no es terminado al 100%, al terminar en total las 4 horas es conveniente mandarlo dado que no se calificará, sólo se utilizará como referencia para conocer tus aptitudes y capacidad de razonamiento.

P1 – Primer pseudocódigo:

```
INICIO
    enOrden = {   rojo:[],
                  naranja:[],
                  verde:[]
              };
    tempNombre = “”;
    tempColor = “”;
    tempPaciente = {};
    SI Llegó nuevo paciente
        IMPRIMIR “Introduzca el nombre”:
        LEER -> tempNombre;
        IMPRIMIR “Introduzca prioridad (rojo/naranja/verde)”:
        LEER -> tempColor;
        tempPaciente={
            nombre: tempNombre,
            color: tempColor
        };
        SEGÚN tempPaciente.color HACER
            CASO rojo
                enOrden.rojo.push(tempPaciente.nombre);
            CASO naranja
                enOrden.naranja.push(tempPaciente.nombre);
            CASO verde
                enOrden.verde.push(tempPaciente.nombre);
        FIN SEGÚN
        IMPRIMIR “Se ordenó al paciente ” + tempPaciente.nombre;
    FIN SI
    SI Atenderá a paciente
        SI enOrden.rojo no está vacío
            IMPRIMIR “Se atenderá al paciente: ” + enOrden.rojo.shift();
            IR A FIN
        SI NO, SI enOrden.naranja no está vacío
            IMPRIMIR “Se atenderá al paciente: ” + enOrden.naranja.shift();
            IR A FIN
        SI NO, SI enOrden.verde está vacío
            IMPRIMIR “Se atenderá al paciente: ” + enOrden.verde.shift();
            IR A FIN
        SI NO
            IMPRIMIR “No hay pacientes en espera”;
            IR A FIN
        FIN SI
    FIN SI
FIN
```

P2 – Segundo pseudocódigo:

INICIO

array = [];

newArray = [];

IMPRIMIR “Introduzca arreglo a invertir”;

LEER -> array;

SI array está vacío

IMPRIMIR “Array vacío”

IR A FIN

FIN SI

MIENTRAS array no esté vacío **HACER**

newArray.push(array.pop());

FIN MIENTRAS

FIN

Ojo: Existe una función que te revierte cualquier arreglo (array.reverse())

P3 - Pregunta

Explicar brevemente las diferencias entre callback, promesas sin usar el async/await, promesas usando el async/await y observadores.

Callback: Es un función que se pasa a otra función como parámetro.

Promesas: Es una función que recibe 2 funciones como parámetros siendo el primero para cuando se cumple una condición y la segunda el resultado de que no se cumpla la condición.

Async/wait: Es una pausa que ponemos durante la ejecución para esperar el resultado de una función para así continuar con la ejecución.

Observadores: No había escuchado el término