****

**Programación Imperativa**

# Examen final de Programación Imperativa

## Objetivo

Evaluar las habilidades adquiridas durante la cursada en el contexto de la Programación Imperativa utilizando Javascript corriendo en un entorno de Node.

## Metodología de evaluación

En base a las consignas requeridas en el examen, se evaluarán los siguientes conceptos sobre el código entregado:

* **FORMA**
  + Que el código esté prolijo e implemente buenas prácticas
  + Que las variables, métodos y funciones tengan nombres descriptivos
  + Que se utilicen nombres en español o en inglés pero no ambos
  + Que se utilice camelCase donde corresponda
* **LÓGICA**
  + Que la lógica corresponda con lo que solicitan las consignas
  + Que se utilicen los métodos más adecuados para resolver cada problema
* **FUNCIONAMIENTO**
  + Que el código funcione correctamente, sin arrojar errores
  + Que el código produzca el resultado esperado a partir de los datos suministrados

## Duración y entrega

El examen tendrá una duración de 2 horas. Durante los primeros 15 minutos se explicará el funcionamiento del mismo y durante los siguientes 105 minutos tendrán tiempo para resolverlo. Las entregas realizadas luego del tiempo estipulado no serán tenidas en cuenta.

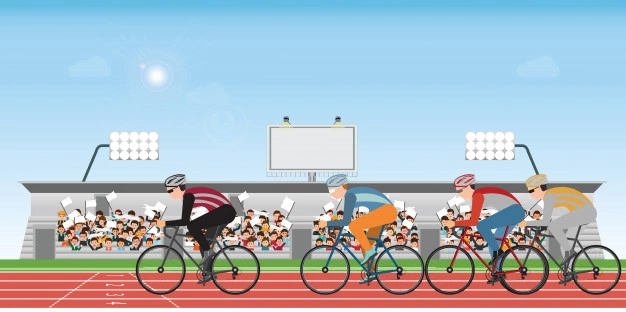
Al terminar el examen deberán entregarlo haciendo uso de [**este formulario**](https://forms.gle/b6JbcVTwXawEoXft6).

*¡Les deseamos lo mejor en el examen! 🙌*

## Consignas

En este examen estaremos modelando una carrera de bicicletas, teniendo en cuenta la inscripción de los mismos en base a determinadas condiciones y generando la lógica necesaria para simular el resultado de la carrera, determinando en qué puestos quedarán los corredores.

Los datos de los autos los obtendremos de un archivo con formato JSON que contendrá un array de objetos literales que representarán los autos.



# Consignas

### **1. Obtener el listado de posibles participantes**

Tomando como base el siguiente [archivo JSON](https://drive.google.com/file/d/194y71JbbWDvLl6gBBXpgtVzzA6kVBLhd/view?usp=sharing), deberán leer el archivo y parsearlo para obtener el listado de autos que desean participar de la carrera.

**Resultado esperado:** variable conteniendo un array con todos los autos disponibles

### **2. Crear un objeto literal que represente la carrera**

Este objeto contendrá, una propiedad con el listado de bicicletas obtenido en el punto anterior y las **funcionalidades** que nos solicitan a continuación:

1. Agregar una propiedad llamada ***bicicletas*** que contenga las bicicletas.
2. Agregar una propiedad llamada **bicisPorTanda** que contenga el valor **7**. Este valor representará la cantidad máxima de bicicletas por tanda.
3. Agregar un método **bicicletasHabilitadas** que permita consultar las bicicletas habilitadas, es decir, que devuelva una lista de las mismas cuyo dopping no sea positivo.

**Resultado esperado:** un array con las bicicletas habilitadas para correr (doping negativo)

1. Agregar un método **listarBicicletas** que reciba como parámetro un array de bicicletas y los imprima por consola.

Este método deberá imprimir por cada elemento:

* + El id.
  + El rodado.
  + El peso en kg
  + *“descalificado”* → En caso de ser true la propiedad doppingPositivo  
    *“habilitado”* → Caso contrario

**Resultado esperado:** un mensaje por consola por cada auto con el siguiente formato: *“Id: X, peso en kg: XXX, estado: descalificado / habilitado”*.

Ej 1: “Id: 5, peso en Kg: 7.569, estado: habilitado”  
Ej 2: “Id: 5, peso en Kg: 7.569, estado: Habilitado”

1. Agregar un método **buscarPorId** que permita buscar una bicicleta en función de su id.
   * Este método debe devolver una bicicleta en caso de encontrar el id en todo el array de la propiedad autos.

**Resultado esperado:** el elemento que coincida con el id enviado por parámetro.

1. Agregar un método **buscarPorRodado** que permita filtrar las bicicletas habilitados, siempre y cuando su rodado sea menor o igual al enviado como argumento.
   * Este método debe usar **bicicletasHabilitadas.**

**Resultado esperado:** un array que contenga todas las bicicletas que cumplan con las condiciones mencionadas.

1. Agregar un método que permita **ordenar por rodado** de menor a mayor según el rodado de cada bicicleta y devolver un array ordenado.

**Resultado esperado:** la lista de bicicletas ordenada por puntaje de menor a mayor.

1. Agregar un método **generarTanda** que permita retornar un array de bicicletas, que cumpla con las siguientes condiciones:
   * únicamente bicicletas habilitadas
   * rodado menor o igual un valor enviado como argumento.
   * peso en kg menor o igual a un valor enviado como argumento

**Resultado esperado:** un array con las bicicletas que cumplan las condiciones especificadas y tenga como máximo la cantidad de bicicletas expresada en la propiedad **bicisPorTanda** del objeto carrera.

1. Agregar un método que permita **calcularPodio**, el mismo deberá calcular al ganador y los siguientes dos puestos en función de su puntaje.
   * El podio debe surgir a partir de la **tanda generada**.
   * Las primeros tres bicicletas con mayor puntaje conformaran el podio

**Resultado esperado:** imprimir por consola el ciclista y el puntaje del podio diferenciando primer, segundo y tercer puesto.

Ej:  
 “El ganador es: Esteban Piazza, con un puntaje de: 8.1  
el segundo puesto es para: Leandro Ezequiel, con un puntaje de: 5.5  
y el tercer puesto es para: Ezequiel Gentile, con un puntaje de: 5.2”