Actividad 019 - POO, Módulos y Mixins.

- Para poder realizar este actividad debes haber realizado los cursos previos junto con los videos online correspondientes a la experiencia 10.
- Crea una carpeta y guarda cada archivo .rb con el número de la pregunta, siguiendo las instrucciones de manera local con **Sublime** o **Atom**.
- Luego guarda los cambios y súbelos a tu repositorio de Github.
- Luego de pusheados los últimos cambios, sube el link de Github en el desafío de la sección correspondiente en la plataforma.

Ejercicio 1: Sintaxis

Corregir los errores para poder ejecutar ambos métodos.

```
class MiClase
    def de_instancia
        puts 'Método de instancia!'
    end
    def.self de_clase
        puts 'Método de clase!'
    end
end

MiClase.de_instancia
MiClase.new.de_clase
```

Ejercicio 2: Sintaxis

Corregir los errores de sintaxis para que las últimas cuatro líneas se ejecuten de manera correcta

La última instrucción debe imprimir *"Hola! Soy la clase MiClase"*

```
class MiClase
    attr_writer :name
    def initialize(name)
        @name = name
    end

    def self.saludar
        "Hola! Soy la clase #{@name}"
    end
end

c = MiClase.new('Clase Uno')
puts c.name
c.name = 'Nombre Nuevo'
puts MiClase.saludar
```

Ejercicio 3: Herencia

Se tiene la clase *Vehicle* que recibe como argumento un modelo y un año. El método *engine_start* enciende el vehículo.

```
class Vehicle
  def initialize(model, year)
    @model = model
    @year = year
    @start = false
  end

  def engine_start
    @start = true
  end
end
```

Se pide:

- Crear una clase *Car* que herede de *Vehicle*
- El constructor de *Car*, además de heredar las propiedades de *Vehicle*, debe incluir un contador de instancias de *Car*.
- Crear un método de clase en *Car* que devuelva la cantidad de instancias.
- El método *engine_start* heredado debe además imprimir *'El motor se ha encendido!'*.
- Instanciar 10 *Cars*.
- Consultar la cantidad de instancias generadas de *Car* mediante el método de clase creado.

Ejercicio 4: Módulos

Transformar la clase *Semana* en un módulo y obtener la misma salida:

```
class Semana
  @@primer_dia = 'Lunes'

  def self.primer_dia
    @@primer_dia
  end

  def self.en_un_meses
    "Un mes tiene 4 semanas."
  end

  def self.en_un_año
    "Un año tiene 52 semanas."
  end
end

puts "La semana comienza el día #{Semana.primer_dia}"
puts Semana.en_un_meses
puts Semana.en_un_año
```

Ejercicio 5: Mixins.

Transformar la clase *Herviboro* en un módulo. Implementar el módulo en la clase *Conejo* mediante *Mixin* para poder acceder al método *dieta* desde la instancia. Finalmente imprimir la definición de Hervíboro.

```
class Herviboro
  @@definir = 'Sólo me alimento de vegetales!'

  def self.definir
    @@definir
  end

  def dieta
    "Soy un Herviboro!"
  end
end

class Animal
  def saludar
    "Soy un animal!"
  end
end

class Conejo < Animal < Herviboro
  def initialize(name)
    @name = name
  end
end

conejo = Conejo.new('Bugs Bunny')
conejo.saludar
conejo.dieta
Herviboro.definir
```

Pregunta: ¿Por qué es mejor solución la implementación de Mixin que mediante Herencia de Herencia?

Ejercicio 6: Rack

Se tiene el archivo *config.ru* :

```
#config.ru
require 'rack'

class MiPrimeraWebApp
 def call(env)
  [000, {}, []]
 end
end

run MiPrimeraWebApp.new
```

Se pide:

- Agregar un código de respuesta ***200***.
- Agregar en los *Response Headers* un *Content-type* de tipo *text/html*.
- Agregar en el *Response Body* una etiqueta de párrafo que contenga un texto *Lorem ipsum*.