Examen #SeProgramar - Diciembre 2021 T4

Este es el examen de certificación de #SeProgramar. Lee bien cada consigna y tené en cuenta la siguiente información:

🕒 Revisá que la fecha y hora de tu dispositivo sean correctas.

⏳ Una vez que ingresás, tendrás 2 horas para resolver el examen, o hasta que termine el turno (lo que suceda primero).

⚠️ Si salís de la plataforma, el tiempo seguirá corriendo.

💾 Cada vez que enviás una solución, ésta se guarda en nuestro sistema.

✅ Una vez que terminás, simplemente podés salir de la plataforma: todas tus soluciones son almacenadas.

🔇 Cuando terminás no recibirás ninguna notificación automática.

▭ Los resultados del examen serán enviados vía mail antes de la próxima semana.

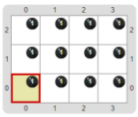
¡Te deseamos mucha suerte y éxitos! 🍀💪

Ejercicios

- 1. Ejercicio 1
- 2. Ejercicio 2
- 3. Ejercicio 3
- 4. Ejercicio 4
- 5. Ejercicio 5
- 6. Ejercicio 6
- 7. Ejercicio 7
- 8. Ejercicio 8
- 9. Ejercicio 9

Ejercicio 1: Ejercicio 1

Una extravagante repostería nos pidió ayuda para decorar su famosa torta rectangular de chocolate 🍫:



Para eso crearemos un programa que ponga confites de color `Azul` en sus extremos de la siguiente forma:

Creá el programa que decore la torta de la forma solicitada. El cabezal comienza en el extremo Sur Oeste y no importa dónde termina.

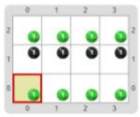
¡Dame una pista!

```
program {
  Poner(Azul)
  IrAlBorde(Norte)
  Poner(Azul)
  IrAlBorde(Este)
  Poner(Azul)
  IrAlBorde(Sur)
  Poner(Azul)
}
```

▶ Enviar

Ejercicio 2: Ejercicio 2

Un grupo de estudiantes nos solicitó ayuda para armar la bandera de su curso utilizando Gobstones 🧑‍💻. La bandera tendrá 3 franjas de 4 celdas cada una y quieren probar la combinación de distintos colores. Para ello definiremos un procedimiento que reciba como argumento el color de las franjas superior e inferior y el color de la franja del medio. Por ejemplo, si lo invocamos con los colores `Verde` y `Negro` haciendo `ProbarCombinacionBandera(Verde, Negro)` la bandera deberá verse así:



Definí el procedimiento `ProbarCombinacionBandera` que reciba dos colores y arme una bandera a partir de ellos. El cabezal comienza en el extremo Sur Oeste y no importa dónde finaliza.

¡Dame una pista!

```
procedure ProbarCombinacionBandera(color1,color2){
  repeat(3){
    Poner(color1)
    Mover(Este)
  }
  Poner(color1)
  Mover(Norte)
    repeat(3){
      Poner(color2)
      Mover(Oeste)
    }
  Poner(color2)
  Mover(Norte)
    repeat(3){
    Poner(color1)
    Mover(Este)
    }
  Poner(color1)
}
```

▶ Enviar

Ejercicio 3: Ejercicio 3

Los colores tienen distintas clasificaciones, dentro de ellas podemos encontrar a los colores primarios: azul, rojo y amarillo 👎. Para distinguirlos tenemos la función `esUnColorPrimario`:

```
esUnColorPrimario("amarillo")
true

esUnColorPrimario("verde")
false
```

Definí la función `esUnColorPrimario` que recibe un color y nos dice si es un color primario o no.

Solución | Consola

```
function esUnColorPrimario(color){
  return (color === "azul") || (color === "rojo") || (color
=== "amarillo") ;
}
```

▶ Enviar

Ejercicio 4: Ejercicio 4

Al salir a la calle es recomendable ver la temperatura 🌡, pero no siempre sabemos si está fresco o caluroso. Para ello definiremos una función que resuelva este dilema. Si la temperatura es menor a 16 diremos que está fresco y en caso contrario que está caluroso:

```
comoEsta(27)
'está caluroso'

comoEsta(16)
'está caluroso'

comoEsta(15)
'está fresco'
```

Definí la función `comoEsta`.

Solución | Consola

```
function comoEsta(temperatura){
  if (temperatura < 16){
    return "esta fresco";
  } else {
    return "esta caluroso" ;
  }
}
```

▶ Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 5: Ejercicio 5

Un vivero quiere saber qué plantas puede trasplantar 🌱. Para ello nos pidió una función que a partir de una lista de alturas de plantas nos diga cuáles tienen más de 28 cms:

```
estanParaTrasplantar([27,30, 10, 28, 29])
[30, 29]
```

Definí la función `estanParaTrasplantar`.

Solución | Biblioteca | Consola

```
function estanParaTrasplantar(listaDeAlturas){
  let sumatoria = []
  for (let altura of listaDeAlturas){
    if (altura > 28){
      agregar (sumatoria, altura);
    }
  }
 return sumatoria;
}
```

▶ Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 6: Ejercicio 6

En una empresa de viajes ✈ guardan registro de los viajes realizados por sus clientes de la siguiente forma:

```
let luca = {
  nombre: "Luca Maggio",
  destinos: ["Neuquén", "Salta", "Mendoza", "Córdoba"],
  fechaUltimoViaje: "15/11/2021"
}

let margarita = {
  nombre: "Margarita Lopez",
  destinos: ["Jujuy", "Tucumán"],
  fechaUltimoViaje: "19/09/2021"
}
```

Definí la función `resumenInformacion` que permita obtener un resumen de la información registrada. Por ejemplo:

```
resumenInformacion(luca)
"Luca Maggio viajó por última vez el 15/11/2021 e hizo en total 4 viajes"

resumenInformacion(margarita)
"Margarita Lopez viajó por última vez el 19/09/2021 e hizo en total 2 viajes"
```

Solución | Biblioteca | Consola

```
function resumenInformacion(cliente) {
  return cliente.nombre+" viajo por ultima vez el "+cliente.fechaUltimoViaje+" e hizo en total "+(longitud(cliente.destinos))+" viajes"
}
```

▶ Enviar

Ejercicio 7: Ejercicio 7

¡Dejemos atrás a JavaScript para pasar a Ruby! 💎

La famosa novela gráfica Sandman tendrá pronto su propia serie ⏳. También quieren desarrollar un videojuego de la misma y para ello vamos a modelar a su protagonista: `Morfeo`. De él sabemos que:

- tiene un descanso inicial de 8;
- puede `dormir!` una cantidad de minutos y aumentar su descanso en esa cantidad;
- si tiene menos de 35 de descanso diremos que `necesita_reposo?`.

Definí en Ruby, el objeto `Morfeo` que tenga un atributo `@descanso` con su getter. El objeto entiende los mensajes `dormir!` (que recibe minutos como argumento) y `necesita_reposo?`. No te olvides de inicializar el atributo `@descanso` con el valor correspondiente.

Solución | Consola

```
module Morfeo
  @descanso=8

  def self.descanso
    @descanso
  end
  def self.dormir!(minutos)
    @descanso+= minutos
  end
  def self.necesita_reposo?
    @descanso < 35
  end
end
```

▶ Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 8: Ejercicio 8

En un curso tenemos un conjunto de estudiantes, a la hora de cerrar las actas es necesario saber quiénes aprobaron ✅. Teniendo en cuenta que cada estudiante sabe responder al mensaje `aprobo?`...

Definí en Ruby el método `personas_aprobadas` que retorne qué estudiantes aprobaron del `Estudiantado`.

Solución | Consola

```
module Estudiantado
  @estudiantes = [May, Gus, Ro, Agus, Lu, Ale]

  def self.personas_aprobadas
    @estudiantes.select {|estudiante|estudiante.aprobo?}
  end
end
```

▶ Enviar

Ejercicio 9: Ejercicio 9

A la hora de viajar sabemos que las instancias de la clase `Persona` usan su boleto `!!`.
Actualmente tenemos dos tipos de boletos:

- `BoletoDescartable`: solo se pueden usar una vez y tienen un atributo `@usado`;
- `BoletoConCarga`: tiene un `@saldo` que disminuye 29 con cada viaje.

> Definí el método `realizar_viaje!` en la clase `Persona` y el método `usar!` en los distintos tipos de boleto.

Solución | Consola

```
class Persona
  def initialize(un_boleto)
    @boleto = un_boleto
  end

  def realizar_viaje!
    @boleto.usar!
  end
end

class BoletoDescartable
  def initialize()
    @usado = false
  end
  def usar!
    @usado = true
  end
end

class BoletoConCarga
  def initialize(un_saldo)
    @saldo = un_saldo
  end
  def usar!
    @saldo -=29
  end
end
```

Enviar