

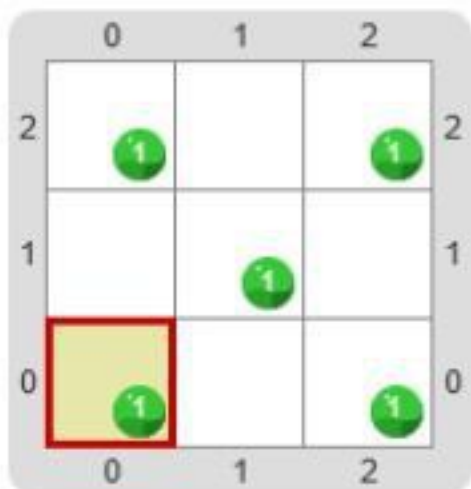
 Te quedan 00:46:53

Ejercicio 1: Ejercicio 1




Es bastante sabido que para recordar dónde se esconde un tesoro hay que marcar el lugar. 🏴‍☠️

Una clásica opción para esto es utilizar una cruz **X**, que en un tablero podría verse así:



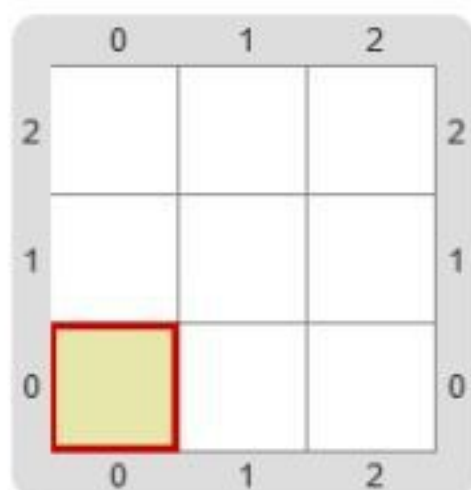
Creá un programa que dibuje una cruz de color Verde. El cabezal empieza en el origen (o sea, en el borde Sur-Oeste) pero no te preocupes por dónde finaliza.

```
1 procedure CruzTablero() {  
2   Poner (Verde)  
3   Mover(Norte)  
4   Mover(Norte)  
5   Poner(Verde)  
6   Mover(Este)  
7   Mover(Sur)  
8   Poner(Verde)  
9   Mover(Sur)  
10  Mover(Este)  
11  Poner(Verde)  
12  Mover(Norte)  
13  Mover(Norte)  
14  Poner(Verde)  
15 }  
16 program {  
17   CruzTablero()  
18 }
```

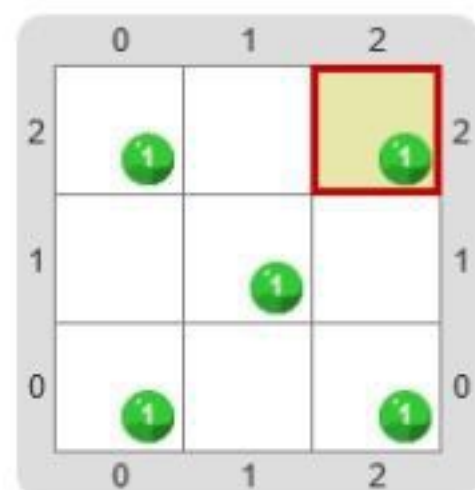
 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



Esta guía fue desarrollada por Gustavo Trucco, Franco Bulgarelli, Rocío Gonzalez bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

Te quedan 00:46:39

Ejercicio 2: Ejercicio 2



La verdad es que en el ejercicio anterior hicimos una cruz de un color específico porque es lo que solemos ver en películas o libros pero ¿qué nos impide que hagamos una cruz de cualquier color para marcar un lugar? 🤔

Definí el procedimiento `DibujarCruz` para que dibuje una cruz con el `color` que reciba por parámetro. No te preocupes por donde termina el cabezal.

```
1 procedure DibujarCruz(color) {
2   Poner(color)
3   Mover(Norte)
4   Mover(Norte)
5   Poner(color)
6   Mover(Este)
7   Mover(Sur)
8   Poner(color)
9   Mover(Sur)
10  Mover(Este)
11  Poner(color)
12  Mover(Norte)
13  Mover(Norte)
14  Poner(color)
15 }
```

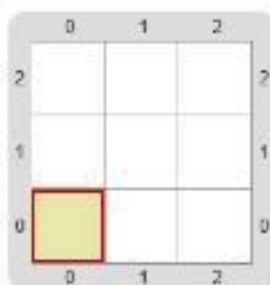
▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

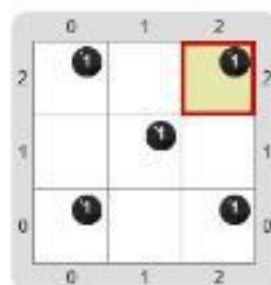
Resultados de las pruebas:



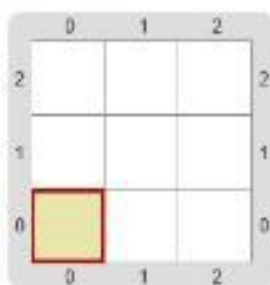
Tablero inicial



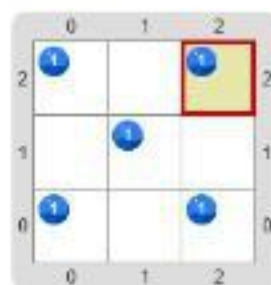
Tablero final



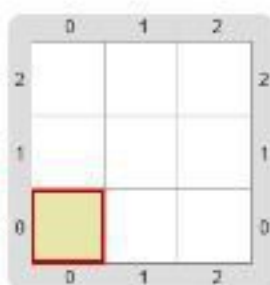
Tablero inicial



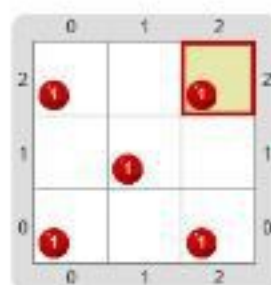
Tablero final



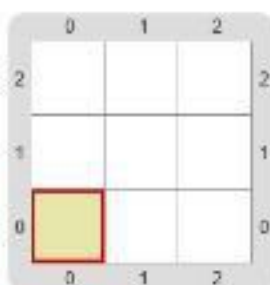
Tablero inicial



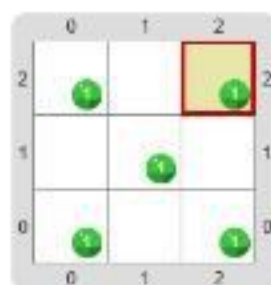
Tablero final



Tablero inicial



Tablero final





🕒 Te quedan 00:46:23

Ejercicio 3: Ejercicio 3

JS

Dejemos atrás los tableros y... ¡Pasemos a JavaScript! 🖱️

A veces la matemática puede ser un poco tediosa 😊. La buena noticia es que ahora podemos crear funciones que nos ayuden a resolver estos problemas. 🧠

Para eso vamos a crear una función que reciba 3 números y nos diga si la resta entre los 2 primeros es mayor al tercero. Por ejemplo:

```
esMasGrandeLaResta(4, 2, 8)
false //Porque 4 menos 2 es 2 y es menor a 8

esMasGrandeLaResta(12, 3, 5)
true //Porque 12 menos 3 es 9 y es mayor a 5
```

Definí la función `esMasGrandeLaResta`.

📌 Solución

> Consola

```
1 function esMasGrandeLaResta(num1, num2, num3) {
2   return num1 - num2 > num3
3 }
```

▶ Enviar

✅ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Gustavo Trucco, Franco Bulgarelli, Rocío Gonzalez bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.



⌚ Te quedan 00:46:07

Ejercicio 4: Ejercicio 4

JS

Ahora vamos a hacer una función un poco particular. 🐼

Queremos crear un mezclador de palabras que reciba 2 palabras y un número. Si el número es menor o igual a 3 el mezclador concatena la primera palabra con la segunda. En cambio, si el número es mayor a 3, concatena la segunda con la primera:

```
mezclarPalabras("planta", "naranja", 3)  
"plantanaranja"
```

```
mezclarPalabras("amor", "amarillo", 2)  
"amoramarillo"
```

```
mezclarPalabras("mate", "pato", 4)  
"patomate"
```

Definí la función `mezclarPalabras`.

Solución > Consola

```
1 function mezclarPalabras(pal1, pal2, num) {  
2   if (num <= 3) {  
3     return pal1 + pal2  
4   } else {  
5     return pal2 + pal1  
6   }  
7 }  
8 }
```

▶ Enviar

✅ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Gustavo Trucco, Franco Bulgarelli, Rocío Gonzalez bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

 Te quedan 00:45:55

Ejercicio 5: Ejercicio 5

JS

Ale está haciendo un trabajo de investigación y nos pidió ayuda 🙏. Necesita poder sumar la cantidad de letras de las palabras cortas +. Una palabra se considera corta si tiene 6 o menos letras. Veamos un ejemplo:

```
sumatoriaDeLetrasDePalabrasCortas(["hola", "murcielago", "caballo", "choclo", "poco", "luz", "sol"])  
20
```

Definí la función `sumatoriaDeLetrasDePalabrasCortas`.

 Solución Consola

```
1 function  
  sumatoriaDeLetrasDePalabrasCortas(palabras) {  
2   let sumatoria = 0;  
3   for (let palabra of palabras) {  
4     if (longitud(palabra) <= 6){  
5       sumatoria += longitud(palabra)  
6     }  
7   }  
8   return sumatoria  
9 }
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Gustavo Trucco, Franco Bulgarelli, Rocío Gonzalez bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).



⌚ Te quedan 00:45:42

Ejercicio 6: Ejercicio 6

JS

Los servicios de películas bajo demanda lograron despertar un interés renovado en la sociedad por el cine y las series 🎬. Es por ello que contamos registros de este estilo:

```
let gus = {
  nick: "Wuisti",
  promedioPeliculasMensuales: 5,
  plataforma: "NetFix"
};

let ariel = {
  nick: "Ari",
  promedioPeliculasMensuales: 10,
  plataforma: "Armazon"
};
```

Ahora debemos definir una función que permita obtener un resumen de la información registrada de manera simple. Por ejemplo:

```
➤ resumenDeInfo(gus)
"Se estima que Wuisti verá 60 películas en un año por NetFix"

➤ resumenDeInfo(ariel)
"Se estima que Ari verá 120 películas en un año por Armazon"
```

Definí la función `resumenDeInfo` que nos permita obtener la información requerida.

✅ ¡Muy bien! Tu solución pasó todas las pruebas

🔍 Solución

> Consola

```
1 function resumenDeInfo(series) {
2   return "se estima que " + series.nick + " verá "
  + series.promedioPeliculasMensuales * 12 + "
  películas en un año por " + series.plataforma
3 }
```

▶ Enviar

Esta guía fue desarrollada por Gustavo Trucco, Franco Bulgarelli, Rocío Gonzalez bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).



 Te quedan 00:45:31

Ejercicio 7: Ejercicio 7



¡Dejemos atrás a JavaScript para pasar a Ruby! 🐘

Vamos a modelar `Camioneta` s para poder:

- cargarle una cantidad de nafta determinada;
- ver si tiene carga suficiente, es decir, si tiene más de 50 litros de nafta.

Definí en Ruby, la clase `Camioneta` que tenga un atributo `@nafta` con su getter. Los autos entienden los mensajes `cargar_nafta!` (que recibe la cantidad a cargar por parámetro) y `carga_suficiente?`. No te olvides de definir un `initialize` que reciba a la nafta inicial como parámetro.

 Solución Consola

```
1 class Camioneta
2
3   def initialize (nafta)
4     @nafta = nafta
5   end
6
7   def nafta
8     @nafta
9   end
10
11  def cargar_nafta!(cargar)
12    @nafta += cargar
13  end
14
15  def carga_suficiente?
16    @nafta > 50
17  end
18
19
20 end
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Gustavo Trucco, Franco Bulgarelli, Rocío Gonzalez bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

 Te quedan 00:45:21

Ejercicio 8: Ejercicio 8



Los compilados son discos que tienen la característica de recopilar canciones que comparten alguna característica, por ejemplo artista, época o género. Algunas de ellas con mayor duración que otras. 🎵

Teniendo en cuenta que las canciones saben responder al mensaje

nombre ...

Definí en Ruby el método `nombres_de_canciones` que responda el nombre de las canciones del `DiscoCompilado`.

[Solución](#)[> Consola](#)

```
1 module DiscoCompilado
2   @canciones = [AmorAusente, Eco, Agujas, ElBalcon,
3                 GuitarrasDeCarton]
4
5   def self.nombres_de_canciones
6     @canciones.map { |canciones| canciones.nombre }
7   end
8 end
```

[▶ Enviar](#)

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Gustavo Trucco, Franco Bulgarelli, Rocío Gonzalez bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

 Te quedan 00:45:10

Ejercicio 9: Ejercicio 9



Como bien sabemos, una `Banda` tiene integrantes. Cuando la banda toca, toca cada integrante:

[Solución](#)[> Consola](#)

- `Guitarrista` pierde una de sus `cuerdas`; 🎸
- `Percusionista` sube su `indice_de_coordinacion` en 13; 🥁
- `Triangulista` no hace nada. 😐

Definí el método `tocar!` tanto en la `Banda` como en los distintos tipos de integrantes. Definí los getters necesarios en cada integrante.

```
1 class Banda
2   def initialize(integrantes)
3     @integrantes = integrantes
4   end
5
6   def integrantes
7     @integrantes
8   end
9
10  def tocar!
11    @integrantes.each{|integrante| integrante.tocar!}
12  end
13 end
14
15 class Guitarrista
16   def initialize(cuerdas)
17     @cuerdas = cuerdas
18   end
19
20   def cuerdas
21     @cuerdas
22   end
23
24   def tocar!
25     @cuerdas -= 1
26   end
27 end
28
29 class Percusionista
30   def initialize(indice_de_coordinacion)
31     @indice_de_coordinacion =
32     indice_de_coordinacion
33   end
34   def indice_de_coordinacion
35     @indice_de_coordinacion
36   end
37   def tocar!
38     @indice_de_coordinacion += 13
39   end
40 end
41 class Triangulista
42   def tocar!
43   end
44 end
```

[▶ Enviar](#)

✓ ¡Muy bien! Tu solución pasó todas las pruebas