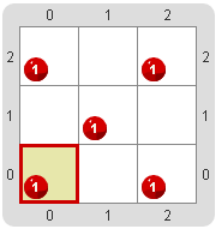
Ejercicio 1: Ejercicio 1

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 , que en un tablero podría verse así:



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

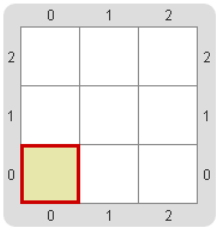
```
program {
  Poner(Rojo)
  IrAlBorde(Norte)
  Poner(Rojo)
  IrAlBorde(Este)
  Poner(Rojo)
  IrAlBorde(Sur)
  Poner(Rojo)
  Mover(Norte)
  Mover(Oeste)
  Poner(Rojo)

}
```

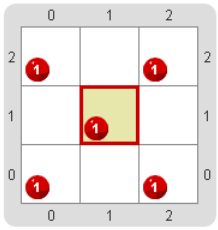
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



Te quedan 01:05:12

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

```
procedure HacerCruz (color) {
  Poner(color)
  IrAlBorde(Norte)
  Poner(color)
  IrAlBorde(Este)
  Poner(color)
  IrAlBorde(Sur)
  Poner(color)
  Mover(Norte)
  Mover(Oeste)
  Poner(color)

}
```

Enviar

Ejercicio 3: Ejercicio 3

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:

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

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

Definí la función `esMayorLaResta`.

Solución | Consola

```
function esMayorLaResta (num1, num2, num3) {
  return num1 - num2 > num3
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 4: Ejercicio 4

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 5 el mezclador concatena la primera palabra con la segunda. En cambio, si el número es mayor a 5, concantena la segunda con la primera:

```
> mezclarPalabras("planta", "naranja", 5)
"plantanaranja"

> mezclarPalabras("amor", "amarillo", 4)
"amoramarillo"

> mezclarPalabras("mate", "pato", 6)
"patomate"
```

Definí la función `mezclarPalabras`.

Solución | Consola

```
function mezclarPalabras (pa1, pa2, num) {
  if (num <= 5) {
    return pa1 + pa2
  } else if (num > 5) {
    return pa2 + pa1
  }
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 5: Ejercicio 5

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:

```
> sumatoriaDeLetrasDeCortas(["hola", "murcielago", "caballo", "choclo", "pu
co", "luz", "sol"])
20
```

Definí la función `sumatoriaDeLetrasDeCortas`.

Solución | Consola

```
function sumatoriaDeLetrasDeCortas (lista) {
  let letrasPalabras = 0;
  for (let palabrascortas of lista) {
    if (longitud(palabrascortas) <= 6) {
      letrasPalabras += (longitud(palabrascortas))
    }

  } return letrasPalabras


}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 6: Ejercicio 6

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:

```
> infoResumida(gus)
"Está estimado que Wuisti verá 60 películas en un año utilizando NetFix"

> infoResumida(ariel)
"Está estimado que Ari verá 120 películas en un año utilizando Armazon"
```

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

Solución | Consola

```
function infoResumida (param) {
  return "Está estimado que " + param.nick + " verá " + param.promedioPeliculasMensuales * 12 + " películas en un año utilizando " + param.plataforma
}
```

Enviar

Ejercicio 7: Ejercicio 7

¡Dejemos atrás a JavaScript para pasar a Ruby!

Vamos a modelar `Auto`s para poder:

- cargarle una cantidad de nafta determinada;
- ver si tiene carga suficiente, es decir, si tiene más de 21 litros de nafta.

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

Solución | Consola

```
class Auto
  def initialize (nafta)
    @nafta =  nafta
  end
  def nafta
    @nafta
  end
  def cargar_combustible! (cantidad)
    @nafta += cantidad
  end
  def suficiente_combustible?
    @nafta > 21
  end

end
```

Enviar

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_las_canciones` que responda el nombre de las canciones del `Disco`.

Solución | Consola

```
module Disco
  @canciones = [AmorAusente, Eco, Agujas, ElBalcon, GuitarrasDeCarton]

  def self.nombres_de_las_canciones
   @canciones.map {|cancion| cancion.nombre}
  end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 9: Ejercicio 9

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

- `Violinista` pierde una de sus `cuerdas`;
- `Pianista` sube su `indice_de_coordinacion` en 15;
- `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.

Solución | Consola

```ruby
class Banda
  def initialize(integrantes)
    @integrantes = integrantes
  end
  def tocar!
   @integrantes.each {|integrante| integrante.tocar!}
  end
end
class Violinista
  def initialize(cuerdas)
    @cuerdas = cuerdas
  end
  def tocar!
    @cuerdas -= 1
  end
  def cuerdas
  @cuerdas
end
end



class Pianista
  def initialize(indice_de_coordinacion)
    @indice_de_coordinacion = indice_de_coordinacion
  end
  def tocar!
    @indice_de_coordinacion += 15
  end
   def indice_de_coordinacion
   @indice_de_coordinacion
 end
end



class Triangulista
  def tocar!
  end
end
```

Enviar