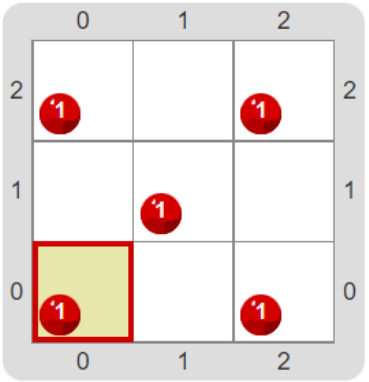
Ejercicio 1: Ejercicio 1

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)
  Mover(Norte)
  Mover(Norte)
  Poner(Rojo)
  Mover(Este)
  Mover(Este)
  Poner(Rojo)
  Mover(Sur)
  Mover(Sur)
  Poner(Rojo)
  Mover(Oeste)
  Mover(Norte)
  Poner(Rojo)
}
```

Enviar

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.

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

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(primero,segundo,tercero){
  return (primero-segundo)>tercero
}
```

Enviar

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 4 el mezclador concatena la primera palabra con la segunda. En cambio, si el número es mayor a 4, concantena la segunda con la primera:

```
> mezclarPalabras("planta", "naranja", 4)
"plantanaranja"

> mezclarPalabras("amor", "amarillo", 3)
"amoramarillo"

> mezclarPalabras("mate", "pato", 5)
"patomate"
```

Definí la función `mezclarPalabras`.

Solución | Consola

```
1 function mezclarPalabras(primera,segunda,numero){
2  if (numero<=4){
3    return (primera+segunda)
4  }
5   else if (numero>4){
6     return (segunda+primera)
7   }
8 }
```

Enviar

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:

```
> sumaLetrasDePalabrasCortas(["hola", "murcielago", "caballo", "cho
clo", "poco", "luz", "sol"])
20
```

Definí la función `sumaLetrasDePalabrasCortas`.

Solución | Consola

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

Enviar

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:

Solución | Consola

```
1 function resumenDeInformacion(nombre){
2   return  ("Está estimado que"+ " "+ nombre.nick+ " "+ "verá"+" "+ nombre.promedioPeliculasMensuales*12 +" "+"películas en un año por la plataforma"+" "+ nombre.plataforma)
3 }
```

Enviar

Ejercicio 7: Ejercicio 7

¡Dejemos atrás a JavaScript para pasar a Ruby!

Vamos a modelar `Moto`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 `Moto` 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_nafta?`. No te olvides de definir un `initialize` que reciba a la nafta inicial como parámetro.

Solución | Consola

```ruby
class Moto
  def initialize(litros)
    @nafta=litros
  end
  def nafta
    @nafta
  end
  def cargar_combustible!(litros)
    @nafta += litros
  end
  def suficiente_nafta?
    @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 `titulo`...

> Definí en Ruby el método `nombres_de_las_canciones` que responda el nombre de las canciones del `Disco`.

Solución | Consola

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

  def self.canciones
    @canciones
  end

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

Enviar

Ejercicio 9: Ejercicio 9

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

- `Bajista` pierde una de sus `cuerdas`;
- `Pianista` sube su `indice_de_coordinacion` en 19;
- `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

```
1  class Banda
2    def initialize(integrantes)
3      @integrantes = integrantes
4    end
5    def integrantes
6      @integrantes
7    end
8    def tocar!
9
   @integrantes.each{|integrante|integrante.tocar!}
10   end
11 end
12
13 class Bajista
14   def initialize(cuerdas)
15     @cuerdas = cuerdas
16   end
17   def cuerdas
18     @cuerdas
```

```ruby
  def cuerdas
    @cuerdas
  end
 def tocar!
    @cuerdas-=1
  end
end

class Pianista
  def initialize(indice_de_coordinacion)
    @indice_de_coordinacion =
indice_de_coordinacion
  end
  def indice_de_coordinacion
    @indice_de_coordinacion
  end
  def tocar!
    @indice_de_coordinacion+=19
  end
end

class Triangulista
 def tocar!
  end
end

```