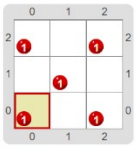
Te quedan 00:51:19

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)
  repeat(2){
    Mover(Norte)
  }
  Poner(Rojo)
  repeat(2){
    Mover(Este)
  }
  Poner(Rojo)
  repeat(2){
    Mover(Sur)
  }
  Poner(Rojo)
  Mover(Oeste)
  Mover(Norte)
  Poner(Rojo)
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:51:04

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)
    repeat(2){
      Mover(Norte)
    }
  Poner(color)
  repeat(2){
    Mover(Este)
  }
  Poner(color)
  repeat(2){
    Mover(Sur)
  }
  Poner(color)
  Mover(Oeste)
  Mover(Norte)
  Poner(color)
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:50:42

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

Te quedan 00:50:29

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"
```

Solución | Consola

```
function mezclarPalabras(palabra1, palabra2, numero){
  if(numero <= 4){
    return palabra1 + palabra2
  } else {
    return palabra2 + palabra1
  }
}
```

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:

```
> resumenDeInformacion(gus)
"Se estima que Wuisti verá 60 películas en un año por la plataforma NetFix"

> resumenDeInformacion(ariel)
"Se estima que Ari verá 120 películas en un año por la plataforma Armazon"
```

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

Solución | Consola

```
function resumenDeInformacion(nomb){
  return "Se estima que" +  " " + nomb.nick + " " + "verá" + " " + nomb.promedioPeliculasMensuales*12 + " " + "películas en un año por la plataforma" + " " + nomb.plataforma;
}
```

▶ Enviar

✔ ¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:50:13

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", "choclo", "poc
o", "luz", "sol"])
20
```

Definí la función `sumaLetrasDePalabrasCortas`.

Solución | Consola

```
function sumaLetrasDePalabrasCortas(l){
  let sumatoria = 0
  for (let palabra of l){
    if (longitud(palabra)<= 6){
      sumatoria += longitud(palabra)
    }
    }
   return sumatoria
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:49:45

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 24 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

```
class Moto
  def initialize (nafta)
    @nafta = nafta
  end
  def nafta
    @nafta
  end
  def cargar_combustible!(litros)
    @nafta += litros
  end
  def suficiente_nafta?
    return @nafta > 24
  end
end

```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:49:34

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 `Compilado`.

Solución | Consola

```ruby
module Compilado
  @canciones = [AmorAusente, Eco, Agujas, ElBalcon, GuitarrasDeCarton]
  def self.nombres_de_las_canciones
    @canciones.map{|cancion|cancion.titulo}
  end
end
```

Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

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

- `Bajista` pierde una de sus `cuerdas`;
- `Baterista` sube su `indice_de_coordinacion` en 35;
- `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 Bajista
  def initialize(cuerdas)
    @cuerdas = cuerdas
  end
  def cuerdas
    @cuerdas
  end
  def tocar!
    @cuerdas -=1
  end
end

class Baterista
  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 = @indice_de_coordinacion + 35
  end
end

class Triangulista
  def tocar!
  end
end
```