Ejercicios examen sé programar

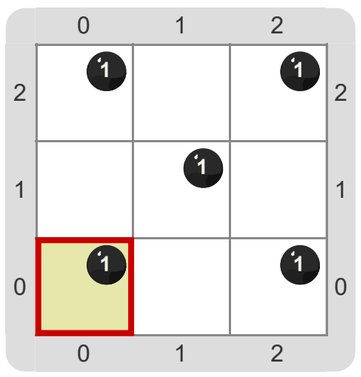
Lenguajes de Programación (Universidad de Buenos Aires)

Te quedan 00:02:42

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:



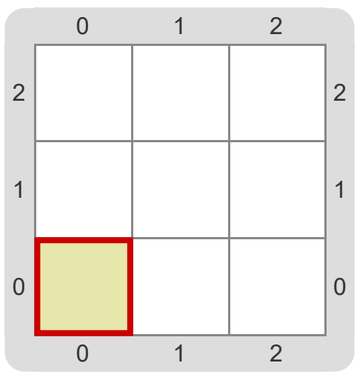
Creá un programa que dibuje una cruz de color `Negro`. El cabezal empieza en el origen (o sea, en el borde Sur-Oeste) pero no te preocupes por dónde finaliza.

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

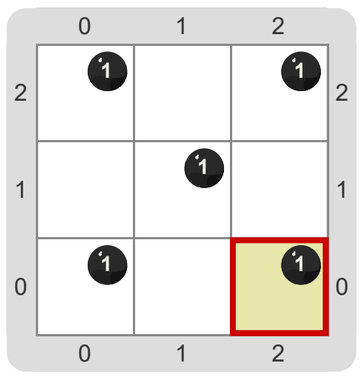
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



Siguiente Ejercicio: Ejercicio 7 ›

Te quedan 00:02:47

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.

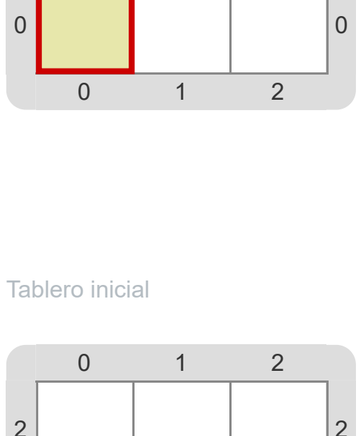
```
procedure DibujarCruz(color) {
  Poner(color)
  repeat(2) {
    Mover(Norte)
  }
  Poner(color)
  repeat(2) {
    Mover(Este)
  }
  Poner(color)
  Mover(Sur)
  Mover(Oeste)
  Poner(color)
  Mover(Este)
  Mover(Sur)
  Poner(color)
}
```

Enviar

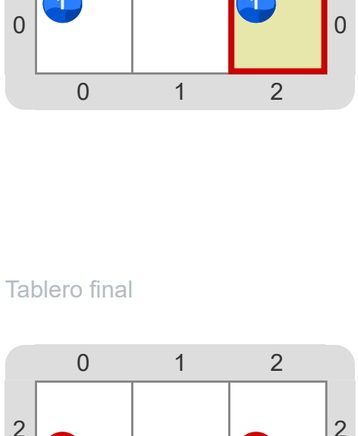
¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

Tablero inicial



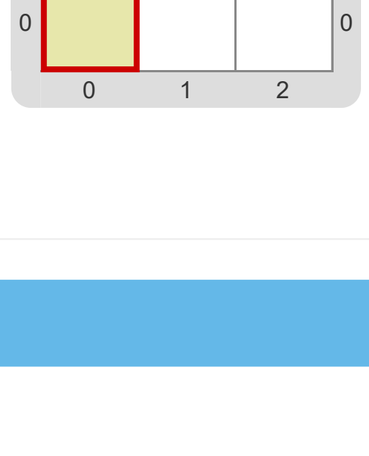
Tablero final



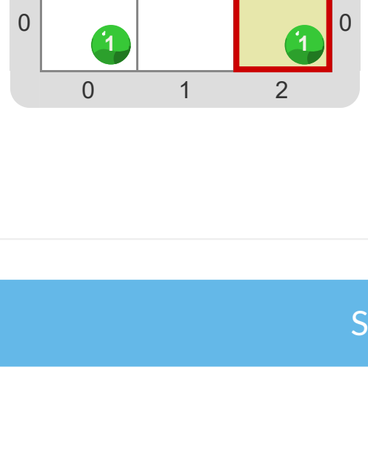
Tablero inicial

Tablero final

Tablero inicial



Tablero final



Tablero inicial

Tablero final

Siguiente Ejercicio: Ejercicio 7 ›

Te quedan 00:02:53

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:

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

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

Definí la función `laRestaEsMayor`.

Solución | Consola

```
function laRestaEsMayor(n1, n2, n3) {
  return n1-n2 > n3;
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 7 >

Te quedan 00:02:58

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 8 el mezclador concatena la primera palabra con la segunda. En cambio, si el número es mayor a 8, concantena la segunda con la primera:

```
> mezcladorDePalabras("planta", "naranja", 8)
"plantanaranja"

> mezcladorDePalabras("amor", "amarillo", 7)
"amoramarillo"

> mezcladorDePalabras("mate", "pato", 9)
"patomate"
```

Definí la función `mezcladorDePalabras`.

Solución | Consola

```
function mezcladorDePalabras(p1, p2, n) {
  if (n <= 8) {
    return p1+p2;
  } if (n > 8) {
    return p2+p1;
  }
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 7 >

Te quedan 00:03:03

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:

```
> sumaDeLetrasDePalabrasCortas(["hola", "murcielago", "caballo", "c
hoclo", "poco", "luz", "sol"])
20
```

Definí la función `sumaDeLetrasDePalabrasCortas`.

Solución | Consola

```
function sumaDeLetrasDePalabrasCortas(palabras) {
  let sumatoria = 0;
  for(let palabra of palabras) {
    if (longitud(palabra) <= 6) {
      sumatoria = sumatoria + longitud(palabra)
    }
  }
  return sumatoria;
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 7 >

Te quedan 00:03:09

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:

```
> resumenInformacion(gus)
"Está estimado que Wuisti verá 60 películas en un año por la plataf
orma NetFix"

> resumenInformacion(ariel)
"Está estimado que Ari verá 120 películas en un año por la platafor
ma Armazon"
```

> Definí la función `resumenInformacion` que nos permita obtener la información requerida.

Solución | Consola

```
function resumenInformacion(persona) {
  return "Está estimado que " + persona.nick + "
verá " + persona.promedioPeliculasMensuales*12 + "
películas en un año por la plataforma " +
persona.plataforma;
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 7 >

Te quedan 00:03:15

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 39 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 `nafta_suficiente?`. No te olvides de definir un `initialize` que reciba a la nafta inicial como parámetro.

Solución | Consola

```ruby
class Camioneta
  def initialize
   @nafta = litros
  end
  def self.nafta
    @nafta
  end
  def self.cargar_nafta!(litros)
    @nafta += litros
  end
  def self.nafta_suficiente?
    @nafta > 39
  end
end

```

Enviar

Tu solución no pasó las pruebas

Resultados de las pruebas:

- Si creo una instancia de Camioneta le puedo especificar su nafta inicial Ver detalles
- Una instancia de Camioneta suma 20 litros de nafta al enviarle cargar_nafta!(20) Ver detalles
- Una instancia de Camioneta suma 5 litros de nafta al enviarle cargar_nafta!(5) Ver detalles
- Una instancia de Camioneta suma 100 litros de nafta al enviarle cargar_nafta!(100) Ver detalles
- Una instancia de Camioneta no tiene carga suficiente si tiene 39 litros de nafta Ver detalles
- Una instancia de Camioneta no tiene carga suficiente si tiene menos de 39 litros de nafta Ver detalles
- Una instancia de Camioneta tiene carga suficiente si tiene más de 39 litros de nafta Ver detalles

Te quedan 00:03:21

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 `titulo`...

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

Solución | Consola

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

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente pendiente: Ejercicio 7 >

Te quedan 00:03:28

Ejercicio 9: Ejercicio 9

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

- `Bajista` pierde una de sus `cuerdas`;
- `Saxofonista` sube su `indice_de_coordinacion` en 34;
- `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 Saxofonista
  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 += 34
  end
end

class Triangulista
  def tocar!
  end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente pendiente: Ejercicio 7 >