Examen AP Tema1

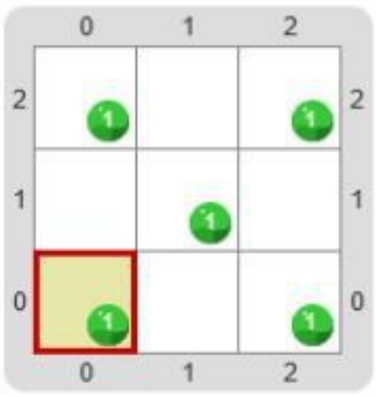
Taller de Programación I (Universidad de Buenos Aires)

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 ❌, que en un tablero podría verse así:



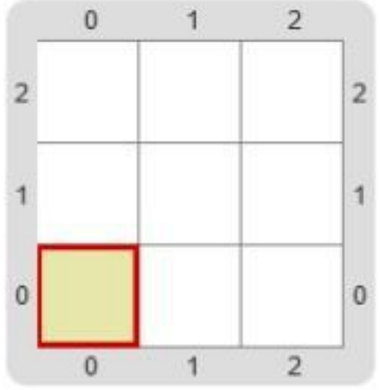
Creá un programa que dibuje una cruz de color `Verde`. El cabezal empieza en el origen (o sea, en el borde Sur-Oeste) pero no te preocupes por dónde finaliza.

```
procedure CruzTablero() {
  Poner (Verde)
  Mover(Norte)
  Mover(Norte)
  Poner(Verde)
  Mover(Este)
  Mover(Sur)
  Poner(Verde)
  Mover(Sur)
  Mover(Este)
  Poner(Verde)
  Mover(Norte)
  Mover(Norte)
  Poner(Verde)
}
program {
  CruzTablero()
  }
```

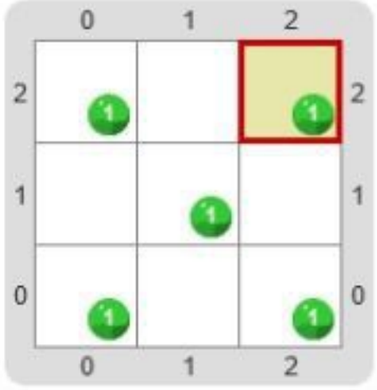
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



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.

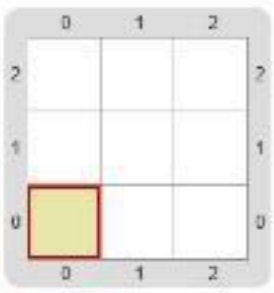
```
procedure DibujarCruz(color) {
  Poner(color)
  Mover(Norte)
  Mover(Norte)
  Poner(color)
  Mover(Este)
  Mover(Sur)
  Poner(color)
  Mover(Sur)
  Mover(Este)
  Poner(color)
  Mover(Norte)
  Mover(Norte)
  Poner(color)
  }
```

Enviar

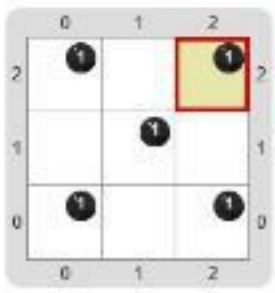
¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

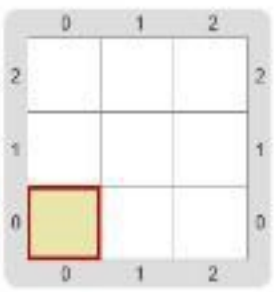
Tablero inicial



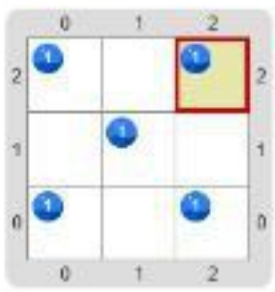
Tablero final



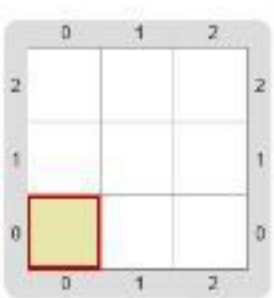
Tablero inicial



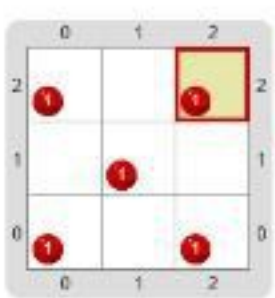
Tablero final



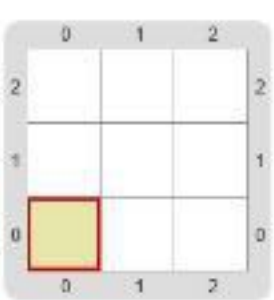
Tablero inicial



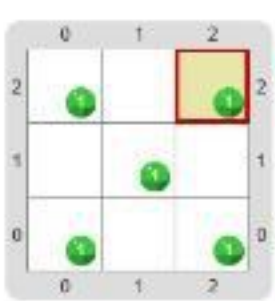
Tablero final



Tablero inicial



Tablero final



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

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

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

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, concantena 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

```
function mezclarPalabras(pal1, pal2, num) {
  if (num <= 3) {
    return pal1 + pal2
  } else {
    return pal2 + pal1
          }

}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:45:55

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:

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

> Definí la función `sumatoriaDeLetrasDePalabrasCortas`.

Solución | Consola

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

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:45:42

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:

```
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.

Solución | Consola

```
function resumenDeInfo(series) {
  return "se estima que " + series.nick + " vera "
+ series.promedioPeliculasMensuales * 12 + "
peliculas en un año por " + series.plataforma
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

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

```ruby
class Camioneta

  def initialize (nafta)
    @nafta = nafta
  end

  def nafta
    @nafta
  end

  def cargar_nafta!(cargar)
    @nafta += cargar
  end

  def carga_suficiente?
    @nafta > 50
  end


end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

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

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

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

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:45:10

Ejercicio 9: Ejercicio 9

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

- `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.

Solución | Consola

```ruby
class Banda
  def initialize(integrantes)
    @integrantes = integrantes
  end

  def integrntes
    @integrantes
  end

  def tocar!
   @integrantes.each{|integrante|integrante.tocar!}
  end
end

class Guitarrista
  def initialize(cuerdas)
    @cuerdas = cuerdas
  end

  def cuerdas
    @cuerdas
  end

  def tocar!
    @cuerdas -= 1
  end
end

class Percusionista
  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 += 13
  end
end

class Triangulista
  def tocar!
  end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas