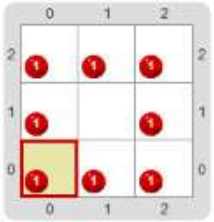
Te quedan 01:57:42

Ejercicio 1: Ejercicio 1

Ale nos regaló una hermosa pintura para que colguemos en nuestra pared. Para eso vamos a tener que ponerle un marco el cual pintaremos de Rojo:



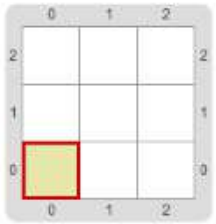
Creá un programa que pinte de `Rojo` los bordes del marco. El cabezal empieza en el origen (o sea, en el borde Sur-Oeste) pero no te preocupes por dónde finaliza.

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

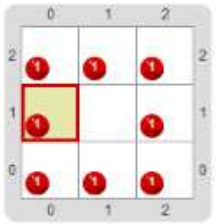
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



Te quedan 01:54:11

Ejercicio 2: Ejercicio 2

El color negro del marco del ejercicio anterior no combinó mucho con la pintura . ¡Hagamos algo para poder probar como quedaría con cualquier color!

> Definí el procedimiento `DarColorAlMarco` para que pinte el marco con el `color` que tome por parámetro. No te preocupes por donde termina el cabezal.

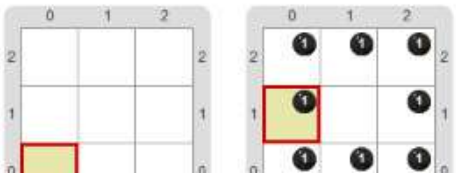
```
procedure DarColorAlMarco(color){
  Poner(color)
  Mover(Este)
  Poner(color)
  Mover(Este)
  Poner(color)
  Mover(Norte)
  Poner(color)
  Mover(Norte)
  Poner(color)
  Mover(Oeste)
  Poner(color)
  Mover(Oeste)
  Poner(color)
  Mover(Sur)
  Poner(color)
}

```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:



Te quedan 01:51:31

Ejercicio 3: Ejercicio 3

Vamos a un maravilloso mundo... ¡el de la matemática!

Los números se pueden operar y comparar. Nada nos impide hacer un poco de ambas al mismo tiempo.

Para eso vamos a crear una función que reciba 3 números y nos diga si la suma de los 2 primeros es menor al tercero. Por ejemplo:

```
> esMenorLaSuma(2, 4, 8)
true //Porque 6 es menor que 8

> esMenorLaSuma(3, 5, 7)
false //Porque 8 es mayor a 7
```

Definí la función `esMenorLaSuma`.

Solución | Consola

```
1 function esMenorLaSuma (nro1, nro2, nro3){
2   return nro1+nro2<nro3
3 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 4 >

Examen #SeProgramar - Mayo 2022

https://seprogramar.inti.gob.ar/inti/exercises/472-examen-seprogramar-mayo-2022-t1-ejercicio-4

Streaming | Claro Comercios | radios | Proveedores | proveedores | Can You RUN It | Campus | SICU | Genshin | Colectivos | Canva | curso | imagenes p ML | descarga foto inst | mapa genshin opoficial | sube | (16) Tejido a dos aguja...

Te quedan 01:35:52

Ejercicio 4: Ejercicio 4

JS

Si hay algo que a Ale le molesta es o pasar frío o abrigarse de más . Pero lo que si sabe, más allá de la temperatura, es de qué color vestirse ese día. Para eso, pensó en una función que recibe una temperatura y un color y responde qué ropa de ese color ponerse. Si la temperatura es 23 grados o más, se pone una remera de ese color. Si no, se pone una campera de ese color:

```
> vestirseAcorde(23, "negra")
"Remera negra"

> vestirseAcorde(22, "verde")
"Campera verde"

> vestirseAcorde(24, "violeta")
"Remera violeta"
```

Definí la función `vestirseAcorde`.

Solución | Consola

```
function vestirseAcorde(temperatura,color){
  if (temperatura>=23){
    return "Remera "+ color;
  }
  else{
    return "Campera "+ color;
  }

}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 5 >

(1) WhatsApp - Mozill... | Examen #SeProgram... | ESP 09:25

Te quedan 01:31:25

Ejercicio 5: Ejercicio 5

Para quienes no suelen leer, la concentración puede variar cuando aparecen palabras largas . Para filtrarlas vamos a crear una función que dada una lista de palabras nos devuelva una lista nueva con las que tengan más de 6 caracteres.

```
> obtenerPalabrasMayores(["jarra", "polilla", "caracol", "gato", "provincia"])
["polilla", "caracol", "provincia"]
```

Definí la función `obtenerPalabrasMayores`.

Solución | Consola

```
function obtenerPalabrasMayores(palabras){
  let palabrasFiltradas = []
  for (let palabra of palabras){
    if (longitud(palabra)>6){
      agregar (palabrasFiltradas,palabra)
    }
  }
  return palabrasFiltradas
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 6 >

Te quedan 01:25:08

Ejercicio 6: Ejercicio 6

JS

En una biblioteca guardan registro de todos los libros leídos por las personas que la concurren. Estos registros tienen la siguiente forma:

```
let juan = {
  nombre: "Juan Arrever",
  librosLeidos: ["El conde de Montecristo", "La palabra", "Mi planta de naranja lima"],
  anioSuscripcion: 1992
};

let elena = {
  nombre: "Elena Chalver",
  librosLeidos: ["Rabia", "Vida de Bob Marley"],
  anioSuscripcion: 1987
};
```

Ahora debemos definir una función que permita obtener un resumen de la información registrada de manera simple. Por ejemplo:

```
> resumenInformacion(juan)
"Juan Arrever tiene suscripción hace 28 años y leyó 3 obras"

> resumenInformacion(elena)
"Elena Chalver tiene suscripción hace 33 años y leyó 2 obras"
```

> Definí la función `resumenInformacion` que nos permita obtener la información requerida. Asumí que estamos en 2021.

Solución | Consola

```
function resumenInformacion(persona){
  return persona.nombre+" tiene suscripción hace "+(2021-persona.anioSuscripcion)+" años y leyó "+longitud(persona.librosLeidos)+" obras"
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 7 >

Te quedan 01:19:09

Ejercicio 7: Ejercicio 7

¡Dejemos atrás a JavaScript para pasar a Ruby!

Vamos a modelar `Comida`s para poder:

- agregarle cucharadas de sal;
- ver si tiene demasiada sal, es decir, si tiene más de 8 cucharadas de sal.

> Definí en Ruby, la clase `Comida` que tenga un atributo `@cucharadas_sal` con su getter. Las instancias de esta clase entienden los mensajes `sumar_cucharadas!` (que recibe la cantidad a agregar por parámetro) y `demasiada_sal?`. No te olvides de definir un `initialize` que reciba las cucharadas de sal iniciales como parámetro.

Solución | >_ Consola

```ruby
class Comida

  def initialize(cantidad)
    @cucharadas_sal = cantidad
  end

  def cucharadas_sal
    @cucharadas_sal
  end

  def sumar_cucharadas!(cantidad)
    @cucharadas_sal+=cantidad
  end

  def demasiada_sal?
    @cucharadas_sal > 8
  end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 8 >

Te quedan 01:13:02

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 `corta?`...

> Definí en Ruby el método `cantidad_de_cortas` que responda a cuántas canciones cortas tiene el `DiscoCompilado`.

Solución | Consola

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

def self.cantidad_de_cortas
  @canciones.count {|cancion| cancion.corta?}
end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Siguiente Ejercicio: Ejercicio 9 >

Ejercicio 9: Ejercicio 9

A la hora de relajarse muchas `Persona`s juegan con su mascota. Los animales hacen distintas cosas cuando juegan:

- A los `Carpincho`s les da hambre;
- los `Gato`s incrementan en 7 su nivel de felicidad;
- las `Tortuga`s no hacen nada.

> Definí el método `jugar_con_mascota!` en la clase `Persona` y el método `jugar!` en los distintos tipos de animales. Definí los getters necesarios en cada una.

Solución | Consola

```ruby
class Persona
  def initialize(mascota)
    @mascota = mascota
  end
  def jugar_con_mascota!
    @mascota.jugar!
  end
end

class Carpincho
  def initialize()
    @tiene_hambre = false
  end
  def tiene_hambre
    @tiene_hambre
  end
  def jugar!
    @tiene_hambre=true
  end
end

class Gato
  def initialize(nivel_de_felicidad)
    @nivel_de_felicidad = nivel_de_felicidad
  end
  def nivel_de_felicidad
    @nivel_de_felicidad
  end
  def jugar!
    @nivel_de_felicidad+=7
  end
end

class Tortuga
  def jugar!
  end
end
```

Enviar