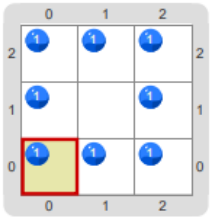
EJERCICIOS DIA 16 A LAS 9 HS DE LA MAÑANA

UNO DE LOS TEMAS:

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 Azul:



Creá un programa que pinte de `Azul` 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(Azul)
  Mover(Norte)
  Poner(Azul)
  Mover(Norte)
  Poner(Azul)
  Mover(Este)
  Poner(Azul)
  Mover(Este)
  Poner(Azul)
  Mover(Sur)
  Poner(Azul)
  Mover(Sur)
  Poner(Azul)
  Mover(Oeste)
  Poner(Azul)


}
```

Enviar

Te quedan 00:59:00

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 `ColorearMarco` para que pinte el marco con el `color` que tome por parámetro. No te preocupes por donde termina el cabezal.

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

```

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:

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

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

Definí la función `laSumaEsMasChica`.

Solución | Consola

```
function laSumaEsMasChica(a, b, c){
  return(a+b) < c
}
```

Enviar

Ejercicio 4: Ejercicio 4

Si hay algo que a Ale le molesta es o pasar frío o abrigarse de más . Pero lo que sí 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 18 grados o más, se pone una remera de ese color. Si no, se pone una campera de ese color:

```
> seleccionarVestimenta(18, "negra")
"Remera negra"

> seleccionarVestimenta(17, "verde")
"Campera verde"

> seleccionarVestimenta(19, "violeta")
"Remera violeta"
```

Solución | Consola

```
function seleccionarVestimenta (temp, color){

  if (temp >= 18){
    return "Remera " + color
  } else {
    return "Campera "+ color
  }
}
```

Enviar

Te quedan 00:58:33

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.

```
> filtrarLargas(["jarra", "polilla", "caracol", "gato", "provinci
a"])
["polilla", "caracol", "provincia"]
```

Definí la función `filtrarLargas`.

Solución | Consola

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

Enviar

Ejercicio 6: Ejercicio 6

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 plant
a 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:

Solución | Consola

```
function resumenDeLaSuscripcion(persona){
  return persona.nombre + " se suscribio hace " +
(2021 -  persona.anioSuscripcion) + " años y leyó "
+ longitud(persona.librosLeidos) + " ejemplares"
}
```

Enviar

Te quedan 00:58:18

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 5 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 `agregar_cucharadas!` (que recibe la cantidad a agregar por parámetro) y `exceso_de_sal?`. No te olvides de definir un `initialize` que reciba las cucharadas de sal iniciales como parámetro.

Solución | Consola

```ruby
class Comida
  def initialize(cant)
    @cucharadas_sal = cant
  end
  def cucharadas_sal
    @cucharadas_sal
  end
  def agregar_cucharadas!(cantidad)
    @cucharadas_sal += cantidad
  end

  def exceso_de_sal?
    return @cucharadas_sal > 5
  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 `demasiado_corta?` ...

> Definí en Ruby el método `cuantas_canciones_cortas` que responda a cuántas canciones cortas tiene el `Compilado`.

Solución | Consola

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

  def self.cuantas_canciones_cortas
    @canciones.count {|cancion| cancion.demasiado_corta?}
  end

end
```

Enviar

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 `Perro`s les da hambre;
- los `Conejo`s incrementan en 4 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

```
class Persona
  def initialize(mascota)
    @mascota = mascota
  end

  def jugar_con_mascota!
    @mascota.jugar!
  end
end

class Perro
  def initialize()
    @tiene_hambre = false
  end

  def tiene_hambre
    @tiene_hambre
  end

  def jugar!
    @tiene_hambre = true
  end
end

```

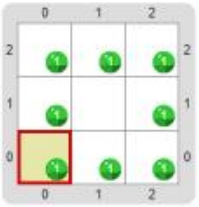
```
  def tiene_hambre
    @tiene_hambre
  end

  def jugar!
    @tiene_hambre = true
  end
end

class Conejo
  def initialize(nivel_de_felicidad)
    @nivel_de_felicidad = nivel_de_felicidad
  end
  def jugar!
    @nivel_de_felicidad += 4
  end
  def nivel_de_felicidad
    @nivel_de_felicidad
  end
end

class Tortuga
  def jugar!

  end
end

```

Enviar

UNO DE LOS TEMAS

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 Verde:



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

```
procedure CuadroVerde(){
  Poner(Verde)
  Mover(Norte)
  Poner(Verde)
  Mover(Norte)
  Poner(Verde)
  Mover(Este)
  Poner(Verde)
  Mover(Este)
  Poner(Verde)
  Mover(Sur)
  Poner(Verde)
  Mover(Sur)
  Poner(Verde)
  Mover(Oeste)
  Poner(Verde)
}

program {

  CuadroVerde()
}
```

▶ Enviar

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 `ColorearMarco` para que pinte el marco con el `color` que tome por parámetro. No te preocupes por donde termina el cabezal.

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

```

▶ Enviar

Te quedan 00:50:31

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:

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

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

Definí la función `esMasChicaLaSuma`.

Solución | Consola

```
function esMasChicaLaSuma(num1,num2,num3){
  let suma = num1 + num2

  return suma < num3
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:49:52

Ejercicio 4: Ejercicio 4

Si hay algo que a Ale le molesta es o pasar frío o abrigarse de más . Pero lo que sí 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 24 grados o más, se pone una remera de ese color. Si no, se pone una campera de ese color:

```
> seleccionarVestimenta(24, "negra")
"Remera negra"

> seleccionarVestimenta(23, "verde")
"Campera verde"

> seleccionarVestimenta(25, "violeta")
"Remera violeta"
```

Definí la función `seleccionarVestimenta`.

Solución | Consola

```
function seleccionarVestimenta(temp,color){

  if(temp >= 24){
    return "Remera " + color
  }

  else {
    return "Campera " + color
  }

}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 00:49:33

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.

```
> filtrarLargas(["jarra", "polilla", "caracol", "gato", "provinci
a"])
["polilla", "caracol", "provincia"]
```

Definí la función `filtrarLargas`.

Solución | Consola

```
function filtrarLargas (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

Ejercicio 6: Ejercicio 6

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 plant
a 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:

```
> resumenDeLaSuscripcion(juan)
"Juan Arrever se registró hace 28 años y leyó 3 ejemplares"

> resumenDeLaSuscripcion(elena)
"Elena Chalver se registró hace 33 años y leyó 2 ejemplares"
```

Solución | Consola

```
function resumenDeLaSuscripcion(persona){
  return persona.nombre + " se registrò hace " +
(2021 -  persona.anioSuscripcion) + " años y leyò "
+ longitud(persona.librosLeidos) + " ejemplares"
}
```

Enviar

Ejercicio 7: Ejercicio 7

¡Dejemos atrás a JavaScript para pasar a Ruby!

Vamos a modelar `Almuerzo`s para poder:

- agregarle cucharadas de sal;
- ver si tiene demasiada sal, es decir, si tiene más de 5 cucharadas de sal.

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

Solución | Consola

```ruby
class Almuerzo
  def initialize(cant)
    @cucharadas_sal = cant
  end

  def cucharadas_sal
    @cucharadas_sal
  end

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

  def exceso_de_sal?
    return @cucharadas_sal > 5
  end

end
```

Enviar

Te quedan 00:48:35

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 `demasiado_corta?`...

> Definí en Ruby el método `cantidad_cortas` que responda a cuántas canciones cortas tiene el `Disco`.

Solución | Consola

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

def self.cantidad_cortas
  @canciones.count {|cancion| cancion.demasiado_corta?}
end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

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 `Hamster`s les da hambre;
- los `Conejo`s incrementan en 8 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

```
class Persona
  def initialize(mascota)
    @mascota = mascota
  end

  def jugar_con_mascota!
    @mascota.jugar!
  end
end

class Hamster
  def initialize()
    @tiene_hambre = false
  end

  def tiene_hambre
    @tiene_hambre
  end

  def jugar!
    @tiene_hambre = true
  end
end

class Conejo
  def initialize(nivel_de_felicidad)
    @nivel_de_felicidad = nivel_de_felicidad
  end
```



```

class Conejo
  def initialize(nivel_de_felicidad)
    @nivel_de_felicidad = nivel_de_felicidad
  end
  def jugar!
    @nivel_de_felicidad += 8
  end
  def nivel_de_felicidad
    @nivel_de_felicidad
  end
end

class Tortuga
  def jugar!

  end
end
```

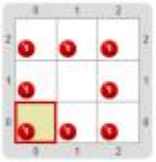
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

UNO DE LOS TEMAS

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

Te quedan 01:55:37

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 01:54:06

Ejercicio 3: Ejercicio 3

JS

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

```
function esMenorLaSuma(n1,n2,n3){
  return (n1 + n2) < n3
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 01:51:04

Te quedan 01:51:04

Ejercicio 4: Ejercicio 4

JS

Si hay algo que a Ale le molesta es o pasar frío o abrigarse de más. Pero lo que sí 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 20 grados o más, se pone una remera de ese color. Si no, se pone una campera de ese color:

```
> vestirseAcorde(20, "negra")
"Remera negra"

> vestirseAcorde(19, "verde")
"Campera verde"

> vestirseAcorde(21, "violeta")
"Remera violeta"
```

Definí la función `vestirseAcorde`.

Solución | Consola

```
function vestirseAcorde (temp,color){
  if(temp >= 20){
    return "Remera "+color
  }else{
    return "Campera "+color
  }
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 01:40:11

Ejercicio 5: Ejercicio 5

JS

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.

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

Definí la función `palabrasMayores`.

Solución | Consola

```
function palabrasMayores(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

Te quedan 01:36:40

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:

```
> resumenDeSuscripcion(juan)
"Juan Arrever se registró hace 28 años y leyó 3 libros"

> resumenDeSuscripcion(elena)
"Elena Chalver se registró hace 33 años y leyó 2 libros"
```

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

Solución | Consola

```
function resumenDeSuscripcion(persona){
  return persona.nombre+" se registró hace "+(2021 - persona.anioSuscripcion)+" años y leyó "+longitud(persona.librosLeidos)+" libros"
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

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 4 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 `poner_cucharadas!` (que recibe la cantidad a agregar por parámetro) y `mucha_sal?`. No te olvides de definir un `initialize` que reciba las cucharadas de sal iniciales como parámetro.

Solución | Consola

```
class Comida
  def initialize(cant)
  @cucharadas_sal = cant
  end
  def cucharadas_sal
    @cucharadas_sal
  end
def poner_cucharadas!(cantidad)
  @cucharadas_sal += cantidad
end

    def mucha_sal?
      return @cucharadas_sal > 4
    end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 01:20:22

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 `demasiado_corta?`...

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

Solución | Consola

```ruby
module Disco
  @canciones = [AmorAusente, Eco, Agujas, ElBalcon, GuitarrasDeCarton]
  def self.cantidad_de_cortas
   @canciones.count {|cancion| cancion.demasiado_corta?}
  end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 01:15:02

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 `Hamster`s les da hambre;
- los `Conejo`s incrementan en 3 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.

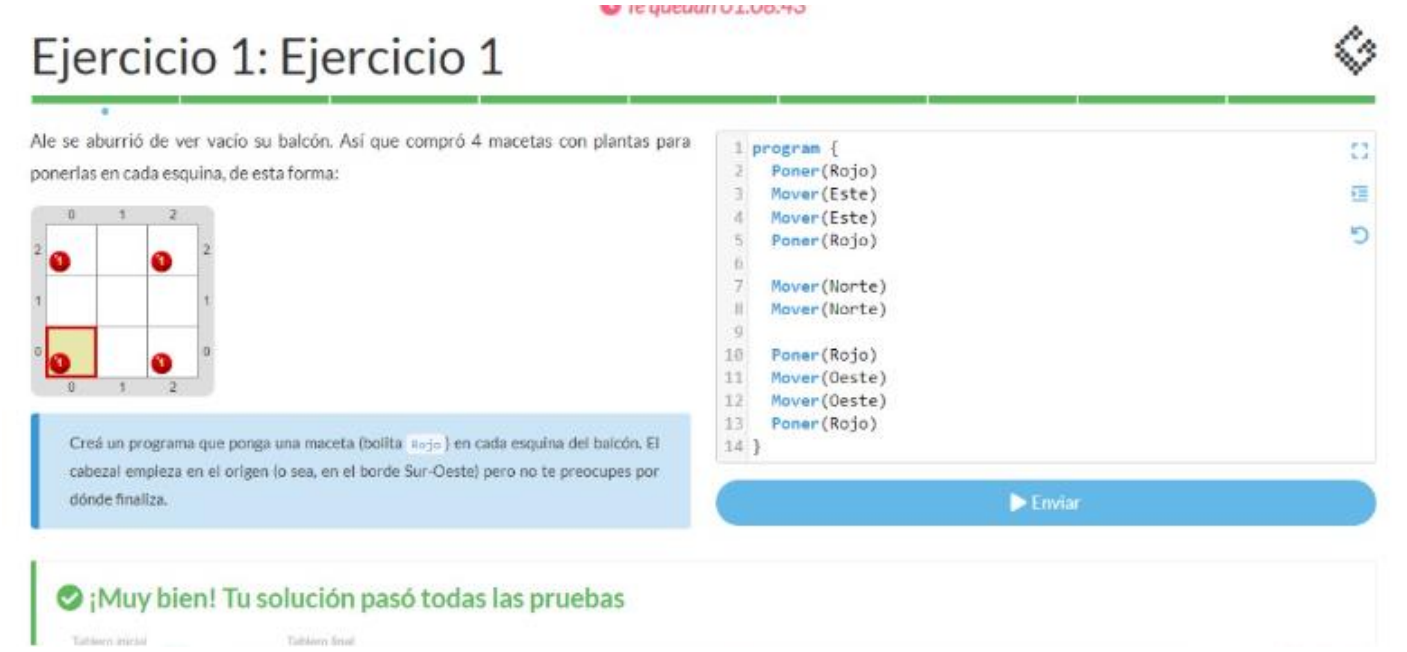
Solución | Consola

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

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

class Conejo
  def initialize(nivel_de_felicidad)
    @nivel_de_felicidad = nivel_de_felicidad
  end
  def jugar!
@nivel_de_felicidad += 3
  end
  end
    def nivel_de_felicidad
@nivel_de_felicidad
  end
end

class Tortuga
  def jugar!

  end
end
```

Enviar

Ejercicios 16 A LAS 2 DE LA TARDE

UNO DE LOS TEMAS:



Te quedan 01:08:33

Ejercicio 2: Ejercicio 2

A Ale no le gustó como quedó ese color en las macetas . ¡Programemos algo para poder probar como quedaría con cualquier color!

Definí el procedimiento `PonerMacetas` para que ponga macetas del `color` que reciba por parámetro en cada esquina. No te preocupes por donde termina el cabezal.

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

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

Te quedan 01:08:25

Ejercicio 3: Ejercicio 3

Dejemos atrás los tableros y... ¡Pasemos a JavaScript!

Cuando hacemos un examen de matemáticas es común verificar más de una vez si las cuentas que hicimos están bien.

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

```
> esCorrectaLaMultiplicacion(2, 4, 8)
true //Porque 2 por 4 es igual a 8

> esCorrectaLaMultiplicacion(3, 5, 12)
false //Porque 3 por 5 es 15, no 12
```

Definí la función `esCorrectaLaMultiplicacion`.

Solución | Consola

```
function esCorrectaLaMultiplicacion(num1, num2, num3){
  return (num1*num2) === num3
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 01:08:17

Ejercicio 4: Ejercicio 4

Ya pasamos por el tesoro de la matemática. Otro bien preciado es el tiempo . Es por ello que tratamos de usarlo sabiamente.

Con esto en mente vamos a crear una función que dados un nombre y un apodo nos diga cuál de los dos es más corto.

```
> cualEsMasCorto("Luis", "Lucho")
"Luis"

> cualEsMasCorto("Carolina", "Caro")
"Caro"

> cualEsMasCorto("Ricardo", "Ringo")
"Ringo"
```

Definí la función `cualEsMasCorto`.

Solución | >_ Consola

```
function cualEsMasCorto(nombre, apodo){
  if(longitud(nombre)>longitud(apodo)){
    return apodo
  }
  else {
    return nombre
  }
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 01:08:07

Ejercicio 5: Ejercicio 5

Vale está haciendo un trabajo de investigación y nos pidió ayuda . Necesita poder distinguir las palabras más cortas de una oración . Una palabra se considera corta si tiene menos de 5 letras. Veamos un ejemplo:

```
> palabrasCortas(["Hari", "Seldon", "nacido", "en", "el", "año", "11988", "d
e", "la", "Era", "Galáctica"])

["Hari", "en", "el", "año", "de", "la", "Era"]
```

Definí la función `palabrasCortas`.

Solución | >_ Consola

```
function palabrasCortas(palabras){
  let listaPalabras= []
  for (let palabra of palabras){
    if (longitud(palabra) < 5){
      agregar(listaPalabras,palabra);
    }
  }
  return listaPalabras
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ale estudia Historia y pensó en crear una función que le ayude a hacer resúmenes . Para eso, consiguió registros de hechos históricos con la siguiente forma:

```
let independenciaArgentina = {
  suceso: "La declaración de la independencia de Argentina",
  anio: 1816,
  ciudad: "San Miguel de Tucumán"
};

let declaracionDerechosHumanos = {
  suceso: "La declaración universal de los Derechos Humanos",
  anio: 1948,
  ciudad: "París"
};
```

La función deberá devolver un resumen de la información registrada de manera simple. Por ejemplo:

```
> resumenHechoHistorico(independenciaArgentina)
"La Declaración de la independencia de Argentina ha sucedido hace 205 años en San Miguel de Tucumán"

> resumenHechoHistorico(declaracionDerechosHumanos)
"La Declaración Universal de los Derechos Humanos ha sucedido hace 73 años en París"
```

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

Solución | Consola

```
function resumenHechoHistorico(datos){
  return datos.suceso + " ha sucedido hace " + (2021 - datos.anio) + " años en " + datos.ciudad
}
```

Enviar

Ejercicio 7: Ejercicio 7

¡Dejemos atrás a JavaScript para pasar a Ruby!

Vamos a modelar la clase `Celular` para poder:

- cargar una cantidad de minutos determinada (si cargamos 10 minutos, su batería incrementa en 10);
- ver si tiene carga suficiente, es decir, si su batería es mayor a 50.

> Definí en Ruby, la clase `Celular` que tenga un atributo `@bateria` con su getter. Las instancias de la clase `Celular` entienden los mensajes `cargar!` (que recibe los minutos por parámetro y lo carga esa cantidad) y `suficiente_bateria?`. No te olvides de definir un `initialize` que reciba a la batería inicial como parámetro.

Solución | Consola

```
class Celular
  def initialize(bateria)
    @bateria = bateria
  end

  def bateria
    @bateria
  end

  def cargar!(minutos)
    @bateria += minutos
  end

  def suficiente_bateria?
    @bateria > 50
  end
end
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 8: Ejercicio 8

A `Lu` le gusta mucho leer , pero le interesa saber cuántos de los libros que leyó son largos. Cada uno de los libros sabe responder al mensaje `demasiado_largo?`.

> Definí en Ruby el método `cantidad_de_libros_largos` que responda a cuántos libros largos leyó `Lu`.

Solución | Consola

```ruby
module Lu
  @libros_leidos = [MartinFierro, Fundacion, ElPrincipito, HarryPotter]

  def self.cantidad_de_libros_largos
    @libros_leidos.count {|libro| libro.demasiado_largo? }
  end
end
```

▶ Enviar

✅ ¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 9: Ejercicio 9

Una productora de cine se encarga de invertir en las películas que trabajan con ella. Cuando esta empresa invierte:

- Las películas de tipo `Documental` duplican su `duracion`;
- las de `Terror` contratan 5 `extras`;
- las de género `Comedia` no se modifican.

> Definí el método `invertir_en_peliculas!` en la clase `Productora` y el método `recibir_inversion!` en los distintos tipos de películas. Definí los getters necesarios en cada una.

Solución | Consola

```ruby
class Productora
  def initialize(peliculas)
    @peliculas = peliculas
  end

  def invertir_en_peliculas!
    @peliculas.each{|pelicula|pelicula.recibir_inversion!}
  end

end

class Documental
  def initialize(duracion)
    @duracion = duracion
  end

  def duracion
    @duracion
  end

  def recibir_inversion!
    @duracion = duracion * 2
  end
end
```

```
class Terror
  def initialize(extras)
    @extras = extras
  end

  def extras
    @extras
  end

  def recibir_inversion!
    @extras += 5
  end
end

class Comedia

  def recibir_inversion!
  end

end
```

Enviar

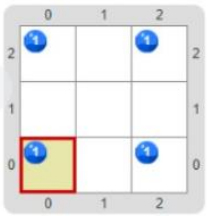
¡Muy bien! Tu solución pasó todas las pruebas

UNO DE LOS TEMAS:

Te quedan 00:34:38

Ejercicio 1: Ejercicio 1

Ale se aburrió de ver vacío su balcón. Así que compró 4 macetas con plantas para ponerlas en cada esquina, de esta forma:



Creá un programa que ponga una maceta (bolita `Azul`) en cada esquina del balcón. El cabezal empieza en el origen (o sea, en el borde Sur-Oeste) pero no te preocupes por dónde finaliza.

```
program{
  Poner(Azul)
  repeat (2){
    Mover(Norte)
  }

  Poner(Azul)
  repeat (2){
    Mover(Este)
  }

  Poner(Azul)
  repeat (2){
    Mover(Sur)
  }

  Poner(Azul)
}
```

Ejercicio 2: Ejercicio 2

A Ale no le gustó como quedó ese color en las macetas . ¡Programemos algo para poder probar como quedaría con cualquier color!

Definí el procedimiento `ColocarMacetas` para que ponga macetas del `color` que reciba por parámetro en cada esquina. No te preocupes por donde termina el cabezal.

```
procedure ColocarMacetas(color){
  Poner(color)
  repeat (2){
    Mover(Norte)
  }

  Poner(color)
  repeat (2){
    Mover(Este)
  }

  Poner(color)
  repeat (2){
    Mover(Sur)
  }

  Poner(color)
}
```

Enviar

Ejercicio 3: Ejercicio 3

Dejemos atrás los tableros y... ¡Pasemos a JavaScript!

Cuando hacemos un examen de matemáticas es común verificar más de una vez si las cuentas que hicimos están bien.

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

```
> laMultiplicacionDaBien(2, 4, 8)
true //Porque 2 por 4 es igual a 8

> laMultiplicacionDaBien(3, 5, 12)
false //Porque 3 por 5 es 15, no 12
```

Definí la función `laMultiplicacionDaBien`.

Solución | Consola

```
function laMultiplicacionDaBien(n1, n2, n3){
  return((n1*n2) === n3)
}
```

Enviar

Ejercicio 4: Ejercicio 4

Ya pasamos por el tesoro de la matemática. Otro bien preciado es el tiempo . Es por ello que tratamos de usarlo sabiamente.

Con esto en mente vamos a crear una función que dados un nombre y un apodo nos diga cuál de los dos es más corto.

```
> elMasCorto("Luis", "Lucho")
"Luis"

> elMasCorto("Carolina", "Caro")
"Caro"

> elMasCorto("Ricardo", "Ringo")
"Ringo"
```

Definí la función `elMasCorto`.

Solución | Consola

```
function elMasCorto(nombre, apodo){
  if (longitud(nombre) > longitud(apodo))
    return apodo;
  else
    return nombre;
}
```

Enviar

Ejercicio 5: Ejercicio 5

Vale está haciendo un trabajo de investigación y nos pidió ayuda . Necesita poder distinguir las palabras más cortas de una oración . Una palabra se considera corta si tiene menos de 5 letras. Veamos un ejemplo:

```
> obtenerPalabrasCortas(["Hari", "Seldon", "nacido", "en", "el", "año", "11988", "de", "la", "Era", "Galáctica"])

["Hari", "en", "el", "año", "de", "la", "Era"]
```

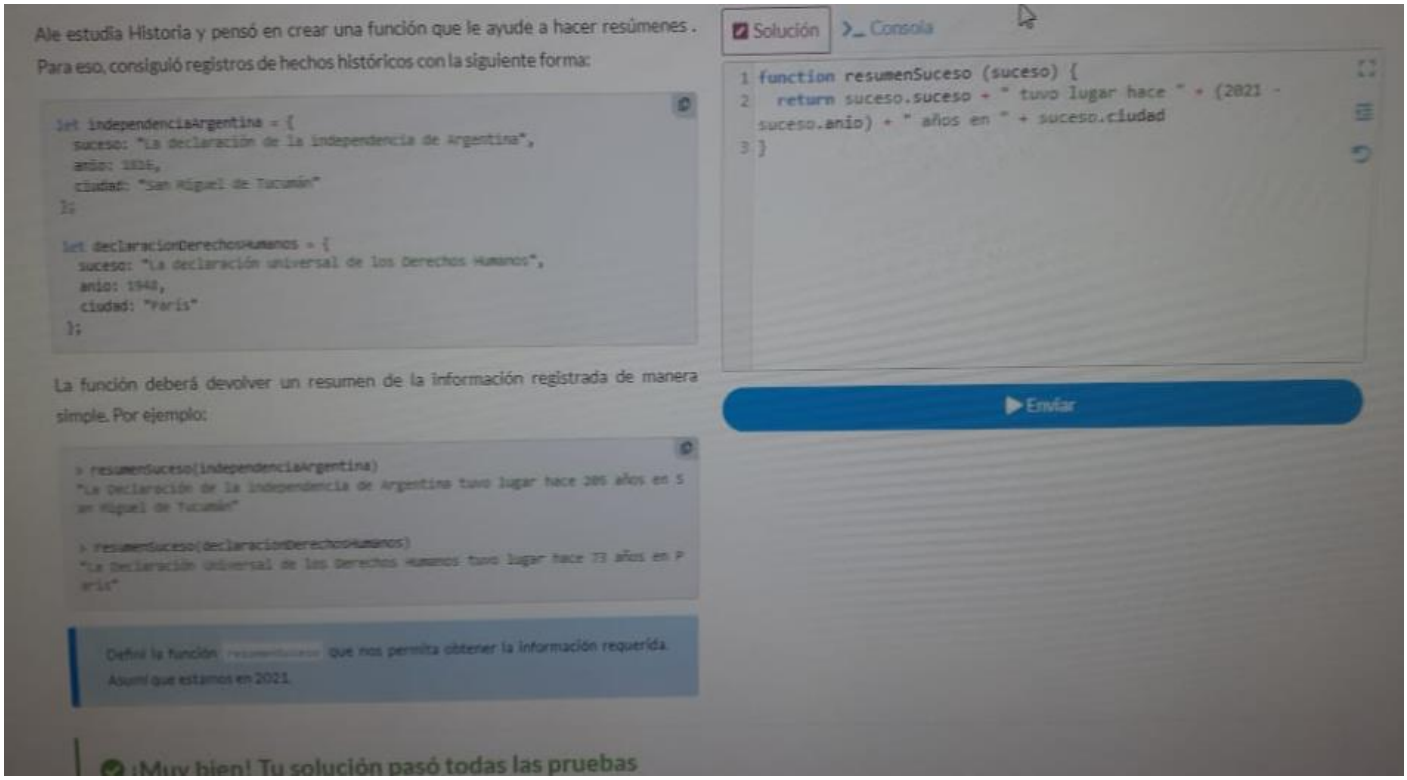
Definí la función `obtenerPalabrasCortas`.

Solución | Consola

```
function obtenerPalabrasCortas(palabras){
  let cortas = [];
  for (let i of palabras){
    if (longitud(i) < 5){
      cortas.push(i)
    }
  }
  return cortas;
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas



Ejercicio 7: Ejercicio 7

¡Dejemos atrás a JavaScript para pasar a Ruby!

Vamos a modelar la clase `Notebook` para poder:

- cargar una cantidad de minutos determinada (si cargamos 10 minutos, su bateria incrementa en 10);
- ver si tiene carga suficiente, es decir, si su bateria es mayor a 56.

> Definí en Ruby, la clase `Notebook` que tenga un atributo `@bateria` con su getter. Las instancias de la clase `Notebook` entienden los mensajes `aumentar_carga!` (que recibe los minutos por parámetro y lo carga esa cantidad) y `carga_suficiente?`. No te olvides de definir un `initialize` que reciba a la bateria inicial como parámetro.

Solución | Consola

```ruby
class Notebook
  @bateria

  def initialize(cant)
    @bateria = cant
  end

  def bateria
    @bateria
  end

  def aumentar_carga!(min)
    @bateria+= min
  end;

  def carga_suficiente?
    @bateria > 56
  end

end
```

Ejercicio 8: Ejercicio 8

A `Pilse` le gusta mucho leer , pero le interesa saber cuántos de los libros que leyó son largos. Cada uno de los libros sabe responder al mensaje `largo?`.

> Definí en Ruby el método `cantidad_largos` que responda a cuántos libros largos leyó `Pilse`.

Solución | Consola

```ruby
module Pilse
  @libros_leidos = [MartinFierro, Fundacion, ElPrincipito, HarryPotter]

  def self.cantidad_largos
    @libros_leidos.count{|libro| libro.largo?}
  end
end
```

Enviar

Cuando esta empresa invierte:

- Las películas de tipo `Documental` duplican su `duracion`;
- las de `Suspenso` contratan 8 `extras`;
- las de género `Comedia` no se modifican.

> Definí el método `invertir!` en la clase `Productora` y el método `recibir_inversion!` en los distintos tipos de películas. Definí los getters necesarios en cada una.

```ruby
class Productora
 def initialize(peliculas)
    @peliculas = peliculas
  end
  def invertir!
    @peliculas.each{|pelicula| pelicula.recibir_inversion!}
  end
end


class Documental
  def initialize(duracion)
    @duracion = duracion
  end
  def duracion
    @duracion
  end
  def recibir_inversion!
    @duracion = duracion * 2
  end
end


class Suspenso
  def initialize(extras)
    @extras = extras
  end
  def extras
    @extras
  end
  def recibir_inversion!
    @extras += 8
  end
end

class Comedia
  def recibir_inversion!
  end
```

Una productora de cine se encarga de invertir en las películas que trabajan con ella. Cuando esta empresa invierte:

- Las películas de tipo `Drama` duplican su `duracion`;
- las de `Horror` contratan 11 `extras`;
- las de género `Comedia` no se modifican.

Definí el método `invertir_en_todas!` en la clase `Productora` y el método `recibir_inversion!` en los distintos tipos de películas. Definí los getters necesarios en cada una.

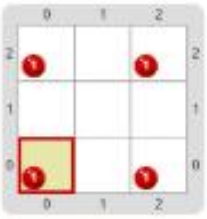
Solución | Consola

```
class Productora
  def initialize(peliculas)
    @peliculas = peliculas
  end

  def invertir_en_todas!
    @peliculas.each{|pelicula|
pelicula.recibir_inversion!}
  end
end

class Drama
  def initialize(duracion)
    @duracion = duracion
  end

  def duracion
    @duracion
  end

  def recibir_inversion!
    @duracion*= 2
  end

end

class Horror
  def initialize(extras)
    @extras = extras
```

UNO DE LOS TEMAS:

Ejercicio 1: Ejercicio 1

Ale se aburrió de ver vacío su balcón. Así que compró 4 macetas con plantas para ponerlas en cada esquina, de esta forma:



Creá un programa que ponga una maceta (bolita `Rojo`) en cada esquina del balcón. 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)Mover(Este)Poner(Rojo)Mover(
Norte)Mover(Norte)Poner(Rojo)Mover(Oeste)Mover(Oest
e)Poner(Rojo)}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Te quedan 01:56:11

Ejercicio 2: Ejercicio 2

A Ale no le gustó como quedó ese color en las macetas . ¡Programemos algo para poder probar como quedaría con cualquier color!

> Definí el procedimiento `PonerMacetas` para que ponga macetas del `color` que reciba por parámetro en cada esquina. No te preocupes por donde termina el cabezal.

```
procedure PonerMacetas(color)
{Poner(color)Mover(Este)Mover(Este)Poner(color)Move
r(Norte)Mover(Norte)Poner(color)Mover(Oeste)Mover(O
este)Poner(color)}
```

Enviar

Ejercicio 3: Ejercicio 3

Dejemos atrás los tableros y... ¡Pasemos a JavaScript!

Cuando hacemos un examen de matemáticas es común verificar más de una vez si las cuentas que hicimos están bien.

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

```
> daBienLaMultiplicacion(2, 4, 8)
true //Porque 2 por 4 es igual a 8

> daBienLaMultiplicacion(3, 5, 12)
false //Porque 3 por 5 es 15, no 12
```

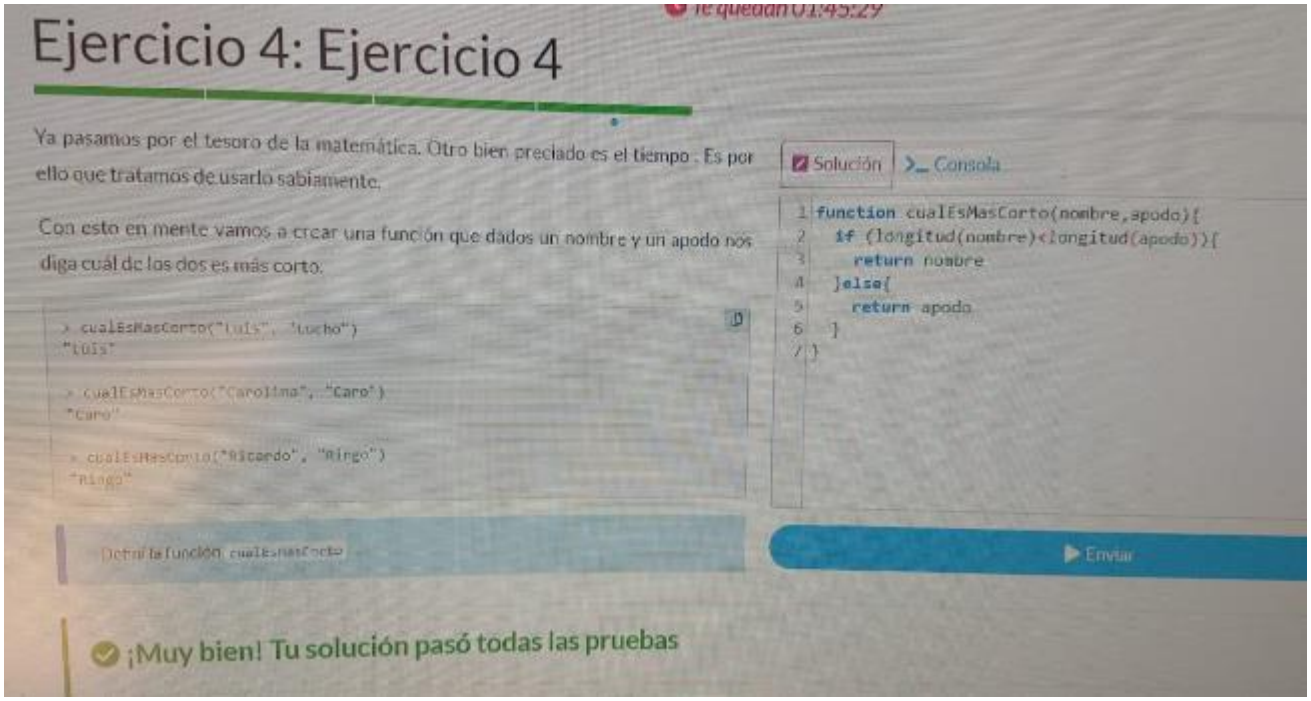
> Definí la función `daBienLaMultiplicacion`

Solución | Consola

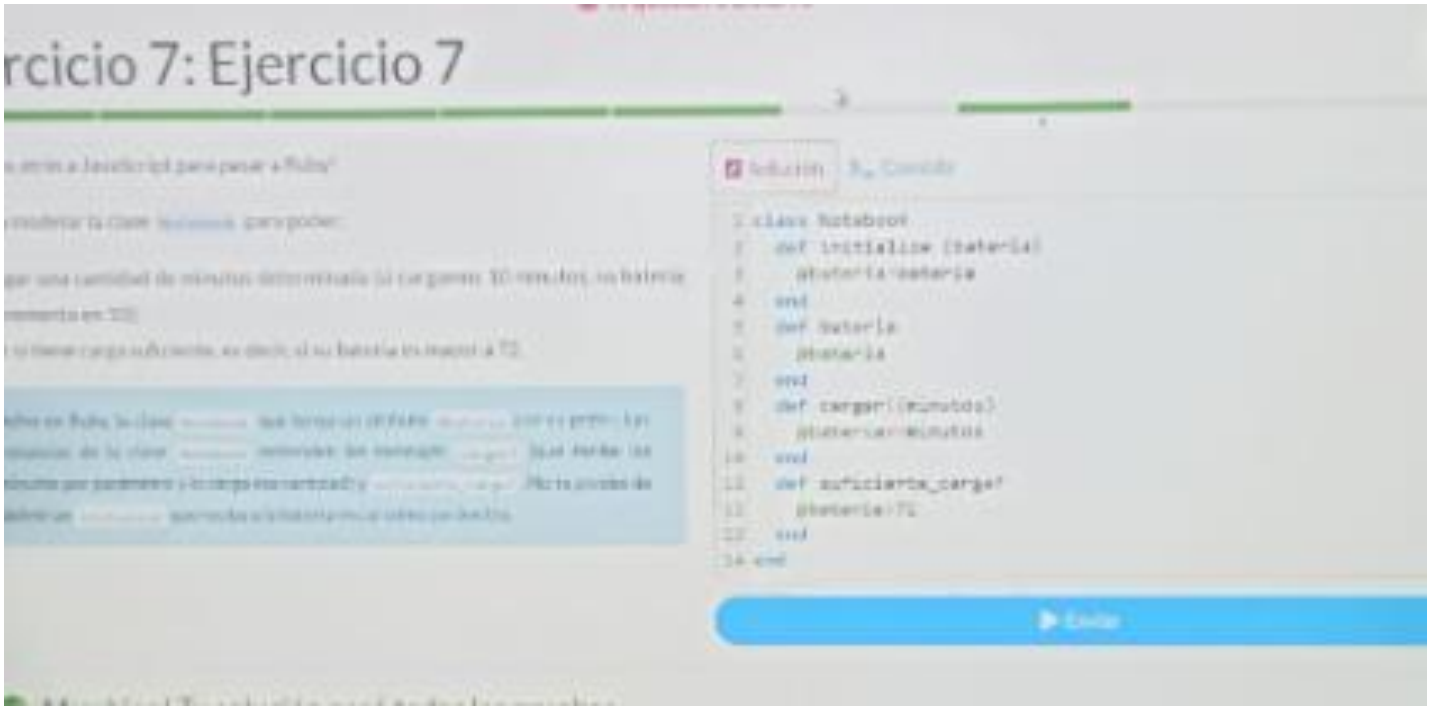
```
function daBienLaMultiplicacion(num1, num2, num3){
  return((num1*num2) === num3)
}
```

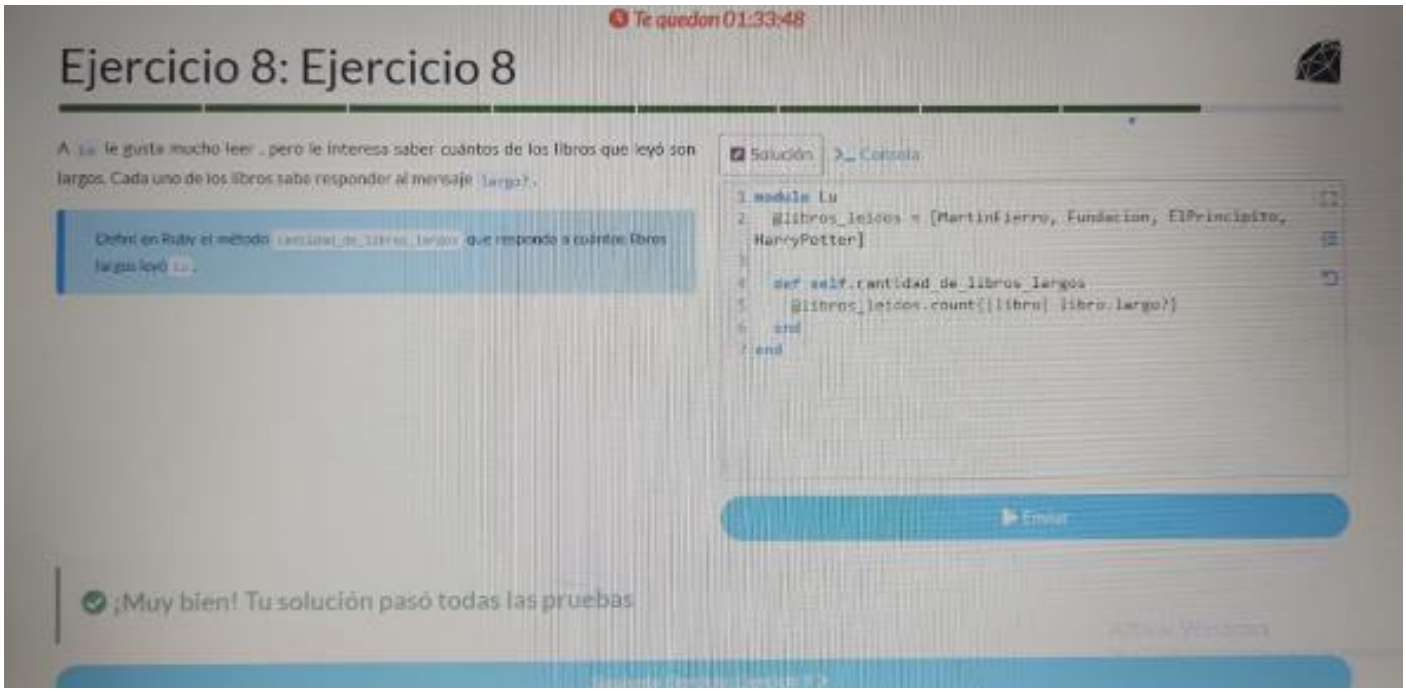
Enviar

¡Muy bien! Tu solución pasó todas las pruebas


Te quedan 01:45:29
Ejercicio 4: Ejercicio 4
Ya pasamos por el tesoro de la matemática. Otro bien preciado es el tiempo . Es por ello que tratamos de usarlo sabiamente.
Con esto en mente vamos a crear una función que dados un nombre y un apodo nos diga cuál de los dos es más corto.
> cualEsMasCorto("Luis", "Lucho")
"Luis"
> cualEsMasCorto("Carolina", "Caro")
"Caro"
> cualEsMasCorto("Ricardo", "Ringo")
"Ringo"
Solución
Consola
function cualEsMasCorto(nombre,apodo){
if (longitud(nombre)<longitud(apodo)){
return nombre
}else{
return apodo
}
}
Enviar
¡Muy bien! Tu solución pasó todas las pruebas


Te quedan 01:37:31
Ejercicio 6: Ejercicio 6
JS
Ale estudia Historia y pensó en crear una función que le ayude a hacer resúmenes . Para eso, consiguió registros de hechos históricos con la siguiente forma:
let independenciaArgentina = {
suceso: "La declaración de la independencia de Argentina",
anio: 1816,
ciudad: "San Miguel de Tucumán"
};
let declaracionDerechosHumanos = {
suceso: "La declaración universal de los Derechos Humanos",
anio: 1948,
ciudad: "París"
};
La función deberá devolver un resumen de la información registrada de manera simple. Por ejemplo:
> resumenHechoMemorable(independenciaArgentina)
"La Declaración de la independencia de Argentina tuvo lugar hace 205 años en San Miguel de Tucumán"
Solución
Consola
function resumenHechoMemorable(suceso){
return suceso.suceso + " tuvo lugar hace " +
(2021 - suceso.anio) + " años en " + suceso.ciudad
}
Enviar


rcicio 7: Ejercicio 7
Solución
>_ Consola
class Notebook
def initialize (bateria)
@bateria=bateria
end
def bateria
@bateria
end
def cargar(minutos)
@bateria+=minutos
end
def suficiente_carga?
@bateria>71
end
end
Enviar


Te quedan 01:33:48
Ejercicio 8: Ejercicio 8
A Lu le gusta mucho leer , pero le interesa saber cuántos de los libros que leyó son largos. Cada uno de los libros sabe responder al mensaje largo?.
Definí en Ruby el método cantidad_de_libros_largos que responde a cuántos libros largos leyó Lu.
Solución
>_ Consola
module Lu
@libros_leidos = [MartinFierro, Fundacion, ElPrincipito, HarryPotter]
def self.cantidad_de_libros_largos
@libros_leidos.count{|libro| libro.largo?}
end
end
Enviar
¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 9: Ejercicio 9

Una productora de cine se encarga de invertir en las películas que trabajan con ella. Cuando esta empresa invierte:

- Las películas de tipo `Documental` duplican su `duracion`;
- las de `Terror` contratan 5 `extras`;
- las de género `Comedia` no se modifican.

> Definí el método `invertir_en_peliculas!` en la clase `Productora` y el método `recibir_inversion!` en los distintos tipos de películas. Definí los getters necesarios en cada una.

Solución >_ Consola

```
class Productora
  def initialize(peliculas)
    @peliculas = peliculas
  end

  def invertir_en_peliculas!
    @peliculas.each{|pelicula|pelicula.recibir_inversion!}
  end

end

class Documental
  def initialize(duracion)
    @duracion = duracion
  end

  def duracion
    @duracion
  end

  def recibir_inversion!
    @duracion = duracion * 2
  end
end
```

14:54

```
class Terror
  def initialize(extras)
    @extras = extras
  end

  def extras
    @extras
  end

  def recibir_inversion!
    @extras += 5
  end
end

class Comedia

  def recibir_inversion!
  end

end
```

▶ Enviar

✔ ¡Muy bien! Tu solución pasó todas las pruebas

14:54