

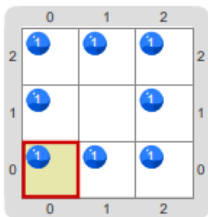
EJERCICIOS DIA 16 A LAS 9 HS DE LA MAÑANA

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.

```
1 program {
2   Poner(Azul)
3   Mover(Norte)
4   Poner(Azul)
5   Mover(Norte)
6   Poner(Azul)
7   Mover(Este)
8   Poner(Azul)
9   Mover(Este)
10  Poner(Azul)
11  Mover(Sur)
12  Poner(Azul)
13  Mover(Sur)
14  Poner(Azul)
15  Mover(Oeste)
16  Poner(Azul)
17
18
19 }
```

▶ Enviar

🔥 le 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.

```
1 procedure ColorearMarco(color){
2   Poner (color)
3   Mover(Norte)
4   Poner (color)
5   Mover(Norte)
6   Poner (color)
7   Mover(Este)
8   Poner (color)
9   Mover(Este)
10  Poner (color)
11  Mover(Sur)
12  Poner (color)
13  Mover(Sur)
14  Poner (color)
15  Mover(Oeste)
16  Poner(color)
17 }
18
```

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:

```
> 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](#)

```
1 function laSumaEsMasChica(a, b, c){
2   return(a+b) < c
3 }
```

▶ Enviar

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 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"
```

Definí la función `seleccionarVestimenta`.

☒ Solución [> Consola](#)

```
1 function seleccionarVestimenta (temp, color){
2
3   if (temp >= 18){
4     return "Remera " + color
5   } else {
6     return "Campera " + color
7   }
8 }
```

▶ Enviar

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", "provincia"])
["polilla", "caracol", "provincia"]
```

Definí la función `filtrarLargas`.

[Solución](#)[Consola](#)

```
1 function filtrarLargas (palabras){
2   let palabrasFiltradas= []
3   for (let palabra of palabras){
4     if(longitud(palabra) > 6){
5       agregar(palabrasFiltradas,palabra)
6     }
7   }
8   return palabrasFiltradas
9 }
```

[▶ 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 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:

[Solución](#)[Consola](#)

```
1 function resumenDeLaSuscripcion(persona){
2   return persona.nombre + " se suscribio hace " +
3     (2021 - persona.anioSuscripcion) + " años y leyó " +
4     longitud(persona.librosLeidos) + " ejemplares"
5 }
```

[▶ Enviar](#)



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

```
1 class Comida
2   def initialize(cant)
3     @cucharadas_sal = cant
4   end
5   def cucharadas_sal
6     @cucharadas_sal
7   end
8   def agregar_cucharadas!(cantidad)
9     @cucharadas_sal += cantidad
10  end
11
12  def exceso_de_sal?
13    return @cucharadas_sal > 5
14  end
15 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

```
1 module Compilado
2   @canciones = [AmorAusente, Eco, Agujas, ElBalcon,
3                 GuitarrasDeCarton]
4
5   def self.cuantas_canciones_cortas
6     @canciones.count {|cancion|
7       cancion.demasiado_corta?}
8   end
9 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](#)

```
1 class Persona
2   def initialize(mascota)
3     @mascota = mascota
4   end
5
6   def jugar_con_mascota!
7     @mascota.jugar!
8   end
9 end
10
11 class Perro
12   def initialize()
13     @tiene_hambre = false
14   end
15
16   def tiene_hambre
17     @tiene_hambre
18   end
19
20   def jugar!
21     @tiene_hambre = true
22   end
23 end
24
```

```
15
16 def tiene_hambre
17   @tiene_hambre
18 end
19
20 def jugar!
21   @tiene_hambre = true
22 end
23 end
24
25 class Conejo
26   def initialize(nivel_de_felicidad)
27     @nivel_de_felicidad = nivel_de_felicidad
28   end
29   def jugar!
30     @nivel_de_felicidad += 4
31   end
32   def nivel_de_felicidad
33     @nivel_de_felicidad
34   end
35 end
36
37 class Tortuga
38   def jugar!
39
40   end
41 end
42
```

▶ 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.

```
1 procedure CuadroVerde(){
2   Poner(Verde)
3   Mover(Norte)
4   Poner(Verde)
5   Mover(Norte)
6   Poner(Verde)
7   Mover(Este)
8   Poner(Verde)
9   Mover(Este)
10  Poner(Verde)
11  Mover(Sur)
12  Poner(Verde)
13  Mover(Sur)
14  Poner(Verde)
15  Mover(Oeste)
16  Poner(Verde)
17 }
18
19 program {
20
21   CuadroVerde()
22 }
```

▶ Enviar

Activar Windows

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.

```
1 procedure ColorearMarco(color){
2   Poner(color)
3   Mover(Norte)
4   Poner(color)
5   Mover(Norte)
6   Poner(color)
7   Mover(Este)
8   Poner(color)
9   Mover(Este)
10  Poner(color)
11  Mover(Sur)
12  Poner(color)
13  Mover(Sur)
14  Poner(color)
15  Mover(Oeste)
16  Poner(color)
17 }
18
```

▶ Enviar

⌚ Te quedan 00:50:31

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:

```
> 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](#)

```
1 function esMasChicaLaSuma(num1,num2,num3){
2   let suma = num1 + num2
3
4   return suma < num3
5 }
```

▶ Enviar

✅ ¡Muy bien! Tu solución pasó todas las pruebas

Activar Windows
Ve a Configuración para activar Windows

⌚ Te quedan 00:49:52

Ejercicio 4: Ejercicio 4

JS

Si hay algo que a Ale le molesta es o pasar frío o abrigarse de más. Pero lo que 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](#)

```
1 function seleccionarVestimenta(temp,color){
2
3   if(temp >= 24){
4     return "Remera " + color
5   }
6
7   else {
8     return "Campera " + color
9   }
10
11 }
```

▶ Enviar

✅ ¡Muy bien! Tu solución pasó todas las pruebas

Activar Windows
Ve a Configuración para activar Windows

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.

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

Definí la función `filtrarLargas`.

[Solución](#) [> Consola](#)

```
1 function filtrarLargas (palabras){
2
3   let palabrasFiltradas = []
4   for (let palabra of palabras){
5     if(longitud(palabra) > 6){
6       agregar(palabrasFiltradas,palabra)
7     }
8   }
9   return palabrasFiltradas
10 }
```

[▶ Enviar](#)

✓ ¡Muy bien! Tu solución pasó todas las pruebas

[Activar Windows](#)

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 Arrevert",
  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:

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

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

[Solución](#) [> Consola](#)

```
1 function resumenDeLaSuscripcion(persona){
2   return persona.nombre + " se registró hace " +
3     (2021 - persona.anioSuscripcion) + " años y leyó " +
4     longitud(persona.librosLeidos) + " ejemplares"
5 }
```

[▶ Enviar](#)[Activar Windows](#)

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](#)

```
1 class Almuerzo
2   def initialize(cant)
3     @cucharadas_sal = cant
4   end
5
6   def cucharadas_sal
7     @cucharadas_sal
8   end
9
10  def poner_cucharadas!(cantidad)
11    @cucharadas_sal += cantidad
12  end
13
14  def exceso_de_sal?
15    return @cucharadas_sal > 5
16  end
17
18 end
```

▶ Enviar

Activar Windows

🕒 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](#)

```
1 module Disco
2   @canciones = [AmorAusente, Eco, Agujas, ElBalcon,
3                 GuitarrasDeCarton]
4
5   def self.cantidad_cortas
6     @canciones.count {|cancion|
7       cancion.demasiado_corta?}
8   end
9 end
```

▶ Enviar

Activar Windows

✅ ¡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

```
1 class Persona
2   def initialize(mascota)
3     @mascota = mascota
4   end
5
6   def jugar_con_mascota!
7     @mascota.jugar!
8   end
9 end
10
11 class Hamster
12   def initialize()
13     @tiene_hambre = false
14   end
15
16   def tiene_hambre
17     @tiene_hambre
18   end
19
20   def jugar!
21     @tiene_hambre = true
22   end
23 end
24
25 class Conejo
26   def initialize(nivel_de_felicidad)
27     @nivel_de_felicidad = nivel_de_felicidad
28   end
```

Activar Windows
Ve a Configuración para activar Windows

```
24
25 class Conejo
26   def initialize(nivel_de_felicidad)
27     @nivel_de_felicidad = nivel_de_felicidad
28   end
29   def jugar!
30     @nivel_de_felicidad += 8
31   end
32   def nivel_de_felicidad
33     @nivel_de_felicidad
34   end
35 end
36
37 class Tortuga
38   def jugar!
39
40   end
41 end
```

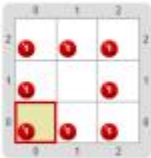
▶ Enviar

✅ ¡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 (0 sea, en el borde Sur-Oeste) pero no te preocupes por dónde finaliza.

```
1 program{
2   Poner(Rojo)
3   Mover(Este)
4   Poner(Rojo)
5   Mover(Este)
6   Poner(Rojo)
7   Mover(Norte)
8   Poner(Rojo)
9   Mover(Norte)
10  Poner(Rojo)
11  Mover(Oeste)
12  Poner(Rojo)
13  Mover(Oeste)
14  Poner(Rojo)
15  Mover(Sur)
16  Poner(Rojo)
17 }
```

▶ 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.

```
1 procedure DarColorAlMarco {color}{
2   Poner(color)
3   Mover(Este)
4   Poner(color)
5   Mover(Este)
6   Poner(color)
7   Mover(Norte)
8   Poner(color)
9   Mover(Norte)
10  Poner(color)
11  Mover(Oeste)
12  Poner(color)
13  Mover(Oeste)
14  Poner(color)
15  Mover(Sur)
16  Poner(color)
17 }
```

▶ 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](#)

```
1 function esMenorLaSuma(n1,n2,n3){
2   return (n1 + n2) < n3
3 }
```

▶ 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(10, "verde")
"Campera verde"
> vestirseAcorde(25, "violeta")
"Remera violeta"
```

Definí la función `vestirseAcorde`.

☒ Solución [> Consola](#)

```
1 function vestirseAcorde (temp,color){
2   if(temp >= 20){
3     return "Remera "+color
4   }else{
5     return "Campera "+color
6   }
7 }
```

▶ 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", "polillo", "caracol", "gato", "provincia"])
["polillo", "caracol", "provincia"]
```

Definí la función `palabrasMayores`.

☒ Solución [> Consola](#)

```
1 function palabrasMayores(palabras){
2   let palabrasFiltradas= []
3   for (let palabra of palabras){
4     if(longitud(palabra) > 6){
5       agregar(palabrasFiltradas,palabra);
6     }
7   }
8   return palabrasFiltradas
9 }
10
11
```

▶ 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 Arreaver",
  librosLeídos: ["El conde de Montecristo", "La palabra", "Mi planta de naranja lima"],
  añoSuscripción: 1992
};

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

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

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

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

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

[Solución](#) [Consola](#)

```
1 function resumenDeSuscripción(persona){
2   return persona.nombre + " se registró hace " + (2021 -
  persona.añoSuscripción) + " años y leyó
  "+ longitud(persona.librosLeídos) + " libros"
3 }
```

▶ 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` 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](#)

```
1 class Comida
2   def initialize(cant)
3     @cucharadas_sal = cant
4   end
5   def cucharadas_sal
6     @cucharadas_sal
7   end
8   def poner_cucharadas!(cantidad)
9     @cucharadas_sal += cantidad
10  end
11
12  def mucha_sal?
13    return @cucharadas_sal > 4
14  end
15 end
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

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](#)

```
1 module Disco
2   @canciones = [AmorAusente, Eco, Agujas, ElBalcon,
3   def self.cantidad_de_cortas
4     @canciones.count {|cancion| cancion.demasiado_corta?}
5   end
6 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 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](#)

```
1 class Persona
2   def initialize(mascota)
3     @mascota = mascota
4   end
5   def jugar_con_mascota!
6     @mascota.jugar!
7   end
8 end
9
10 class Hamster
11   def initialize()
12     @tiene_hambre = false
13   end
14   def tiene_hambre
15     @tiene_hambre
16   end
17   def jugar!
18     @tiene_hambre = true
19   end
20 end
21
22 class Conejo
23   def initialize(nivel_de_felicidad)
24     @nivel_de_felicidad = nivel_de_felicidad
25   end
26   def jugar!
27     @nivel_de_felicidad += 3
28   end
29
30   def nivel_de_felicidad
31     @nivel_de_felicidad
32   end
33
34 class Tortuga
35   def jugar!
36   end
37 end
38
```

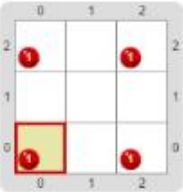
▶ Enviar

Ejercicios 16 A LAS 2 DE LA TARDE

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 roja) en cada esquina del balcón. El cabeza! empieza en el origen (o sea, en el borde Sur-Oeste) pero no te preocupes por dónde finaliza.

```
1 program {
2   Poner(Rojo)
3   Mover(Este)
4   Mover(Este)
5   Poner(Rojo)
6
7   Mover(Norte)
8   Mover(Norte)
9
10  Poner(Rojo)
11  Mover(Oeste)
12  Mover(Oeste)
13  Poner(Rojo)
14 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial Tablero final



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.

```
1 procedure PonerMacetas (color){
2   Poner(color)
3   Mover(Este)
4   Mover(Este)
5   Poner(color)
6   Mover(Norte)
7   Mover(Norte)
8   Poner(color)
9   Mover(Oeste)
10  Mover(Oeste)
11  Poner(color)
12 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:



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

```
1 function esCorrectaLaMultiplicacion(num1, num2, num3){
2   return (num1*num2) === num3
3 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 4: Ejercicio 4

JS

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](#)

```
1 function cualEsMasCorto(nombre, apodo){
2   if(longitud(nombre)>longitud(apodo)){
3     return apodo
4   }
5   else {
6     return nombre
7   }
8 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Ejercicio 5: Ejercicio 5

JS

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

Definí la función `palabrasCortas`.☒ Solución [> Consola](#)

```
1 function palabrasCortas(palabras){
2   let listaPalabras= []
3   for (let palabra of palabras){
4     if (longitud(palabra) < 5){
5       agregar(listaPalabras,palabra);
6     }
7   }
8   return listaPalabras
9 }
```

▶ 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",
  año: 1816,
  ciudad: "San Miguel de Tucumán"
};

let declaracionDerechosHumanos = {
  suceso: "La declaración universal de los Derechos Humanos",
  año: 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](#)

```
1 function resumenHechoHistorico(datos){
2   return datos.suceso + " ha sucedido hace " + (2021 -
3     datos.año) + " años en " + datos.ciudad
4 }
```

▶ 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](#)

```
1 class Celular
2   def initialize(bateria)
3     @bateria = bateria
4   end
5
6   def bateria
7     @bateria
8   end
9
10  def cargar!(minutos)
11    @bateria += minutos
12  end
13
14  def suficiente_bateria?
15    @bateria > 50
16  end
17 end
18
```

▶ 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](#)

```
1 module Lu
2   @libros_leidos = [MartinFierro, Fundacion, ElPrincipito,
3                     HarryPotter]
4
5   def self.cantidad_de_libros_largos
6     @libros_leidos.count { |libro| libro.demasiado_largo? }
7   end
8 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](#)

```
1 class Productora
2   def initialize(peliculas)
3     @peliculas = peliculas
4   end
5
6   def invertir_en_peliculas!
7     @peliculas.each { |pelicula| pelicula.recibir_inversion! }
8   end
9
10 end
11
12 class Documental
13   def initialize(duracion)
14     @duracion = duracion
15   end
16
17   def duracion
18     @duracion
19   end
20
21   def recibir_inversion!
22     @duracion = duracion * 2
23   end
24 end
```

```

26 class Terror
27   def initialize(extras)
28     @extras = extras
29   end
30
31   def extras
32     @extras
33   end
34
35   def recibir_inversion!
36     @extras += 5
37   end
38 end
39
40 class Comedia
41
42   def recibir_inversion!
43     end
44
45 end

```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

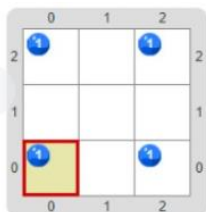
UNO DE LOS TEMAS:

🕒 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.

```

1 program{
2   Poner(Azul)
3   repeat (2){
4     Mover(Norte)
5   }
6
7   Poner(Azul)
8   repeat (2){
9     Mover(Este)
10  }
11
12  Poner(Azul)
13  repeat (2){
14    Mover(Sur)
15  }
16
17  Poner(Azul)
18 }

```

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.

```
1 procedure ColocarMacetas(color){
2   Poner(color)
3   repeat (2){
4     Mover(Norte)
5   }
6
7   Poner(color)
8   repeat (2){
9     Mover(Este)
10  }
11
12  Poner(color)
13  repeat (2){
14    Mover(Sur)
15  }
16
17  Poner(color)
18 }
```

▶ Enviar

Ejercicio 3: Ejercicio 3

JS

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

```
1 function laMultiplicacionDaBien(n1, n2, n3){
2   return((n1*n2) === n3)
3 }
```

▶ Enviar

Ejercicio 4: Ejercicio 4

JS

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](#)

```
1 function elMasCorto(nombre, apodo){
2   if (longitud(nombre) > longitud(apodo))
3     return apodo;
4   else
5     return nombre;
6 }
```

▶ Enviar

Ejercicio 5: Ejercicio 5

JS

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](#)

```
1 function obtenerPalabrasCortas(palabras){
2   let cortas = [];
3   for (let i of palabras){
4     if (longitud(i) < 5){
5       cortas.push(i)
6     }
7   }
8   return cortas;
9 }
```

▶ 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",
  año: 1816,
  ciudad: "San Miguel de Tucumán"
};

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

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

```
> resumenSuceso(independenciaArgentina)
"La Declaración de la independencia de Argentina tuvo lugar hace 205 años en S
an Miguel de Tucumán"

> resumenSuceso(declaracionDerechosHumanos)
"La Declaración Universal de los Derechos Humanos tuvo lugar hace 73 años en P
arís"
```

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

Solución **Consola**

```
1 function resumenSuceso (suceso) {
2   return suceso.suceso + " tuvo lugar hace " + (2021 -
   suceso.año) + " años en " + suceso.ciudad
3 }
```

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 batería incrementa en 10);
- ver si tiene carga suficiente, es decir, si su batería 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 batería inicial como parámetro.

Solución **Consola**

```
1 class Notebook
2   @bateria
3
4   def initialize(cant)
5     @bateria = cant
6   end
7
8   def bateria
9     @bateria
10  end
11
12  def aumentar_carga!(min)
13    @bateria += min
14  end;
15
16  def carga_suficiente?
17    @bateria > 56
18  end
19
20 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](#)

```
1 module Pilse
2   @libros_leidos = [MartinFierro, Fundacion,
3                     ElPrincipito, HarryPotter]
4
5   def self.cantidad_largos
6     @libros_leidos.count{|libro| libro.largo?}
7   end
8 end
```

[▶ Enviar](#)

Cuando esta empresa invierte:

- Las películas de tipo `documental` duplican su `duracion`;
- las de `Suspense` 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.

```
1 class Productora
2   def initialize(peliculas)
3     @peliculas = peliculas
4   end
5   def invertir!
6     @peliculas.each{|pelicula|
7       pelicula.recibir_inversion!
8     }
9   end
10 end
11
12 class Documental
13   def initialize(duracion)
14     @duracion = duracion
15   end
16   def duracion
17     @duracion
18   end
19   def recibir_inversion!
20     @duracion = duracion * 2
21   end
22 end
23
24 class Suspense
25   def initialize(extras)
26     @extras = extras
27   end
28   def extras
29     @extras
30   end
31   def recibir_inversion!
32     @extras += 8
33   end
34 end
35
36 class Comedia
37   def recibir_inversion!
38   end
39 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](#)

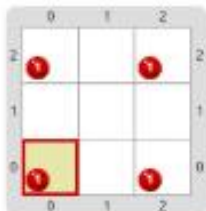
```
1 class Productora
2   def initialize(peliculas)
3     @peliculas = peliculas
4   end
5
6   def invertir_en_todas!
7     @peliculas.each{|pelicula|
8       pelicula.recibir_inversion!
9     }
10  end
11
12 class Drama
13   def initialize(duracion)
14     @duracion = duracion
15   end
16
17   def duracion
18     @duracion
19   end
20
21   def recibir_inversion!
22     @duracion *= 2
23   end
24 end
25
26 class Horror
27   def initialize(extras)
28     @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.

```
1 program
2 {Poner(Rojo)Mover(Este)Mover(Este)Poner(Rojo)Mover(
3 Norte)Mover(Norte)Poner(Rojo)Mover(Oeste)Mover(Oeste)
4 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.

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

▶ Enviar

Ejercicio 3: Ejercicio 3

JS

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:

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

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

Definí la función `deBienLaMultiplicacion`

✓ Solución > Consola

```
1 function deBienLaMultiplicacion(num1, num2, num3){
2   return((num1*num2) === num3)
3 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

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"
```

Detalla función: `cualEsMasCorto`

☒ Solución [> Consola](#)

```
1 function cualEsMasCorto(nombre, apodo){
2   if (longitud(nombre) < longitud(apodo)){
3     return nombre
4   } else {
5     return apodo
6   }
7 }
```

[▶ 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",
  año: 1816,
  ciudad: "San Miguel de Tucumán"
};

let declaracionDerechosHumanos = {
  suceso: "La declaración universal de los Derechos Humanos",
  año: 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](#)

```
1 function resumenHechoMemorable(suceso){
2
3   return suceso.suceso + " tuvo lugar hace " +
4     (2021 - suceso.año) + " años en " + suceso.ciudad
5
6 }
```

[▶ Enviar](#)

Ejercicio 7: Ejercicio 7

¿Quieres leer un libro pero no tienes tiempo?

Considera la clase `Notabook`, que puedes:

• cargar una cantidad de minutos determinada (si cargamos 10 minutos, la batería aumenta en 10).

• saber si tiene carga suficiente, es decir, si su batería es mayor a 70.

Definí en Ruby la clase `Notabook` que tenga un método `carregar` que reciba como parámetro la cantidad de minutos que se quiere cargar y otro método `suficiente_carga?` que devuelva `true` si la batería es mayor a 70, y `false` en caso contrario.

☒ Solución [Ver Solución](#)

```
1 class Notabook
2   def initialize(bateria)
3     @bateria = bateria
4   end
5   def bateria
6     @bateria
7   end
8   def cargar(minutos)
9     @bateria += minutos
10  end
11  def suficiente_carga?
12    @bateria > 70
13  end
14 end
```

[Enviar](#)

Te quedan 01:33:48

Ejercicio 8: Ejercicio 8

Ale 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 `largos?`.

Definí en Ruby el método `cantidad_de_libros_largos` que responde a cuántos libros largos leyó.

☒ Solución [Ver Solución](#)

```
1 module Lu
2   @libros_leidos = [MartinFierro, Fundacion, ElPrincipito,
3                     HarryPotter]
4   def self.cantidad_de_libros_largos
5     @libros_leidos.count{|libro| libro.largos?}
6   end
7 end
```

[Enviar](#)

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Active Windows

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.

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

Solución

[Ver Solución](#)

```
1 class Productora
2   def initialize(peliculas)
3     @peliculas = peliculas
4   end
5
6   def invertir_en_peliculas!
7     @peliculas.each{|pelicula|pelicula.recibir_inversion!}
8   end
9
10  end
11
12  class Documental
13    def initialize(duracion)
14      @duracion = duracion
15    end
16
17    def duracion
18      @duracion
19    end
20
21    def recibir_inversion!
22      @duracion = duracion * 2
23    end
24  end
```

14:54

```
26 class Terror
27   def initialize(extras)
28     @extras = extras
29   end
30
31   def extras
32     @extras
33   end
34
35   def recibir_inversion!
36     @extras += 5
37   end
38 end
39
40 class Comedia
41   def recibir_inversion!
42   end
43 end
44
45 end
```

[Enviar](#)

✅ ¡Muy bien! Tu solución pasó todas las pruebas

14:54

