




Examen 10-04-2022 Tema

Programación I (Universidad Tecnológica Nacional)

Te quedan 01:59:56

## Ejercicio 1: Ejercicio 1



Una fábrica de chocolates nos pidió un programa que se encargue de armar una caja de bombones con distintos sabores . Actualmente venden bombones de frutilla, menta y chocolate amargo que representaremos con bolitas de color Rojo, Verde y Negro respectivamente. Las cajas tienen cuatro bombones y esta en particular tendrá los siguientes gustos:



Es decir, una bolita de color Rojo, al Este una de color Verde, al Este una de color Negro y al Este una de color Negro.

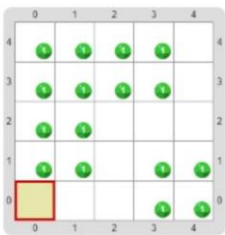
Creá el programa que haga la caja de bombones solicitada. El cabezal comienza en el extremo Sur Oeste y no importa dónde termina.

1 ...escribí tu solución acá...

▶ Enviar

## Ejercicio 1: Ejercicio 1

Como primer desafío de este examen vamos a salir de este laberinto :




La entrada empieza donde comienza el cabezal, es decir, en la esquina inferior izquierda y la salida en la esquina superior derecha.

Creá un programa que dibuje el camino a la salida con una bolita en cada celda de color Rojo.

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

## Ejercicio 1: Ejercicio 1

Como primer desafío de este examen vamos a ayudar a Hansel y Gretel para que puedan regresar a casa . Para ello tienen que desandar el camino que marcaron con migas que representaremos con bolitas de color Negro:




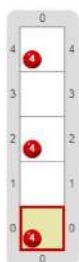
El camino empieza donde comienza el cabezal, es decir, en la esquina inferior izquierda y termina en la esquina superior derecha.

Creá un programa que quite las bolitas de color Negro del camino que lleva a Hansel y Gretel a su casa.

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

## Ejercicio 1: Ejercicio 1

Como primer desafío de este examen vamos a sembrar unos tomates . Las semillas las representaremos con bolitas de color Rojo y, para que den fruto, por cada celda tenemos que poner 4 semillas dejando un espacio entre cada celda. El tablero debería quedar así:



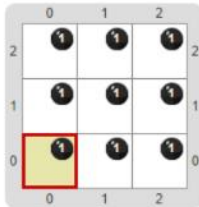
```
1 program {
2   repeat(2){
3     repeat(4){
4       Poner(Rojo)
5     }
6   }
7   repeat(2){
8     Mover(Norte)}}
9   repeat(4){
10    Poner(Rojo)
11  }
12 }
```

Te quedan 01:52:36

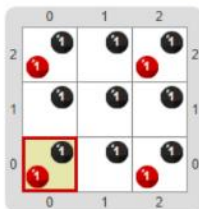
## Ejercicio 2: Ejercicio 2



Una extravagante repostería 🍰 nos pidió ayuda para decorar su famosa torta cuadrada de chocolate:



La decoración consta de un confite de un mismo color en cada extremo de la torta. El color puede ser Azul, Rojo, Verde o Negro, ¡eso depende del gusto de quien encargue la torta! Si por ejemplo, alguien pide una torta con confites de color Rojo, la torta decorada debería verse así:



Definí el procedimiento `Decorar` que recibe un `color` como argumento y decora la torta con confites de ese color comenzando en el extremo Sur Oeste. No importa dónde termina el cabezal.

1 ...escribí tu solución acá...

▶ Enviar

## Ejercicio 2: Ejercicio 2

Una librería que vende combos con útiles escolares 📚 nos pidió un procedimiento que se encargue de armar una caja de acuarelas. Actualmente sus combos incluyen acuarelas de color Rojo, Verde, Azul y Negro.

Las cajas tienen cuatro acuarelas y recibiremos como argumento sus colores para armar la caja. Por ejemplo, si lo invocamos haciendo `ArmarComboDeAcuarelas(Rojo, Azul, Azul, Verde)` la caja deberá verse así:



Definí el procedimiento `ArmarComboDeAcuarelas` que recibe 4 colores y arma la caja de acuarelas. El cabezal comienza en el extremo Oeste y no importa dónde finaliza.

```
1 procedure ArmarComboDeAcuarelas(color1, color2, color3,
2   color4) {
3   Poner(color1)
4   Mover(Este)
5   Poner(color2)
6   Mover(Este)
7   Poner(color3)
8   Mover(Este)
9   Poner(color4)
10 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:



Tablero inicial



Tablero final



## Ejercicio 2: Ejercicio 2

Una florería nos pidió un procedimiento que se encargue de armar ramos con distintas flores . Actualmente tienen rosas, tulípanes, orquídeas y fresias que representaremos con bolitas de color Rojo, Azul, Negro y Verde respectivamente. Los ramos tienen cuatro flores que recibiremos como argumento para poder armarlos con una flor al lado de la otra. Por ejemplo, si lo invocamos haciendo

`ArmarFlores(Azul, Negro, Rojo, Azul)` el ramo deberá verse así:



Definí el procedimiento `ArmarFlores` que recibe 4 colores y arma el ramo de flores. El cabezal comienza en el extremo Oeste y no importa dónde finaliza.

```
1 procedure ArmarFlores(color1, color2, color3, color4){
2   Poner(color1)
3   Mover(Este)
4   Poner(color2)
5   Mover(Este)
6   Poner(color3)
7   Mover(Este)
8   Poner(color4)
9
10 }
```

▶ Enviar

## Ejercicio 2: Ejercicio 2

Un emprendimiento que vende medias nos pidió un procedimiento que se encargue de armar un combo de medias de distintos colores . Actualmente venden medias de color Rojo, Verde, Negro y Azul. Los combos tienen cuatro pares de medias que pueden especificarse al hacer el pedido. Por ejemplo `AgregarMedias(Azul, Rojo, Negro, Negro)` nos dará como resultado el siguiente tablero :



Es decir, una bolita de color Azul , al Este una de color Rojo , al Este una de color Negro y al Este una de color Negro .

Definí el procedimiento `AgregarMedias` que recibe cuatro colores como argumento y arme los combos de medias. El cabezal comienza en el extremo Oeste y no importa dónde termina.

```
1 procedure AgregarMedias (color1, color2, color3,
2   color4){
3   Poner(color1)
4   Mover(Este)
5   Poner(color2)
6   Mover(Este)
7   Poner(color3)
8   Mover(Este)
9   Poner(color4)
10 }
```

▶ Enviar



Te quedan 01:46:19

## Ejercicio 3: Ejercicio 3

JS

Sabemos que no es saludable para nuestros oídos escuchar música a volúmenes muy altos 🔊. Sin embargo, si está muy bajita tampoco escucharemos. Lo ideal es escucharla a un nivel entre 22 y 72. Para ello tenemos la función `esRecomendable`:

```

esRecomendable(40)
true // Porque está entre 22 y 72

esRecomendable(19)
false // Porque es menor que 22

esRecomendable(80)
false // Porque es mayor que 72

```

Definí la función `esRecomendable` que dado un volumen nos diga si está en el rango recomendable.

Solución &gt; Consola

1 ...escribí tu solución acá...

▶ Enviar

## Ejercicio 3: Ejercicio 3

Dejemos atrás Gobstones y pasemos a JavaScript. 🐼

En este ejercicio vamos a definir una función que nos permita saber si la tierra es fértil para plantar 🌱, es decir, si su ph es 5 y su porcentaje de humedad está entre 26 y 39. Por ejemplo:

```

esTierraFertil(5, 30)
true

esTierraFertil(5, 20)
false //porque 20 es menor que 26

esTierraFertil(3, 30)
false //porque 3 es distinto de 5

```

Definí la función `esTierraFertil` que reciba un ph y un porcentaje de humedad y nos diga si la tierra es fértil.

Solución &gt; Consola

```

1 function esTierraFertil(ph, humedad) {
2   return (ph===5) && (humedad>=26) && (humedad<=39)
3 }

```

▶ Enviar

## Ejercicio 3: Ejercicio 3

Dejemos atrás Gobstones y pasemos a JavaScript. 🐼

3. Ejercicio 3

En este ejercicio vamos a definir una función que nos permita saber si el agua es potable 💧, es decir, si su ph está entre 7 y 9.1 y tiene 0.5 mg/L de cloro. Por ejemplo:

```

esConsumible(8, 0.5)
true

esConsumible(6, 0.5)
false //porque 6 es menor que 7

esConsumible(8, 1.2)
false //porque 1.2 es distinto de 0.5

```

Definí la función `esConsumible` que reciba un ph y una cantidad de cloro y nos diga si el agua es potable.

Solución &gt; Consola

```

1 function esConsumible(ph, cloro){
2   return ((ph>=7) && (ph<=9.1) && cloro===0.5)}

```

▶ Enviar

# Ejercicio 3: Ejercicio 3

Dejemos atrás Gobstones y pasemos a JavaScript. ☐

Ro está programando una aplicación con distintas funcionalidades del clima ☐. En esta oportunidad nos pidió ayuda para definir una función que nos indique si el clima está agradable la cuál llamaremos `esDialindo`. Para que esto suceda la temperatura tiene que ser `20` y no tiene que estar lloviendo. Por ejemplo:

```
esDialindo(20, false)
true

esDialindo(20, true)
false //porque está lloviendo

esDialindo(18, false)
false //porque la temperatura es menor a 20
```

Definí la función `esDialindo` que recibe como argumentos una temperatura y un booleano que indica si está lloviendo.

[Solución](#) [> Consola](#)

```
1 function esDialindo (temp, llueve){
2   return temp === 20 && !llueve
3 }
```

▶ Enviar

Argentina Programa / Examen #SeProgramar - Diciembre 2021 T1 / 4. Ejercicio 4



⌚ Te quedan 01:42:24

## Ejercicio 4: Ejercicio 4

JS

Vamos a desarrollar un GPS que nos recomiende un destino a partir de una dirección 📍. Para ello definiremos una función que reciba una dirección y dos destinos y según el valor del primer argumento nos recomiende hacia donde ir. Las únicas direcciones posibles son `"noreste"` y `"sur"`. En caso que el primer argumento sea `"noreste"` nos dirá que vayamos al primer destino, si es `"sur"` nos recomendará que vayamos al segundo:

```
h dondeVamos("noreste", "Gral. Las Heras", "Merlo")
"Vamos a Gral. Las Heras"

h dondeVamos("sur", "Iguazú", "El Pato")
"Vamos a El Pato"
```

Definí la función `adondeVamos`.

🔗 ¡Dame una pista!

[Solución](#) [> Consola](#)

```
1 ...escribí tu solución acá...
```

▶ Enviar

## Ejercicio 4: Ejercicio 4

JS

Un local que vende productos de forma mayorista y minorista nos pidió ayuda para simplificar qué precio le corresponde a un producto a partir de las unidades a comprar.



Para ello definiremos una función que reciba las unidades a comprar, el precio minorista y el precio mayorista por unidad. Sabemos que a partir de 18 unidades venden a precio mayorista. Por ejemplo:

```
precioACobrarPorUnidad(18, 60, 50)
50

precioACobrarPorUnidad(19, 80, 60)
60

precioACobrarPorUnidad(17, 20, 10)
20
```

Definí la función `precioACobrarPorUnidad`.

**Solución** > Consola

```
1 function precioACobrarPorUnidad(unidades, precioMi, precioMa) {
2   if (unidades >= 18) {
3     return precioMa
4   } else
5     return precioMi
6 }
```

▶ Enviar

## Ejercicio 4: Ejercicio 4

Un cine nos pidió ayuda para automatizar el sistema de cobros. Para ello definiremos una función que reciba una edad, un precio para menores y uno para mayores. Sabemos que a partir de 15 años corresponde cobrar el precio para mayores. Por ejemplo:

```
precioSegunEdad(15, 100, 200)
200

precioSegunEdad(16, 120, 210)
210

precioSegunEdad(14, 150, 250)
150
```

Definí la función `precioSegunEdad`.

**Solución** > Consola

```
1 function precioSegunEdad (edad, precio1, precio2){
2   if (edad >= 15) {return precio2}
3   else { return precio1 }}
```

## Ejercicio 4: Ejercicio 4

En este ejercicio necesitamos definir una función que nos indique si una serie terminó. Para saber si terminó o no vamos a considerar que las series que emitieron su última temporada antes del 2010 están finalizadas.

Para ello vamos a definir la función `situacionSerie` que recibe un año como argumento y retornará "Concluida" o "Pendiente" según corresponda. Por ejemplo:

```
situacionSerie(2007)
"Concluida"

situacionSerie(2021)
"Pendiente"
```

Definí la función `situacionSerie` con el argumento correspondiente.

**Solución** > Consola

```
1 function situacionSerie(anio){
2   if (anio >= 2010) {
3     return "Pendiente"
4   }
5   else return "Concluida"
6 }
```


▶ Enviar



Te quedan 01:36:47

## Ejercicio 5: Ejercicio 5

JS

Un local gastronómico quiere clasificar su vajilla  y contar cuántos "recipiente" s tiene a partir de una lista:

```

cuantosHay(["jarra", "recipiente", "taza", "recipiente", "recipiente", "bowl"])
3

cuantosHay(["recipiente", "taza", "taza", "bowl"])
1

```


Definí la función `cuantosHay` que a partir de una lista con la vajilla nos dice la cantidad de "recipiente" s que tiene.

☒ Solución > Consola

```
1 ...escribí tu solución acá...
```

## Ejercicio 5: Ejercicio 5

JS

Tanto en libros e historietas como en series y películas nos podemos encontrar personajes ficticios. Algunos de Argentina, pero también de otros países. Nos gustaría poder saber a partir de una lista de personajes cuáles son argentinos . Para ello contamos con la función `esPersonajeArgentino` que recibe un personaje y nos devuelve un booleano:

☒ Solución > Consola

```

1 function personajesDeArgentina(personajes) {
2   let listaArgentinos = []
3   for(let personaje of personajes)
4   {
5     if(esPersonajeArgentino(personaje)) {
6       agregar(listaArgentinos, personaje)
7     }
8   }
9   return listaArgentinos
10 }

```

```

esPersonajeArgentino("Patoruzú")
true

esPersonajeArgentino("Mafalda")
true

esPersonajeArgentino("Mujer Maravilla")
false

```

Definí la función `personajesDeArgentina` que recibe una lista de personajes y nos retorna una nueva lista solo con aquellos que son argentinos. No hace falta que definas `esPersonajeArgentino`, solo que la invoques. Por ejemplo:



```

personajesDeArgentina(["Mafalda", "Batman", "Lisa", "Oakly"])
["Mafalda", "Oakly"]

```

▶ Enviar

## Ejercicio 5: Ejercicio 5

Nuestro planeta tiene lugares hermosos por doquier pero vamos a centrarnos en aquellos que estén en nuestra bella América Latina . Lo que vamos a hacer en este ejercicio es obtener a partir de una lista con lugares turísticos cuántos quedan en Latinoamérica . Para ello contamos con la función `quedaEnAmericaLatina` que recibe un lugar y retorna un booleano:

☒ Solución > Consola

```

1 function cuantosDeAmericaLatina (lugares){
2   let listaDeLugares=[]
3   for (let lugar of lugares){
4     if (quedaEnAmericaLatina (lugar)){
5       listaDeLugares.push (lugar)
6     }
7   }
8   return listaDeLugares.length
9 }

```

```

quedaEnAmericaLatina("Machu Pichu")
true

quedaEnAmericaLatina("Cataratas de Iguazú")
true

quedaEnAmericaLatina("Torre Eiffel")
false

```

Definí la función `cuantosDeAmericaLatina` que recibe una lista de lugares turísticos y nos retorna cuántos quedan en América Latina. No hace falta que definas

▶ Enviar



# Ejercicio 5: Ejercicio 5

A veces queremos llevar registro de la distancia recorrida y calcularla puede ser una tarea tediosa. ☐ ¡Pero la programación puede ayudarnos! ☐

Para eso vamos a definir una función que reciba una lista con la cantidad de kilómetros recorridos por una persona y nos retorne el total en metros. Por ejemplo:

```
metrosRecorridos([1,3,20,2])
26000 // porque 1 + 3 + 20 + 2 = 26 y 26 * 1000 = 26000

metrosRecorridos([1,0,1,2])
4000 // porque 1 + 0 + 1 + 2 = 4 * y 4 * 1000 = 4000
```

Definí la función `metrosRecorridos`.

 Solución  Consola

```
1 function metrosRecorridos(kms){
2   let metros = 0
3   for (let km of kms)
4     metros = metros + (km*1000)
5
6   return metros
7 }
```

Argentina Programa / Examen #SeProgramar - Diciembre 2021 T1 / 6. Ejercicio 6



 Te quedan 01:25:09

## Ejercicio 6: Ejercicio 6

JS

En una casa de comidas guardan registro de los envíos que realizan a sus clientes 🍱. Estos registros tienen la siguiente forma:

```
let envioCalleFalsa = {
  direccion: "Calle Falsa 123",
  pedidos: ["Muzzarella", "Empanadas de verdura", "Papas fritas"],
  ultimoPedido: "15/11/2021"
}

let envioWallaby = {
  direccion: "Wallaby 42",
  pedidos: ["Ravioles con fileto", "10 piezas de sushi"],
  ultimoPedido: "16/12/2021"
}
```

Definí la función `resumenInformacion` que permita obtener un resumen de la información registrada de esta manera:

```
resumenInformacion(envioCalleFalsa)
"Calle Falsa 123 tiene como fecha de último pedido el 15/11/2021 y realizó en total 3 pedidos"

resumenInformacion(envioWallaby)
"Wallaby 42 tiene como fecha de último pedido el 16/12/2021 y realizó en total 2 pedidos"
```

 Solución  Consola

1 ...escribí tu solución acá...

 Enviar

## Ejercicio 6: Ejercicio 6

JS

A partir de un censo nos pidieron poder pasar en limpio la información de los datos obtenidos. Para ello contamos con registros habitacionales de distintas familias. Por ejemplo:

```
let casaChaldu = {
  familia: "Chaldu",
  integrantes: ["Elena", "Tita", "Marucha"],
  direccion: "Pordomingo 400"
}

let casaDominguez = {
  familia: "Dominguez",
  integrantes: ["Olivia", "Micaela", "Sol", "Alex"],
  direccion: "Belgrano 350"
}
```

Definí la función `resumenDeInformacion` que permita obtener un resumen de la información registrada. Por ejemplo:

```
resumenDeInformacion(casaChaldu)
"La familia Chaldu cuenta con 3 integrantes que viven en Pordomingo 400"

resumenDeInformacion(casaDominguez)
"La familia Dominguez cuenta con 4 integrantes que viven en Belgrano 350"
```

**Solución** > Consola

```
1 function resumenDeInformacion (casas) {
2   return "La familia " + casas.familia + " cuenta con " +
  (longitud(casas.integrantes)) + " integrantes" + " que viven
  en " + casas.direccion
3 }
```

▶ Enviar

## Ejercicio 6: Ejercicio 6

Una revista de videojuegos cuenta con datos sobre los títulos que salen en el mercado y quieren publicar la información registrada en su sección de lanzamientos. Los registros tienen la siguiente forma:

```
let assassinsCreedValhalla = {
  titulo: "Assassins Creed Valhalla",
  plataformas: ["PlayStation 4", "PlayStation 5", "Xbox One", "Xbox Series X|S", "PC"],
  compania: "Ubisoft"
}

let pes2022 = {
  titulo: "PES 2022",
  plataformas: ["PlayStation 4", "PlayStation 5", "Xbox One", "Xbox Series X|S", "PC", "Android", "iOS"],
  compania: "Konami"
}
```

6. Ejercicio 6

**Solución** > Consola

```
1 function resumenDeJuego (juego){
2   return "El título " + juego.titulo + " de la compañía " +
  juego.compania + " corre en " + juego.plataformas.length + "
  plataformas"
3 }
```

▶ Enviar

## Ejercicio 6: Ejercicio 6

En un curso de química, nos pidieron una función que se encargue de hacer un resumen de la información de los elementos de la tabla periódica. Para ello contamos con registros como estos:

```
let radio = {
  elemento: "Radio",
  anioDeDescubrimiento: 1898,
  fueDescubiertoPor: "Madame Curie"
}

let krypton = {
  elemento: "Kriptón",
  anioDeDescubrimiento: 1898,
  fueDescubiertoPor: "William Ramsay"
}
```

Definí la función `resumenDelAtomo` que permita obtener un resumen de la información registrada. Por ejemplo:

```
resumenDelAtomo(radio)
```

**Solución** > Consola

```
1 function resumenDelAtomo(atomo){
2   return "El elemento " + atomo.elemento + " fue
  descubierto en el año " +
  atomo.anioDeDescubrimiento + " por " +
  atomo.fueDescubiertoPor
3 }
```

▶ Enviar



Te quedan 01:23:29

## Ejercicio 7: Ejercicio 7



¡Dejemos atrás a JavaScript para pasar a Ruby! 🐾

En esta ocasión queremos desarrollar parte de un juego, para ello vamos a modelar a su personaje principal: `Atrix`. Este personaje va a recolectar monedas y sabemos que:

- inicialmente tiene 8 monedas;
- puede duplicar sus monedas;
- si tiene más de 72 monedas diremos que es `profesional?`.

Definí en Ruby, el objeto `Atrix` que tenga un atributo `@monedas` con su getter. El objeto entiende los mensajes `duplicar_monedas!` (que multiplica por 2 su cantidad de monedas) y `profesional?`. No te olvides de inicializar el atributo `@monedas` con el valor correspondiente.

Solución

&gt; Consola

1 ...escribí tu solución acá...

## Ejercicio 7: Ejercicio 7

Acércandonos al final del examen vamos a pasar al mundo de los objetos. ☐

`Almendra` es una artista plástica que se dedica a la venta de cuadros y sabemos que:

- tiene una cantidad de cuadros que inicialmente es `0`;
- esa cantidad aumenta en 1 cada vez que termina un cuadro;
- la cantidad de cuadros disminuye en 1 cuando vende un cuadro. ☐

Definí al objeto `Almendra` con su atributo `cantidad_cuadros`. El objeto entiende los mensajes `finalizar_cuadro!` y `vender_cuadro!`. No te olvides de inicializar el atributo con el valor correspondiente.

Solución

&gt; Consola

```
1 module Almendra
2   @cantidad_cuadros=0
3   def self.finalizar_cuadro!
4     @cantidad_cuadros +=1
5   end
6   def self.vender_cuadro!
7     @cantidad_cuadros -=1
8   end
9 end
```

## Ejercicio 7: Ejercicio 7

Acércandonos al final del examen vamos a pasar al mundo de los objetos. 🐾

`Fran` es docente de música 🎵 y sabemos que:

- su cantidad de cursos inicialmente es `1`;
- cada uno de sus cursos es de `3` horas;
- puede tomar un nuevo curso si tiene menos de `9` cursos.

Definí al objeto `Fran` con su atributo `cantidad_de_cursos`. El objeto entiende los mensajes:

- `cantidad_de_horas_totales` que retorna la cantidad de horas de todos sus cursos;
- `puede_tomar_nuevo_curso?` que nos dice si puede tomar un nuevo curso.

No te olvides de inicializar el atributo con el valor correspondiente.

Solución

&gt; Consola

7. Ejercicio 7

```
1 module Fran
2   @cantidad_de_cursos = 1
3   def self.cantidad_de_cursos
4     @cantidad_de_cursos
5   end
6   def self.cantidad_de_horas_totales
7     @cantidad_de_cursos*3
8   end
9   def self.puede_tomar_nuevo_curso?
10    @cantidad_de_cursos<9
11  end
12 end
```

## Ejercicio 7: Ejercicio 7

Acércandonos al final del examen vamos a pasar al mundo de los objetos. ☐

`Numir` es un planeta del cuál sabemos que:

- tiene una cantidad de cráteres que inicialmente es `63`;
- puede sufrir una lluvia de meteoros, cuando esto sucede su cantidad de cráteres aumenta tanto como la cantidad de meteoros que caigan;
- diremos que tiene demasiados cráteres si tiene más de `169`. ☐

Definí al objeto `Numir` con su atributo `cantidad_de_crateres`. El objeto entiende los mensajes `sufrir_lluvia_meteoritos!` que toma una cantidad de meteoros por parámetro, y `exceso_de_crateres?`. No te olvides de inicializar el atributo con el valor correspondiente.

Solución

&gt; Consola

```
1 module Numir
2   @cantidad_de_crateres = 63
3
4   def self.sufrir_lluvia_meteoritos! (meteoros)
5     @cantidad_de_crateres += meteoros
6   end
7
8   def self.exceso_de_crateres?
9     @cantidad_de_crateres > 169
10  end
11
12 end
```



Te quedan 01:16:06

## Ejercicio 8: Ejercicio 8

En un curso tenemos un conjunto de estudiantes, a la hora de cerrar las actas es necesario saber cuántas personas aprobaron ☒. Teniendo en cuenta que cada estudiante sabe responder al mensaje `ha_aprobado?` ...

Definí en Ruby el método `cantidad_de_gente_aprobada` que responda a cuántas personas aprobaron de `Estudiantado`.

[Solución](#) [\\_ Consola](#)

```
1 ...escribí tu solución acá...
```

## Ejercicio 8: Ejercicio 8

La Ley 26.588 establece que los restaurantes deben tener como mínimo una opción sin TACC (sin gluten) en su menú ☐. Teniendo en cuenta que nuestro restaurante `Restuki` tiene una lista con las distintas comidas y que cada una sabe responder al mensaje `no_tiene_gluten?` ...

Definí en Ruby el método `cuantos_sin_tacc` que responda a cuántas opciones del menú de `Restuki` son sin TACC.

[Solución](#) [\\_ Consola](#)

```
1 module Restuki
2   @menu = [Tortilla, BerenjenasAlEscabeche, FideosDeArroz,
3     Pizza, Canelones, Helado]
4   def self.cuantos_sin_tacc
5     @menu.count { |menu| menu.no_tiene_gluten? }
6   end
7 end
```

## Ejercicio 8: Ejercicio 8

La plataforma de películas `MumuPlus` necesita de nuestra ayuda para poder saber cuántas de ellas son aptas para todo público ☐. Teniendo en cuenta que `MumuPlus` tiene una lista con las distintas películas y que cada una sabe responder al mensaje `es_apta_todo_publico?` ...

Definí en Ruby el método `cuantas_atp` que responda a cuántas películas son aptas para todo público.

[Solución](#) [\\_ Consola](#)

```
1 module MumuPlus
2   @peliculas = [ElHijoDeLaNovia, ToyStory, Contacto,
3     EsperandoLaCarroza, Memento, Intensamente]
4   def self.cuantas_atp
5     @peliculas.count {|peliculas|
6       peliculas.es_apta_todo_publico?}
7   end
8 end
```

## Ejercicio 8: Ejercicio 8

Salvor tiene una huerta con distintas frutas y verduras, y necesita saber cuáles de ellas están listas para ser cosechadas ☐ ☐ ☐ ☐. Teniendo en cuenta que `HuertaDeSalvor` tiene una lista con los distintos alimentos y que cada uno sabe responder al mensaje `para_cosechar?` ...

Definí en Ruby el método `cuales_para_cosechar` que responda cuáles frutas y verduras de `HuertaDeSalvor` están listas para ser cosechadas.

[Solución](#) [\\_ Consola](#)

```
1 module HuertaDeSalvor
2   @alimentos = [Tomate, Berenjena, Frutilla,
3     Frambuesa, Palta, Zapallo, Papa, Batata]
4   def self.cuales_para_cosechar
5     @alimentos.select {|frutasyverd|
6       frutasyverd.para_cosechar?}
7   end
8 end
```

Te quedan 01:10:14

## Ejercicio 9: Ejercicio 9



A la hora de hacer turismo, es recomendable tener en cuenta qué lugares son interesantes para recorrerlos 📍. Sabemos que:

- Los `Monumento`s son interesantes si tienen más de 300 años.
- Los `Museo`s siempre son interesantes.
- Los `Puente`s no son interesantes.

Definí el método `atracciones_interesantes` en la clase `Localidad` que devuelva un listado de atracciones interesantes. Para eso deberás definir el método `interesante?` en los distintos tipos de atracciones.

[Solución](#) > [Consola](#)

```
1 class Localidad
2   def initialize(unas_atracciones)
3     @atracciones = unas_atracciones
4   end
5 end
6
7 class Monumento
8   def initialize(unos_anios)
9     @anios = unos_anios
10  end
11 end
12
13 class Museo
14 end
15
16 class Puente
17 end
```

## Ejercicio 9: Ejercicio 9

En una juguetería venden distintos juguetes donde cada uno tiene su precio. De ellos sabemos que:

- los `Rompecabezas` cuestan \$4 por cada pieza que tengan;
- cada `Autito` tiene su propio precio;
- las `Pista`s tienen un precio de \$476.

A partir de esto, las jugueterías quieren saber cuál es el precio total de todos los juguetes que venden. 🐱

Definí el método `precio_de_juguetes` en la clase `Jugueteria` que retorne la sumatoria de precios, y el método `precio_de_venta` en las distintas clases de juguetes.

[Solución](#) > [Consola](#)

```
1 class Jugueteria
2   def initialize(unos_juguetes)
3     @juguetes = unos_juguetes
4   end
5
6   def precio_de_juguetes
7     @juguetes.sum{|juguete| juguete.precio_de_venta}
8   end
9 end
10
11 class Rompecabezas
12   def initialize(unas_piezas)
13     @piezas = unas_piezas
14   end
15
16   def precio_de_venta
17     4*@piezas
18   end
19 end
20
21 class Autito
22   def initialize(un_precio)
23     @precio = un_precio
24   end
25
26   def precio_de_venta
27     @precio
28   end
29 end
30
31 class Pista
32   def precio_de_venta
33     476
34   end
35 end
36 end
```

En una juguetería venden distintos juguetes donde cada uno tiene su precio. De ellos sabemos que:

- los `Rompecabezas` cuestan \$6 por cada pieza que tengan;
- cada `Peluche` tiene su propio precio;
- las `Pelotas` tienen un precio de \$491.

A partir de esto, las jugueterías quieren saber cuál es el precio total de todos los juguetes que venden. □

Defini el método `precio_total` en la clase `Jugueteria` que retorna la sumatoria de precios, y el método `precio` en las distintas clases de juguetes.

**Solución** >\_ Consola

```
1 class Jugueteria
2   def initialize(unos_juguetes)
3     @juguetes = unos_juguetes
4   end
5   def precio_total
6     @juguetes.sum { |juguetes| juguetes.precio }
7   end
8 end
9
10 class Rompecabezas
11   def initialize(unas_piezas)
12     @piezas = unas_piezas
13     @precio = (unas_piezas * 6)
14   end
15   def precio
16     @precio
17   end
18 end
19
20 class Peluche
21   def initialize(un_precio)
22     @precio = un_precio
23   end
24   def precio
25     @precio
26   end
27 end
28
29 class Pelota
30   def initialize
31     @precio = 491
32   end
33   def precio
34     @precio
35   end
36 end
37 end
```



Quienes estudian veterinaria nos pidieron un sistema para analizar el efecto de alimentar a un animal que tengan a su cuidado 🐾. Al comer no todos los animales responden de la misma forma:

- los `Perro`s duplican su energía;
- las `Lagartija`s no muestran ningún efecto;
- las `Aguila`s aumentan su energía en `25`.

Definí el método `dar_alimento!` en la clase `EstudianteDeVeterinaria`, y el método `recibir_comida!` en las distintas clases de animales.

**Solución** > Consola

```
1 class EstudianteDeVeterinaria
2   def initialize(un_animal)
3     @animal = un_animal
4   end
5
6   def dar_alimento!
7     @animal.recibir_comida!
8   end
9
10 end
11
12 class Perro
13   def recibir_comida!
14     @energia *= 2
15   end
16   def initialize(una_energia)
17     @energia = una_energia
18   end
19
20   def energia
21     @energia
22   end
23 end
24
25 class Lagartija
26   def initialize(una_energia)
27     @energia = una_energia
28   end
29   def recibir_comida!
30
31   end
32   def energia
33     @energia
34   end
35 end
36
37 class Aguila
38   def initialize(una_energia)
39     @energia = una_energia
40   end
41   def recibir_comida!
42     @energia += 25
43   end
44   def energia
45     @energia
46   end
47 end
```

# Ejercicio 9: Ejercicio 9

A la hora de viajar ☐ las personas tienen que elegir dónde llevar sus pertenencias. Para elegir el equipaje es importante tener en cuenta que:

- las **Carteras** admiten 1 kilos por bolsillo;
- las **Maletas** pueden llevar 12 kilos;
- los **Bolsos** tienen un peso permitido en kilos que depende de cada uno.

Definí el método `cuanto_puede_transportar` en la clase `Persona` y el método `peso_limite` en las distintas clases de equipaje.

💡 ¡Dame una pista!

 **Solución**  **Consola**

```
1 class Persona
2   def initialize(un_equipaje)
3     @equipaje = un_equipaje
4   end
5
6   def cuanto_puede_transportar
7     @equipaje.peso_limite
8   end
9 end
10
11 class Cartera
12   def initialize(unos_bolsillos)
13     @bolsillos = unos_bolsillos
14   end
15
16   def peso_limite
17     @bolsillos
18   end
19 end
20
21 class Maleta
22   def peso_limite
23     12
24   end
25 end
26
27 class Bolso
28   def initialize(un_peso)
29     @peso = un_peso
30   end
31
32   def peso_limite
33     @peso
34   end
35 end
```