Introducción a Listas

Francisco Soulignac

Tecnicatura en Programación Informática,
Universidad Nacional de Quilmes

Introducción a la programación
Algoritmos y Programación

¿Qué vemos hoy?

XGobstones

- XGobstones es una extensión de Gobstones que provee:
  - Listas y registros: que vemos en la materia
  - Procedimientos sobre otros tipos: no lo vemos

XGobstones

- XGobstones es una extensión de Gobstones que provee:
  - Listas y registros: que vemos en la materia
  - Procedimientos sobre otros tipos: no lo vemos

- Hay que adaptar los procedimientos y funciones de Gobstones

XGobstones

- XGobstones es una extensión de Gobstones que provee:
  - Listas y registros: que vemos en la materia
  - Procedimientos sobre otros tipos: no lo vemos

- Hay que adaptar los procedimientos y funciones de Gobstones
- Si una función usa el tablero tiene un parámetro más
  - Por convención es el primer parámetro y lo llamamos `t`

XGobstones

- XGobstones es una extensión de Gobstones que provee:
  - Listas y registros: que vemos en la materia
  - Procedimientos sobre otros tipos: no lo vemos

- Hay que adaptar los procedimientos y funciones de Gobstones
- Si una función usa el tablero tiene un parámetro más
  - Por convención es el primer parámetro y lo llamamos `t`
- Los procedimiento que modifican el tablero tienen un parámetro "por fuera"
  - Por convención tambien lo llamamos `t`

XGobstones: adaptación

```
//Ponemos t antes del nombre como parametro
procedure t.PonerUnaDeCada() {
1. foreach c in [minColor()..maxColor()]
2. {
       //Análogamente, ponemos t antes del nombre como argumento
3.     t.Poner(c)
4. }
}
```

XGobstones: adaptación

```
//Ponemos t antes del nombre como parametro
procedure t.PonerUnaDeCada() {
1. foreach c in [minColor()..maxColor()]
2. {
       //Análogamente, ponemos t antes del nombre como argumento
3.     t.Poner(c)
4. }
}

//Ponemos t como primer parametro
function esCromatica(t) {
1. cromatica := True
2. foreach c in [minColor()..maxColor()]
3. {
       //Análogamente, usamos t como primer argumento
4.     cromatica := cromatica && hayBolitas(t, c)
5. }
6. return(cromatica)
}
```

XGobstones: adaptación

```
//Ponemos t antes del nombre como parametro
procedure t.PonerUnaDeCada() {
1. foreach c in [minColor()..maxColor()]
2. {
       //Análogamente, ponemos t antes del nombre como argumento
3.     t.Poner(c)
4. }
}

//Ponemos t como primer parametro
function esCromatica(t) {
1. cromatica := True
2. foreach c in [minColor()..maxColor()]
3. {
       //Análogamente, usamos t como primer argumento
4.     cromatica := cromatica && hayBolitas(t, c)
5. }
6. return(cromatica)
}

t.program {
1. t.PonerUnaDeCada()
2. t.BoomSi(not esCromatica(t))
}
```

Motivación I

- Supongamos que estamos programando el juego Sokoban
- Para evitar tener que reiniciar el juego cada vez que se pierde, se quiere ofrecer la posibilidad de deshacer una movida.

- ¿Sugerencias para implementarlo?

Motivación II

- Estamos programando el buscaminas.
- Queremos usar el tablero y las bolitas para visualizar, pero no queremos que se puedan ver las pistas y minas debajo de las celdas tapadas

- ¿Sugerencias para implementarlo?

Idea informal

- Una lista es una colección de valores llamados elementos
  - Sokoban: lista de jugadas
  - Buscaminas: lista de minas, lista de lugares destapados

Idea informal

- Una lista es una colección de valores llamados elementos
  - Sokoban: lista de jugadas
  - Buscaminas: lista de minas, lista de lugares destapados
- Cada valor puede aparecer muchas veces en la lista
  - Sokoban: movimiento hacia la derecha

Idea informal

- Una lista es una colección de valores llamados elementos
  - Sokoban: lista de jugadas
  - Buscaminas: lista de minas, lista de lugares destapados
- Cada valor puede aparecer muchas veces en la lista
  - Sokoban: movimiento hacia la derecha
- Los valores tienen un orden en la lista
  - Sokoban: ¿primero arriba o a la derecha?
  - Buscaminas: posición de una mina = posición en un recorrido del tablero

Definición

Definición

Una lista es una colección (finita) ordenada de valores de un mismo tipo.

Definición

Definición

Una lista es una colección (finita) ordenada de valores de un mismo tipo.

- Las listas se escriben $[v_1, v_2, v_3, \ldots, v_n]$ donde:

Definición

Definición

Una lista es una colección (finita) ordenada de valores de un mismo tipo.

- Las listas se escriben $[v_1, v_2, v_3, \ldots, v_n]$ donde:
  1. v_1 es el primer elemento,

Definición

Definición

Una lista es una colección (finita) ordenada de valores de un mismo tipo.

- Las listas se escriben $[v_1, v_2, v_3, \ldots, v_n]$ donde:
  1. v_1 es el primer elemento,
  2. v_2 es el segundo elemento,

Definición

Definición

Una lista es una colección (finita) ordenada de valores de un mismo tipo.

- Las listas se escriben $[v_1, v_2, v_3, \ldots, v_n]$ donde:
  1. v_1 es el primer elemento,
  2. v_2 es el segundo elemento,
  3. v_3 es el tercer elemento,

Definición

Definición

Una lista es una colección (finita) ordenada de valores de un mismo tipo.

- Las listas se escriben $[v_1, v_2, v_3, \ldots, v_n]$ donde:
  1. v_1 es el primer elemento,
  2. v_2 es el segundo elemento,
  3. v_3 es el tercer elemento,
  4. $\ldots$
  5. v_n es el último elemento,

Definición

Definición

Una lista es una colección (finita) ordenada de valores de un mismo tipo.

- Las listas se escriben $[v_1, v_2, v_3, \ldots, v_n]$ donde:
  1. v_1 es el primer elemento,
  2. v_2 es el segundo elemento,
  3. v_3 es el tercer elemento,
  4. ...
  5. v_n es el último elemento,

- La cantidad de elementos n es la longitud de la lista

Ejemplos

- [**1**, **2**, **3**] es la lista con los valores **1**, **2**, **3**

Ejemplos

- [**1**, **2**, **3**] es la lista con los valores **1**, **2**, **3**
- [**1**, **-1**, **1**, **-1**, **1**, **-1**]] es la lista que alterna 3 veces entre **1** y **-1**, empezando por **1**

Ejemplos

- [**1**, **2**, **3**] es la lista con los valores **1**, **2**, **3**
- [**1**, **-1**, **1**, **-1**, **1**, **-1**]] es la lista que alterna 3 veces entre **1** y **-1**, empezando por **1**
- [↑, →, ↓, ←] es la lista de direcciones ordenada en el sentido de las agujas del reloj, empezando por ↑

Ejemplos

- [**1**, **2**, **3**] es la lista con los valores **1**, **2**, **3**
- [**1**, **-1**, **1**, **-1**, **1**, **-1**]] es la lista que alterna 3 veces entre **1** y **-1**, empezando por **1**
- [↑, →, ↓, ←] es la lista de direcciones ordenada en el sentido de las agujas del reloj, empezando por ↑
- [Verdadero, Verdadero, Verdadero, Verdadero] es una lista con 4 Verdadero

Ejemplos

- [**1**, **2**, **3**] es la lista con los valores **1**, **2**, **3**
- [**1**, **-1**, **1**, **-1**, **1**, **-1**]] es la lista que alterna 3 veces entre **1** y **-1**, empezando por **1**
- [↑, →, ↓, ←] es la lista de direcciones ordenada en el sentido de las agujas del reloj, empezando por ↑
- [Verdadero, Verdadero, Verdadero, Verdadero] es una lista con 4 Verdadero

- [] es la lista vacía

Ejemplos inválidos

¿Por qué los siguientes ejemplos son inválidos?

- $[\mathbf{1}, \uparrow, \mathbf{2}, \rightarrow]$

Ejemplos inválidos

¿Por qué los siguientes ejemplos son inválidos?

- [**1**, ↑, **2**, →]
- [→, Verdadero, Verdadero]

Ejemplos inválidos

¿Por qué los siguientes ejemplos son inválidos?

- [**1**, ↑, **2**, →]
- [→, VERDADERO, VERDADERO]
- La lista de todos los números pares.

Valores, expresiones (recordatorio)

- Valor: igual a sí mismo y distinto del resto
- Expresión: forma de denotar un valor
  - Distintas expresiones pueden denotar el mismo valor
  - La misma expresión puede denotar distintos valores
  - Se usan como argumentos y SOLO como argumentos

Valores, expresiones (recordatorio)

- Valor: igual a sí mismo y distinto del resto
- Expresión: forma de denotar un valor
  - Distintas expresiones pueden denotar el mismo valor
  - La misma expresión puede denotar distintos valores
  - Se usan como argumentos y SOLO como argumentos

- Hay que definirlo antes de poder usar listas

Valores de Lista

- Valor: [v_1, ..., v_n], donde $v_1, \ldots, v_n$ son valores del mismo tipo.
  - SI: [**1**, **2**, **3**], [↑], [], etc
  - NO: [**1**, ↑] por distintos tipos de elementos
  - NO: [`minColor`()] ya que `minColor`() no es un valor
- Notar: cada lista es igual a sí misma y distinta del resto de las listas (y valores)

Valores de Lista

- Valor: `[`v_1`, ..., `v_n`]`, donde $v_1, \ldots, v_n$ son valores del mismo tipo.
  - SI: `[`**1**`, `**2**`, `**3**`]`, `[`↑`]`, `[]`, etc
  - NO: `[`**1**`, `↑`]` por distintos tipos de elementos
  - NO: `[minColor()]` ya que `minColor()` no es un valor
- Notar: cada lista es igual a sí misma y distinta del resto de las listas (y valores)

- Como siempre, los valores no se pueden escribir en XGobstones
- Para denotar listas usamos expresiones.

Listas por extension

- Si $e_1, \ldots, e_n$ son expresiones del mismo tipo...

Listas por extension

- Si $e_1, \ldots, e_n$ son expresiones del mismo tipo...
- ...y $v_1, \ldots, v_n$ son los valores denotados por $e_1, \ldots, e_n$...

Listas por extension

- Si $e_1, \ldots, e_n$ son expresiones del mismo tipo...
- ...y $v_1, \ldots, v_n$ son los valores denotados por $e_1, \ldots, e_n$...
- ... $[e_1, \ldots, e_n]$ es una expresión que $[v_1, \ldots, v_n]$

Listas por extension

- Si $e_1, \ldots, e_n$ son expresiones del mismo tipo...
- ...y $v_1, \ldots, v_n$ son los valores denotados por $e_1, \ldots, e_n$...
- ... `[`e_1`, ..., `e_n`]` es una expresión que `[`v_1`, ..., `v_n`]`

- `[1, 2, 3 * 1]` denota la lista **[1, 2, 3]**
- `[Norte, Este, Sur, Oeste]` denota la lista [↑, →, ↓, ←]
- `[siguiente(Azul), previo(Verde)]` denota la lista [•, •]

Listas por extension

- Si $e_1, \ldots, e_n$ son expresiones del mismo tipo...
- ...y $v_1, \ldots, v_n$ son los valores denotados por $e_1, \ldots, e_n$...
- ... `[`e_1`, ..., `e_n`]` es una expresión que `[`v_1`, ..., `v_n`]`

- `[1, 2, 3 * 1]` denota la lista **[1, 2, 3]**
- `[Norte, Este, Sur, Oeste]` denota la lista [↑, →, ↓, ←]
- `[siguiente(Azul), previo(Verde)]` denota la lista [•, •]

- ¿`[hayBolitas(Azul), hayBolitas(Rojo)]`?

Ejercicio I: hola mundo de las listas

- Escribir una función singleton que, dado un valor `x` retorne la lista `[x]`.

Ejercicio I: hola mundo de las listas

- Escribir una función singleton que, dado un valor `x` retorne la lista `[x]`.

```
function singleton(x) {
 1. return([x])
}
```

Ejercicio I: hola mundo de las listas

- Escribir una función singleton que, dado un valor `x` retorne la lista `[x]`.

```
function singleton(x) {
 1. return([x])
}
```

- `[x]` es una expresión...

Ejercicio I: hola mundo de las listas

- Escribir una función singleton que, dado un valor `x` retorne la lista `[x]`.

```
function singleton(x) {
 1. return([x])
}
```

- `[x]` es una expresión...
- ...ergo, como cualquier expresión, puede usarse como argumento y solo como argumento

Rangos

- En teoría, un rango [⟨inicio⟩..⟨final⟩] ES una lista:
  - es una colección de valores,
  - finita,
  - cuyos elementos son del mismo tipo.

Rangos

- En teoría, un rango [⟨inicio⟩..⟨final⟩] ES una lista:
  - es una colección de valores,
  - finita,
  - cuyos elementos son del mismo tipo.

- En la práctica también

Rangos

- En teoría, un rango [⟨inicio⟩..⟨final⟩] ES una lista:
  - es una colección de valores,
  - finita,
  - cuyos elementos son del mismo tipo.

- En la práctica también
- La expresión [⟨inicio⟩..⟨final⟩] denota la lista que:
  - en la primer posición tiene ⟨inicio⟩
  - en la segunda posicion tiene ⟨`siguiente`(inicio)⟩
  - ...
  - en la ultima posicion tiene ⟨final⟩

Rangos

- En teoría, un rango [⟨inicio⟩..⟨final⟩] ES una lista:
  - es una colección de valores,
  - finita,
  - cuyos elementos son del mismo tipo.

- En la práctica también
- La expresión [⟨inicio⟩..⟨final⟩] denota la lista que:
  - en la primer posición tiene ⟨inicio⟩
  - en la segunda posicion tiene ⟨`siguiente`(inicio)⟩
  - ...
  - en la ultima posicion tiene ⟨final⟩

- `[1..3]` denota la lista [**1**, **2**, **3**]

Rangos

- En teoría, un rango [⟨inicio⟩..⟨final⟩] ES una lista:
  - es una colección de valores,
  - finita,
  - cuyos elementos son del mismo tipo.

- En la práctica también
- La expresión [⟨inicio⟩..⟨final⟩] denota la lista que:
  - en la primer posición tiene ⟨inicio⟩
  - en la segunda posicion tiene ⟨`siguiente`(inicio)⟩
  - ...
  - en la ultima posicion tiene ⟨final⟩

- `[1..3]` denota la lista [**1**, **2**, **3**]
- `[minDir()..maxDir()]` denota la lista [↑, →, ↓, ←]

Rangos

- En teoría, un rango `[`⟨inicio⟩`..`⟨final⟩`]` ES una lista:
  - es una colección de valores,
  - finita,
  - cuyos elementos son del mismo tipo.

- En la práctica también
- La expresión `[`⟨inicio⟩`..`⟨final⟩`]` denota la lista que:
  - en la primer posición tiene ⟨inicio⟩
  - en la segunda posicion tiene ⟨`siguiente`(inicio)⟩
  - ...
  - en la ultima posicion tiene ⟨final⟩

- `[1..3]` denota la lista [**1**, **2**, **3**]
- `[minDir()..maxDir()]` denota la lista [↑, →, ↓, ←]
- `[4..0]` denota la lista `[]`

Ejercicio I: hola mundo de los rangos

- Escribir una función rangoAA que, dados dos valores `x` e `y`, retorne el rango de valores entre `x` e `y`, excluyendo `x` e `y`

Ejercicio I: hola mundo de los rangos

- Escribir una función rangoAA que, dados dos valores `x` e `y`, retorne el rango de valores entre `x` e `y`, excluyendo `x` e `y`

```
function rangoAA(x,y) {
1. rango := []
2. if(x < y)
3. {
4.       rango := [siguiente(x)..previo(y)]
5. }
6. return(rango)
}
```

Ejercicio I: hola mundo de los rangos

- Escribir una función rangoAA que, dados dos valores `x` e `y`, retorne el rango de valores entre `x` e `y`, excluyendo `x` e `y`

```
function rangoAA(x,y) {
1. rango := []
2. if(x < y)
3. {
4.       rango := [siguiente(x)..previo(y)]
5. }
6. return(rango)
}
```

- Puedo tener variables que recuerden listas,

Ejercicio I: hola mundo de los rangos

- Escribir una función rangoAA que, dados dos valores `x` e `y`, retorne el rango de valores entre `x` e `y`, excluyendo `x` e `y`

```
function rangoAA(x,y) {
1. rango := []
2. if(x < y)
3. {
4.       rango := [siguiente(x)..previo(y)]
5. }
6. return(rango)
}
```

- Puedo tener variables que recuerden listas,
- y los rangos denotan listas que se pueden asignar a estas variables.

Concatenación de listas: motivación

- ¿Qué expresión denota una lista con 3 elementos •?

Concatenación de listas: motivación

- ¿Qué expresión denota una lista con 3 elementos •?
- ¿Qué expresión denota una lista con 10 elementos •?

Concatenación de listas: motivación

- ¿Qué expresión denota una lista con 3 elementos •?
- ¿Qué expresión denota una lista con 10 elementos •?
- ¿Qué expresión denota una lista con 10000 elementos •?

Concatenación de listas: motivación

- ¿Qué expresión denota una lista con 3 elementos •?
- ¿Qué expresión denota una lista con 10 elementos •?
- ¿Qué expresión denota una lista con 10000 elementos •?
- Necesitamos una forma de generar listas programáticamente, acumulando los valores en la lista

Concatenación de listas: motivación

- ¿Qué expresión denota una lista con 3 elementos •?
- ¿Qué expresión denota una lista con 10 elementos •?
- ¿Qué expresión denota una lista con 10000 elementos •?
- Necesitamos una forma de generar listas programáticamente, acumulando los valores en la lista

```
//Propósito: denota una lista con n repeticiones del valor v
function repeticion(n,v) {
1. lista := []
2. repeat(n)
3. {
4.       lista := ‘‘Agregar v al final de lista’’
5. }
6. return(lista)
}
```

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,
- `ms` denota $[w_1, \ldots, w_m]$,

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,
- `ms` denota $[w_1, \ldots, w_m]$,
- y v_1 y w_1 tienen el mismo tipo (o las listas son vacías), entonces

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,
- `ms` denota $[w_1, \ldots, w_m]$,
- y v_1 y w_1 tienen el mismo tipo (o las listas son vacías), entonces

`ls ++ ms` denota la lista $[v_1, \ldots, v_n, w_1, \ldots, w_m]$.

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,
- `ms` denota $[w_1, \ldots, w_m]$,
- y v_1 y w_1 tienen el mismo tipo (o las listas son vacías), entonces

`ls ++ ms` denota la lista $[v_1, \ldots, v_n, w_1, \ldots, w_m]$.

- En otras palabras, `ls ++ ms` denota la lista que se obtiene de poner `ms` atrás de `ls`

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,
- `ms` denota $[w_1, \ldots, w_m]$,
- y v_1 y w_1 tienen el mismo tipo (o las listas son vacías), entonces

`ls ++ ms` denota la lista $[v_1, \ldots, v_n, w_1, \ldots, w_m]$.

- En otras palabras, `ls ++ ms` denota la lista que se obtiene de poner `ms` atrás de `ls`

- `[1, 2, 3] ++ [4, 5, 6]` denota **[1, 2, 3, 4, 5, 6]**

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,
- `ms` denota $[w_1, \ldots, w_m]$,
- y v_1 y w_1 tienen el mismo tipo (o las listas son vacías), entonces

`ls ++ ms` denota la lista $[v_1, \ldots, v_n, w_1, \ldots, w_m]$.

- En otras palabras, `ls ++ ms` denota la lista que se obtiene de poner `ms` atrás de `ls`

- `[1, 2, 3] ++ [4, 5, 6]` denota **[1, 2, 3, 4, 5, 6]**
- `[Este..maxDir()] ++ [Norte]` denota la lista $[\rightarrow, \downarrow, \leftarrow, \uparrow]$

Concatenación de listas: definición

función de concatenación (++)

- Si `ls` denota $[v_1, \ldots, v_n]$,
- `ms` denota $[w_1, \ldots, w_m]$,
- y v_1 y w_1 tienen el mismo tipo (o las listas son vacías), entonces

`ls ++ ms` denota la lista $[v_1, \ldots, v_n, w_1, \ldots, w_m]$.

- En otras palabras, `ls ++ ms` denota la lista que se obtiene de poner `ms` atrás de `ls`

- `[1, 2, 3] ++ [4, 5, 6]` denota **[1, 2, 3, 4, 5, 6]**
- `[Este..maxDir()] ++ [Norte]` denota la lista $[\rightarrow, \downarrow, \leftarrow, \uparrow]$
- `[] ++[1]` denota `[1]`

Concatenación: agregar adelante y agregar atrás

- Si `ls` es una lista y `e` denota un valor, entonces
  - `ls ++ [e]` denota la lista de agregar `e` atrás de `ls`
  - `[e] ++ ls` denota la lista de agregar `e` adelante de `ls`

- Notar que `ls ++ e` tiene un error de tipos
  - `e` no denota una lista

Concatenación de listas: uso

- Escribir una función `repeticion(n, v)` que retorne una lista con `n` repeticiones de `v`

Concatenación de listas: uso

- Escribir una función `repeticion(n, v)` que retorne una lista con `n` repeticiones de `v`

```
//Propósito: denota una lista con n repeticiones del valor v
function repeticion(n,v) {
 1. acumulador := []
 2. repeat(n)
 3. {
 4.       acumulador := acumulador ++ [v]
          //acumulador := acumulador ++ v → error de tipos
 5. }
 6. return(acumulador)
}
```

Ejercicio II

- Escribir una función agregarSiHayColor que, dada una lista de colores `ls` y un color `c`, devuelva la lista que se obtiene de agregar `c` al final de `ls` si en la celda actual hay bolitas de color `c`.

Ejercicio II

- Escribir una función agregarSiHayColor que, dada una lista de colores `ls` y un color `c`, devuelva la lista que se obtiene de agregar `c` al final de `ls` si en la celda actual hay bolitas de color `c`.

```
function agregarSiHayColor(t, ls, c) {
1. res := ls
2. if(hayBolitas(t, c))
3. {
4.       res := ls ++ [c]
5. }
6. return(res)
}
```

Ejercicio II

- Escribir una función agregarSiHayColor que, dada una lista de colores `ls` y un color `c`, devuelva la lista que se obtiene de agregar `c` al final de `ls` si en la celda actual hay bolitas de color `c`.

```
function agregarSiHayColor(t, ls, c) {
1. res := ls
2. if(hayBolitas(t, c))
3. {
4.       res := ls ++ [c]
5. }
6. return(res)
}
```

- ¿Por qué es necesaria la variable res?
- ¿Qué tiene de malo el nombre de la función?

Ejercicio III

- Escribir una función colores que denote la lista de colores que "aparecen" en las celda actual.

Ejercicio III

- Escribir una función colores que denote la lista de colores que "aparecen" en las celda actual.

```
function colores(t) {
1. res := []
2. foreach c in [minColor()..maxColor()]
3. {
4.       res := agregarSiHayColor(t, res, c)
5. }
6. return(res)
}
```

Ejercicio IV

- Escribir una función listaBolitas que, dado un color `c`, denote la lista que en la *n*-ésima posición tenga la cantidad de bolitas de color `c` que aparece en la `n`-ésima celda de un recorrido del tablero (elegir cualquier forma de recorrer el tablero).

Ejercicio IV

- Escribir una función listaBolitas que, dado un color c, denote la lista que en la *n*-ésima posición tenga la cantidad de bolitas de color c que aparece en la n-ésima celda de un recorrido del tablero (elegir cualquier forma de recorrer el tablero).

```
function listaBolitas(t, c) {
1. res := []
2. t.IrAInicioT(Norte, Este)
3. while(puedeMoverT(t, Norte, Este))
4. {
5.       res := res ++ [nroBolitas(t, c)]
6.       t.MoverT(Norte, Este);
7. }
8. return(res ++ nroBolitas(t, c))
}
```

Recorrido de listas con foreach

Repetición indexada (foreach)

foreach ⟨indice⟩ **in** ⟨lista⟩
    ⟨bloque a repetir⟩

donde,

- ⟨indice⟩ es un identificador
- ⟨lista⟩ denota una lista

- Funciona con rangos porque, en realidad, los rangos son listas!

Recorrido de listas con foreach

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento de la lista

Recorrido de listas con foreach

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento de la lista
- Esta ejecución es en orden

Recorrido de listas con foreach

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento de la lista
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento de la lista

Recorrido de listas con foreach

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento de la lista
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento de la lista
  - Segunda repetición: corresponde al segundo elemento de la lista

Recorrido de listas con foreach

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento de la lista
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento de la lista
  - Segunda repetición: corresponde al segundo elemento de la lista
  - *n*-ésima repetición: corresponde al *n*-ésimo elemento de la lista

Recorrido de listas con foreach

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento de la lista
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento de la lista
  - Segunda repetición: corresponde al segundo elemento de la lista
  - *n*-ésima repetición: corresponde al *n*-ésimo elemento de la lista
  - Última repetición: corresponde al último elemento de la lista

Recorrido de listas con foreach

- **foreach** ejecuta ⟨bloque a repetir⟩ una vez por cada elemento de la lista
- Esta ejecución es en orden
  - Primer repetición: corresponde al primer elemento de la lista
  - Segunda repetición: corresponde al segundo elemento de la lista
  - *n*-ésima repetición: corresponde al *n*-ésimo elemento de la lista
  - Última repetición: corresponde al último elemento de la lista
- Índice: denota el valor correspondiente de la lista
  - Escribiendo su identificador se accede al valor denotado

Ejercicio I

- Escribir un procedimiento RenderCamino que, dada una lista de direcciones y un color, recorra cada una de las direcciones desde la celda actual, poniendo una bolita del color en cada celda. ¿Cuál es la precondición del procedimiento?

Ejercicio I

- Escribir un procedimiento RenderCamino que, dada una lista de direcciones y un color, recorra cada una de las direcciones desde la celda actual, poniendo una bolita del color en cada celda. ¿Cuál es la precondición del procedimiento?

```
procedure t.RenderCamino(camino, color) {
1. foreach dir in camino
2. {
3.       t.Poner(color);
4.       t.Mover(dir);
5. }
}
```

Ejercicio II

- Escribir un procedimiento RenderColores que, dada una lista de colores, recorra cada las celdas del tablero poniendo en la posición *n* una bolita del color que indique la posición *n* de la lista
  Suponer: la longitud de la lista es menor al tamaño del tablero

Ejercicio II

- Escribir un procedimiento RenderColores que, dada una lista de colores, recorra cada las celdas del tablero poniendo en la posición *n* una bolita del color que indique la posición *n* de la lista
  Suponer: la longitud de la lista es menor al tamaño del tablero

```
procedure t.RenderColores(colores) {
   //Recorremos el tablero y la lista en paralelo
1. t.IrAInicioT(Norte, Este)
2. foreach color in colores
3. {
4.       t.Poner(color)
5.       t.MoverT(Norte, Este);
6. }
}
```

- ¿Cómo hacemos si la longitud de la lista es mayor o igual al tamaño del tablero?

Ejercicio III

- Escribir una función `incremento(ns, inc)` que, dada una lista de números `ns` y un número `n`, denote la lista que se obtiene de incrementar en `inc` cada elemento de la lista
  Ejemplo: `incremento([1, 0, 2, 1], 3)` $\longrightarrow$ `[`**4**, **3**, **5**, **4**`]`

Ejercicio III

- Escribir una función `incremento(ns, inc)` que, dada una lista de números `ns` y un número `n`, denote la lista que se obtiene de incrementar en `inc` cada elemento de la lista
  Ejemplo: `incremento([1, 0, 2, 1], 3)` $\longrightarrow$ **[4, 3, 5, 4]**

```
function incremento(ns, inc) {
1. res := []
2. foreach n in ns
3. {
4.       res := res ++ [n + inc]
5. }
6. return(res)
}
```

Ejercicio IV

- Escribir una función `sinElemento(ls, elem)` que, dada una lista `ls` y un elemento `elem`, denote la lista que se obtiene de sacar todas las apariciones de `elem` en `ls`
  Ejemplo: `sinElemento([1, 0, 2, 1], 1)` $\longrightarrow$ `[0, 2]`

Ejercicio IV

- Escribir una función `sinElemento(ls, elem)` que, dada una lista `ls` y un elemento `elem`, denote la lista que se obtiene de sacar todas las apariciones de `elem` en `ls`
  Ejemplo: `sinElemento([1, 0, 2, 1], 1)` $\longrightarrow$ `[0, 2]`

```
function sinElemento(ls, elem) {
1. res := []
2. foreach x in ls
3. {
4.       if(x /= elem)
5.       {
6.             res := res ++ [x]
7.       }
8. }
9. return(res)
}
```

Ejercicio V

- Escribir un procedimiento `t.PonerEscaleraDescenciente(n, c)` que, dado un número `n` y un color `c`, recorra el tablero y ponga `n` bolitas de color `c` en la primer celda, `n - 1` en la segunda, y asi siguiendo hasta que pone `1` bolita de color `c` en la celda `n`.

Ejercicio V

- Escribir un procedimiento `t.PonerEscaleraDescenciente(n, c)` que, dado un número `n` y un color `c`, recorra el tablero y ponga `n` bolitas de color `c` en la primer celda, `n - 1` en la segunda, y asi siguiendo hasta que pone 1 bolita de color `c` en la celda `n`.

```
procedure t.PonerEscaleraDescenciente(n, c) {
1. t.IrAInicioT(Norte, Este)
2. t.PonerN(n, c)
3. foreach i in reverso([1..n])
4. {
5.       t.MoverT(Norte, Este)
6.       t.PonerN(i, c)
7. }
}
```

Ejercicio V

- Escribir un procedimiento `t.PonerEscaleraDescenciente(n, c)` que, dado un número `n` y un color `c`, recorra el tablero y ponga `n` bolitas de color `c` en la primer celda, `n − 1` en la segunda, y asi siguiendo hasta que pone 1 bolita de color `c` en la celda `n`.

```
procedure t.PonerEscaleraDescenciente(n, c) {
1. t.IrAInicioT(Norte, Este)
2. t.PonerN(n, c)
3. foreach i in reverso([1..n])
4. {
5.       t.MoverT(Norte, Este)
6.       t.PonerN(i, c)
7. }
}
```

- **foreach** es poderoso, porque la lista a recorrer puede ser resultado de una función.

Acceso a los elementos de una lista

- Funciones para denotar al primer y último elemento.
- Funciones para saber si una lista está vacía.
- Funciones para denotar la lista sin el primer o último elemento.

Acceso a los elementos de una lista

- Funciones para denotar al primer y último elemento.
- Funciones para saber si una lista está vacía.
- Funciones para denotar la lista sin el primer o último elemento.

- Las funciones de lista sólo operan con el primer o último elemento.
- Para procesar elementos intermedios, deshacernos de los primeros/últimos

Funciones de lista

- Funciones de lista.
  - `ls` es una lista que denota $[v_1, \ldots, v_n]$
  - `tail`(ls) e `init`(ls) denotan listas nuevas.

`isNil`(`ls`):	Denota VERDADERO cuando `ls` es `[]`
`head`(`ls`):	Denota el primer elemento de `ls`, i.e., v_1 Precondición: `ls` $/=$ `[]`
`last`(`ls`):	Denota el último elemento de `ls`, i.e, v_n Precondición: `not isNil`(`ls`)
`tail`(`ls`):	Denota la lista que se obtiene de sacar el primer elemento de `ls`, i.e., $[v_2, \ldots, v_n]$. Precondición: `not isNil`(`ls`)
`init`(`ls`):	Denota la lista que se obtiene de sacar el último elemento de `ls`, i.e., $[v_1, \ldots, v_{n-1}]$. Precondición: `not isNil`(`ls`)

- `tail`(ls) e `init`(ls) denotan listas nuevas.

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])` $\longrightarrow$ `Oeste`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])` $\longrightarrow$ `Oeste`
- `last([])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])` $\longrightarrow$ `Oeste`
- `last([])` $\longrightarrow$ BOOM

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])` $\longrightarrow$ `Oeste`
- `last([])` $\longrightarrow$ BOOM
- `tail([2, 3, 4])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])` $\longrightarrow$ `Oeste`
- `last([])` $\longrightarrow$ BOOM
- `tail([2, 3, 4])` $\longrightarrow$ `[3, 4]`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])` $\longrightarrow$ `Oeste`
- `last([])` $\longrightarrow$ BOOM
- `tail([2, 3, 4])` $\longrightarrow$ `[3, 4]`
- `init([2, 3, 4])`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` $\longrightarrow$ Verdadero
- `isNil([2, 4])` $\longrightarrow$ Falso
- `head([2, 3])` $\longrightarrow$ **2**
- `head([])` $\longrightarrow$ BOOM
- `last([Norte, Este, Sur, Oeste])` $\longrightarrow$ `Oeste`
- `last([])` $\longrightarrow$ BOOM
- `tail([2, 3, 4])` $\longrightarrow$ `[3, 4]`
- `init([2, 3, 4])` $\longrightarrow$ `[2, 3]`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` ⟶ Verdadero
- `isNil([2, 4])` ⟶ Falso
- `head([2, 3])` ⟶ **2**
- `head([])` ⟶ BOOM
- `last([Norte, Este, Sur, Oeste])` ⟶ `Oeste`
- `last([])` ⟶ BOOM
- `tail([2, 3, 4])` ⟶ `[3, 4]`
- `init([2, 3, 4])` ⟶ `[2, 3]`
- `head(tail(init([2, 3, 4])))`

Ejercicio

¿Qué denota cada una de las siguientes expresiones?

- `isNil([])` ⟶ Verdadero
- `isNil([2, 4])` ⟶ Falso
- `head([2, 3])` ⟶ **2**
- `head([])` ⟶ BOOM
- `last([Norte, Este, Sur, Oeste])` ⟶ `Oeste`
- `last([])` ⟶ BOOM
- `tail([2, 3, 4])` ⟶ `[3, 4]`
- `init([2, 3, 4])` ⟶ `[2, 3]`
- `head(tail(init([2, 3, 4])))` ⟶ `3`

Funcionando ando I

- Escribir una función `rotacion` que, dada la lista que denota `[`v_1, ..., v_n`]` denote la lista `[`v_2, ..., v_n, v_1`]`. En otras palabras, la lista que se obtiene de pasar el primer elemento hacia el final. ¿Cuál es la precondición de la función?

Funcionando ando I

- Escribir una función `rotacion` que, dada la lista que denota $[v_1, \ldots, v_n]$ denote la lista $[v_2, \ldots, v_n, v_1]$. En otras palabras, la lista que se obtiene de pasar el primer elemento hacia el final. ¿Cuál es la precondición de la función?

```
function rotacion(lista) {
1. res := []
2. if(not isNil(lista))
3. {
4.       res := tail(lista) ++ [head(lista)];
5. }
6. return(res)
}
```

Funcionando ando II (en casa)

- Escribir una función rotacionN que, dada la lista y un numero n, rote n veces la lista como en la función anterior.

Funcionando ando III

- Escribir una función `sublistaHasta(ls, elem)` que, dada una lista `ls` y un valor `elem`, denote la sublista de `ls` desde el primer elemento hasta la primer aparición de `elem`
  Ejemplo: `sublistaHasta([1, 0, 2, 1], 2)` $\longrightarrow$ `[`**1**, **0**`]`
  Ejemplo: `sublistaHasta([1, 0, 2, 1], 4)` $\longrightarrow$ BOOM

Funcionando ando III

- Escribir una función `sublistaHasta(ls, elem)` que, dada una lista `ls` y un valor `elem`, denote la sublista de `ls` desde el primer elemento hasta la primer aparición de `elem`
  Ejemplo: `sublistaHasta([1, 0, 2, 1], 2)` $\longrightarrow$ **[1, 0]**
  Ejemplo: `sublistaHasta([1, 0, 2, 1], 4)` $\longrightarrow$ BOOM

```
function sublistaHasta(ls, elem) {
1. rec_ls := ls
2. sublista := []
3. while(not head(rec_ls /= elem)
4. {
5.       sublista := sublista ++ [head(rec_ls)]
6.       rec_ls := tail(rec_ls);
7. }
8. return(sublista)
}
```

Funcionando ando III

- Escribir una función `sublistaHasta(ls, elem)` que, dada una lista `ls` y un valor `elem`, denote la sublista de `ls` desde el primer elemento hasta la primer aparición de `elem`
  Ejemplo: `sublistaHasta([1, 0, 2, 1], 2)` $\longrightarrow$ **[1, 0]**
  Ejemplo: `sublistaHasta([1, 0, 2, 1], 4)` $\longrightarrow$ BOOM

```
function sublistaHasta(ls, elem) {
1. rec_ls := ls
2. sublista := []
3. while(not head(rec_ls /= elem)
4. {
5.       sublista := sublista ++ [head(rec_ls)]
6.       rec_ls := tail(rec_ls);
7. }
8. return(sublista)
}
```

- Usar **foreach** no es conveniente, porque no queremos recorrer todos los elementos.

Recorrido de listas con while

- Esquema básico de recorrido para listas con **while**
- Procesa de primero a ultimo.

```
esquema ⟨Recorrido(lista)⟩ {
1. rec_lista := lista
2. while(not isNil(rec_lista))
3. {
4.       ⟨Procesar⟩(head(rec_lista))
5.       rec_lista := tail(rec_lista)
6. }
}
```

Recorrido de listas con while

- Esquema básico de búsqueda en listas
- Busca de primero a ultimo

```
esquema ⟨Busqueda(lista)⟩ {
1. rec := lista
2. while(/*not isNil(rec) &&*/ not ⟨elBuscado⟩(head(rec)))
3. {
4.       rec := tail(rec)
5. }
   //return(/*not isNil(rec) &&*/ ⟨elBuscado⟩(head(rec))
}
```

Ejercicio

- Reescribir usando **while** la función `incremento(ns, inc)` que, dada una lista de números `ns` y un número `n`, denote la lista que se obtiene de incrementar en `inc` cada elemento de la lista

Ejercicio

- Reescribir usando **while** la función `incremento(ns, inc)` que, dada una lista de números `ns` y un número `n`, denote la lista que se obtiene de incrementar en `inc` cada elemento de la lista

```
function incremento(ns, inc) {
1. res := []
2. rec_ns := ns
3. while(notisNil(rec_ns))
4. {
5.       res := res ++ [head(rec_ns) + inc]
6.       rec_ns := tail(rec_ns)
7. }
8. return(res)
}
```

foreach vs. while

- **foreach** es más simple porque se encarga del recorrido
- **while** es más flexible porque puedo controlar cómo se recorre

foreach vs. while

- **foreach** es más simple porque se encarga del recorrido
- **while** es más flexible porque puedo controlar cómo se recorre

- Si es necesario recorrer todos los elementos exactamente una vez, conviene **foreach**
- Si se recorre sólo una parte (inicio/fin) de los elementos, **while** es más simple

foreach vs. while

- **foreach** es más simple porque se encarga del recorrido
- **while** es más flexible porque puedo controlar cómo se recorre

- Si es necesario recorrer todos los elementos exactamente una vez, conviene **foreach**
- Si se recorre sólo una parte (inicio/fin) de los elementos, **while** es más simple

- Se pueden combinar los dos esquemas cuando se recorre más de una lista
- La clase que viene se ven muchos ejemplos...

Tipos de listas

- En XGobstones (y Gobstones) todo valor tiene un tipo...

Tipos de listas

- En XGobstones (y Gobstones) todo valor tiene un tipo...
- ...que determina cuándo puede ser argumento de una funcion/procedimiento

Tipos de listas

- En XGobstones (y Gobstones) todo valor tiene un tipo...
- ...que determina cuándo puede ser argumento de una funcion/procedimiento

- Para "concatenar" dos listas, sus elementos deben tener el mismo tipo

Tipos de listas

- En XGobstones (y Gobstones) todo valor tiene un tipo...
- ...que determina cuándo puede ser argumento de una funcion/procedimiento

- Para "concatenar" dos listas, sus elementos deben tener el mismo tipo
- Las listas se distinguen, pues, por el tipo de sus elementos
  - Lista de números: cuando sus elementos son números
  - Lista de booleanos: cuando sus elementos son booleanos
  - etc.

Tipos de listas

- En XGobstones (y Gobstones) todo valor tiene un tipo...
- ...que determina cuándo puede ser argumento de una funcion/procedimiento

- Para "concatenar" dos listas, sus elementos deben tener el mismo tipo
- Las listas se distinguen, pues, por el tipo de sus elementos
  - Lista de números: cuando sus elementos son números
  - Lista de booleanos: cuando sus elementos son booleanos
  - etc.

- Una lista puede a la vez contener listas
  - Las listas son valores, ergo, puedo meterlas en una lista.
  - Tipos: Lista de lista de números, Lista de lista de lista de números, etc

Tipos de listas

- En XGobstones (y Gobstones) todo valor tiene un tipo...
- ...que determina cuándo puede ser argumento de una funcion/procedimiento

- Para "concatenar" dos listas, sus elementos deben tener el mismo tipo
- Las listas se distinguen, pues, por el tipo de sus elementos
  - Lista de números: cuando sus elementos son números
  - Lista de booleanos: cuando sus elementos son booleanos
  - etc.

- Una lista puede a la vez contener listas
  - Las listas son valores, ergo, puedo meterlas en una lista.
  - Tipos: Lista de lista de números, Lista de lista de lista de números, etc
- XGobstones tiene infinitos tipos

Tipos de listas: lista vacía

- ¿Cuál es el tipo de una lista sin elementos?

Tipos de listas: lista vacía

- ¿Cuál es el tipo de una lista sin elementos?
- Tiene que poder "concatenarse" con listas de cualquier tipo

Tipos de listas: lista vacía

- ¿Cuál es el tipo de una lista sin elementos?
- Tiene que poder "concatenarse" con listas de cualquier tipo
- Entonces, `[]` tiene infinitos tipos, ya que tiene tipo...

Tipos de listas: lista vacía

- ¿Cuál es el tipo de una lista sin elementos?
- Tiene que poder "concatenarse" con listas de cualquier tipo
- Entonces, [] tiene infinitos tipos, ya que tiene tipo...
- ...lista de T para cualquier tipo T
  - [] tiene tipo lista de número,
  - [] tiene tipo lista de direcciones,
  - [] tiene tipo lista de lista de números,
  - [] etc.

Tipos de listas: lista vacía

- ¿Cuál es el tipo de una lista sin elementos?
- Tiene que poder "concatenarse" con listas de cualquier tipo
- Entonces, `[]` tiene infinitos tipos, ya que tiene tipo...
- ...lista de T para cualquier tipo T
  - `[]` tiene tipo lista de número,
  - `[]` tiene tipo lista de direcciones,
  - `[]` tiene tipo lista de lista de números,
  - `[]` etc.
- Cuando se usa `[]`, el valor asume el tipo que le conviene

Tipos de listas: lista vacía

- ¿Cuál es el tipo de una lista sin elementos?
- Tiene que poder "concatenarse" con listas de cualquier tipo
- Entonces, [] tiene infinitos tipos, ya que tiene tipo...
- ...lista de T para cualquier tipo T
  - [] tiene tipo lista de número,
  - [] tiene tipo lista de direcciones,
  - [] tiene tipo lista de lista de números,
  - [] etc.
- Cuando se usa [], el valor asume el tipo que le conviene

- Decimos que [] tiene tipo lista de $*$

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]` $\longrightarrow$ lista de lista de números

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]` $\longrightarrow$ lista de lista de números
- `[[], [], []]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]` $\longrightarrow$ lista de lista de números
- `[[], [], []]` $\longrightarrow$ lista de lista de $*$

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]` $\longrightarrow$ lista de lista de números
- `[[], [], []]` $\longrightarrow$ lista de lista de $*$
- `[[[1]], [], []]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]` $\longrightarrow$ lista de lista de números
- `[[], [], []]` $\longrightarrow$ lista de lista de $*$
- `[[[1]], [], []]` $\longrightarrow$ lista de lista de lista de número

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]` $\longrightarrow$ lista de lista de números
- `[[], [], []]` $\longrightarrow$ lista de lista de $*$
- `[[[1]], [], []]` $\longrightarrow$ lista de lista de lista de número
- `[[[]], [], []]`

Ejercicio de tipos

- ¿Qué tipo tienen las siguientes expresiones que denotan listas?

- `[1, 2]` $\longrightarrow$ lista de números
- `[hayBolitas(Rojo)]` $\longrightarrow$ lista de booleanos
- `[]` $\longrightarrow$ lista de $*$
- `[[1, 2], [0]]` $\longrightarrow$ lista de lista de números
- `[[1, 2], []]` $\longrightarrow$ lista de lista de números
- `[[], [], []]` $\longrightarrow$ lista de lista de $*$
- `[[[1]], [], []]` $\longrightarrow$ lista de lista de lista de número
- `[[[]], [], []]` $\longrightarrow$ lista de lista de lista de $*$

Clase que viene

Vamos a hacer ejercicios que incluyan:

- Funciones de listas sin utilizar tableros.
- Procesamientos usuales sobre listas:
  - búsqueda, filtro, transformación, mezcla.
- Algún procesamiento inusual sobre listas.
- Pizca de Listas de Listas, quizá

Hoy va a estar la nueva práctica