Laboratorio de Computación III

Clase XV

Andrés Fortier

Esta semana

- Vamos a ver un poco en detalle el concepto de bloques y un par de mensajes nuevos de colecciones.
- Aprovechen para avanzar con la práctica.

Bloques

- Contienen una secuencia de expresiones que se pueden ejecutar.
- Se escriben enmarcando dichas expresiones entre corchetes.
- Ejemplos:
  - [1+2].
  - []. *Bloque sin expresiones.*
  - [| sum|

    sum:= 1 + 5.

    sum ** 2

    ].

Bloques

- Los bloques pueden tomar parámetros.
- Ejemplos:
  - [:x | x + 1].
  - [:x :y :z | (x + y) * z].

Bloques

- Para evaluar un bloque se utiliza el mensaje #value o alguna de sus variantes.
- Ejemplos:
  - [1+5] value. *“printIt => 6”*
  - [:x | x + 1] value: 7. *“printIt => 8”*

Bloques

- Los bloques retornan el resultado de la última expresión evaluada.
- Ejemplos:
  - [] value. *“printIt => nil”*
  - [4] value. *“printIt => 4”*
  - [:x |

    x + 1.

    3 + 3.

    ] value: 99. *“printIt => 6”*

Bloques

- Los bloques son objetos y pueden evaluarse muchas veces.
- Ejemplos:
  - | bloque |

    bloque := [1+1].

    bloque value. *“printIt => 2”*

    bloque value. *“printIt => 2”*
  - | bloque |

    bloque := [:x | x+1].

    bloque value: 3. *“printIt => 4”*

    bloque value: 18. *“printIt => 19”*

Bloques en acción

```
CuentaBancaria>>extraer: unMonto

    (self puedeExtraer: unMonto)
        ifTrue:[self realizarExtracción: unMonto].
```

```
True>>ifTrue: aBlock

    ^aBlock value.
```

- Supongamos que se puede hacer la extracción.

```
    (self puedeExtraer: unMonto)
        ifTrue:[self realizarExtracción: unMonto].
```

~>

```
    true
        ifTrue:[self realizarExtracción: unMonto].
```

~>

```
    [self realizarExtracción: unMonto] value.
```

~>

```
    self realizarExtracción: unMonto.
```

Iteradores

```
SequenceableCollection>>do: aBlock

    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

```
#(1 3 5) do: [:numero | Transcript show: numero]
```

```
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

~>

```
    1 to: 3 do:
        [:index | aBlock value: (self at: index)]
```

~>

```
    aBlock value: (self at: 1).
    aBlock value: (self at: 2).
    aBlock value: (self at: 3).
```

~>

```
    aBlock value: 1.
    aBlock value: 3.
    aBlock value: 5.
```

Iteradores

```
SequenceableCollection>>do: aBlock

    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

```
#(1 3 5) do: [:numero | Transcript show: numero]
```

```
aBlock value: 1.
aBlock value: 3.
aBlock value: 5.
```

~>

```
[:numero | Transcript show: numero] value: 1.
[:numero | Transcript show: numero] value: 3.
[:numero | Transcript show: numero] value: 5.
```

~>

```
Transcript show: 1.
Transcript show: 3.
Transcript show: 5.
```

Bloques

- Los bloques no son algo único de Smalltalk
  - *Closures* o funciones anónimas.
- Smalltalk

```
#(1 2 3) collect: [:n | n + 1]
```

- Ruby

```
[1,2,3].map {|n| n + 1}
```

- Javascript

```
[1,2,3].map(function(n) {return n + 1})
```

Bloques

- Smalltalk

```
| block |
block := [:nombre | 'Hola ' , nombre].
block value: 'Pepe'.
```

'Hola Pepe'

- Ruby

```
block = Proc.new {|nombre| 'Hola ' + nombre}
block.call('Pepe')
```

“Hola Pepe”

- Javascript

```
block = function(nombre) {return "Hola " + nombre}
block("Pepe")
```

“Hola Pepe”