References


Smalltalk Best Practice Patterns, Beck

Smalltalk With Style, Klimas, Skublics, Thomas

Reading

Smalltalk by Example, Alex Sharp,
  Chapter 2 Methods
  Chapter 8 Control Structures
if

(boolean expression) ifTrue: trueBlock

(boolean expression) ifFalse: falseBlock

(boolean expression) ifFalse: falseBlock ifTrue: trueBlock

(boolean expression) ifTrue: trueBlock ifFalse: falseBlock

a < 1 ifTrue: [Transcript show: 'hi mom' ]

difference := (x > y)
    ifTrue: [ x - y]
    ifFalse: [ y - x]
## Boolean Expressions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>a</td>
</tr>
<tr>
<td>And</td>
<td>a &amp; b</td>
</tr>
<tr>
<td>Exclusive or</td>
<td>a xor: (b &gt; c)</td>
</tr>
<tr>
<td>Negation</td>
<td>(a&lt; b) not</td>
</tr>
</tbody>
</table>

## Lazy Logical Operations

<table>
<thead>
<tr>
<th>Message</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>a or: [b &gt; c]</td>
</tr>
<tr>
<td>And</td>
<td>a and: [c</td>
</tr>
</tbody>
</table>
This is not C

This is a runtime error

5 ifTrue: [1 + 3]

Of course you could just add the ifTrue: method to the Number class if you want to do the above.
A Style Issue

Both do the same thing

difference := (x > y)
  ifTrue: [ x - y]
  ifFalse: [ y - x]

(x > y)
  ifTrue: [difference := x - y]
  ifFalse: [difference := y - x]

The one on the left may seem strange. Other language do allow this. Some (many Smalltalkers) consider the one on the left to better convey the intent of the code.
isNil

Answers true if receiver is nil otherwise answers false

x isNil
   ifTrue: [ do something]
   ifFalse: [ do something else]

Shortcuts

ifNil:ifNotNil:
ifNotNil:ifNil:
ifNil:
ifNotNil:

x
   ifNil: [ do something]
   ifNotNil: [ do something else]
Blocks

A deferred sequence of actions – a function without a name
Can have 0 or more arguments
Executed when sent the message 'value'

Similar to
  Lisp's Lambda- Expression
  Erlang's funs
  Ruby's Blocks
  Python's lambda
  Anonymous functions

[:variable1 :variable2 ... :variableN |
  | blockTemporary1 blockTemporary2 ... blockTemporaryK |
  expression1.
  expression2.
  ...]
Blocks and Return Values

Blocks return the value of the last executed statement in the block

```
| block x |
block := [:a :b |
    | c |
    c := a + b.
    c + 5].
```
x := block value: 1 value: 2.

x has the value 8
Blocks know their Environment

| a  b |
a := 1.
b := 2.
aBlock := [a + b].
result := aBlock value
result is now 3

| a  b |
a := 1.
b := 2.
aBlock := [a + b].
a := 5
result := aBlock value
result is now 6
Blocks and Arguments

Using the `value:` keyword message up to 4 arguments can be sent to a block.

\[
[2 + 3 + 4 + 5] \text{ value}
\]
\[
[:x | x + 3 + 4 + 5 ] \text{ value: 2}
\]
\[
[ :x :y | x + y + 4 + 5] \text{ value: 2 value: 3}
\]
\[
[ :x :y :z | x + y + z + 5] \text{ value: 2 value: 3 value: 4}
\]
\[
[ :x :y :z :w | x + y + z + w] \text{ value: 2 value: 3 value: 4 value: 5}
\]

valueWithArguments: can be used with 1 or more arguments

\[
[ :a :b :c :d :e | a + b + c + d + e ] \text{ valueWithArguments: #( 1 2 3 4 5)}
\]
\[
[ :a :b | a + b ] \text{ valueWithArguments: #( 1 2 )}
\]
Where is the Value Message

difference := (x > y)
  ifTrue: [ x - y]
  ifFalse: [ y - x]

In the False class we have:

  ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock
    ^falseAlternativeBlock value

In the True class we have:

  ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock
    ^trueAlternativeBlock value

This is an example of Polymorphism. More on this later.
While Loop

aBlockTest whileTrue
aBlockTest whileTrue: aBlockBody
aBlockTest whileFalse
aBlockTest whileFalse: aBlockBody

The last expression in aBlockTest must evaluate to a boolean

|x y difference |
| count |

x := 8.
count := 0.
y := 6.
count := 0.
difference := 0.
[x > y] whileTrue:
  [difference := difference + 1.
   y := y + 1].
^difference

[count := count + 1.
  count < 100] whileTrue.
Transcript
  clear;
  show: count printString
More Loops

Transcript
   clear.
3 timesRepeat:
   [Transcript
      cr;
      show: 'Testing!'].
1 to: 3 do:
   [ :n |
      Transcript
      cr;
      show: n printString;
      tab;
      show: n squared printString].
9 to: 1 by: -2 do:
   [ :n |
      Transcript
      cr;
      show: n printString].