5. Blocks
6. Messages

What we do not see included in this list is perhaps more interesting than what is included: we do not see any elements for expressing control flow, no conditionals or loops. They are not needed, as they are implemented in terms of messages, objects, and blocks (which are objects). The following method implements the factorial function in class `Integer`:

```
factorial
    "Answer the factorial of the receiver."

    self = 0 ifTrue: [^ 1].
    self > 0 ifTrue: [^ self * (self - 1) factorial].
    self error: 'Not valid for negative integers'
```

Operators `=` and `>` are message selectors returning objects of abstract class `Boolean`, which has two subclasses, `True` and `False`. If the receiver of selector `ifTrue:` is an instance of `True`, then its argument is executed. The argument is the block delimited by `[ ]`. There is a symmetrical selector `ifFalse:` with the opposite semantics. In general it is a good idea to use loops instead of recursion, so here is a loop implementation of the factorial function:

```
factorial
    "Implement factorial function using a loop"

    | returnVal |
    returnVal := 1.
    self >= 0
          ifTrue: [2
                    to: self
                    do: [:n | returnVal := returnVal * n]]
          ifFalse: [self error: 'Not valid for negative integers'].
    ^ returnVal
```

The bulk of the job is carried out inside two blocks. The first block is executed for positive values starting at 2 until the value of the receiver. Each iterated value is passed to the inner block, where we calculate the result. Block arguments are prefixed by a colon (`:`) and separated from the body of the block with a vertical bar (`|`). There may be more than one block argument, as in the following definition of factorial (Black et al. 2007, p. 212):

```
factorial := [:n | (1 to: n) inject: 1 into: [:product :each | product * each ] ].
```

The `to:` selector returns an instance of class `Interval`, which effectively enumerates the values from 1 to 10. For a number `n`, the `factorial` block will do the following. First, it will set the `product` argument of the inner block to 1. Then, it will call the inner block for each value from 1 to n, calculating the product of each iterated number and the current `product`, and storing the result to `product`. To evaluate the factorial of 10, we need to write `factorial value: 10`. To borrow Herbert Simon's quotation of early Dutch physicist Simon Stevin in *The Sciences of the Artificial* (1996):

> *Wonder, en is gheen wonder*. That is to say: "Wonderful, but not incomprehensible."