Chapter 1

Syntax in a nutshell

Pharo のシンタックスは他の近代的な Smalltalk 実装と同様， Smalltalk-80 にとてもよく似ています．そして，プログラムをピジン英語のように声に出して読めるよう シンタックス が設計されています.

```
(Smalltalk includes: Class) ifTrue: [ Transcript show: Class superclass ]
```

Pharo のシンタックスはとてもシンプルです．本質的には，メッセージ(*i.e.,* 式)を送信するためにのみシンタックスが存在します．式は少数の基本要素を使って構築します. キーワードは6つだけしかありません.さらに，制御構造のためのシンタックスも，新しいクラスを宣言するためのシンタックスもありません. 代わりに，オブジェクトにメッセージを送信することにより，ほとんどのことが達成できるのです. 例えば，if-then-else制御構造の代わりに， Smalltalk は Boolean オブジェクトに ifTrue: のようなメッセージを送ります. また，新しい(サブ)クラスを作成するときには作成したいクラスのスーパークラスにメッセージを送ります.

1.1 Syntactic elements

式は以下の要素からできています．(i) 6つの予約語もしくは擬似変数: self, super, nil, true, false, and thisContext, (ii) 数，文字，文字列，シンボルおよび配列を含む リテラルオブジェクト 用定数式，(iii) 変数宣言，(iv) 代入，(v) ブロック クロージャー(vi) メッセージ

Table 1.1 は様々なシンタックスの例です.

ローカル変数 startPoint は変数名もしくは識別子です．慣習により，識別子は“キャメルケース” (*i.e.,* 2番目以降の単語の先頭を大文字にし，単語を連結したもの)で記述します．インスタンス変数，メソッドとブロックの引

シンタックス	意味
startPoint	変数名
Transcript	グローバル変数名
self	擬似変数
1	10進整数
2r101	2進整数
1.5	浮動小数点数
2.4e7	指数表現
$a	文字 'a'
'Hello'	文字列 "Hello"
#Hello	シンボル #Hello
#(1 2 3)	リテラル配列
{1. 2. 1+2}	動的配列
"a comment"	コメント
| x y |	変数 x と y の宣言
x := 1	x に 1 を代入
[x + y]	x+y を評価するブロック
<primitive: 1>	バーチャルマシン・プリミティブもしくはアノテーシ
3 factorial	単項メッセージ
3+4	二項メッセージ
2 raisedTo: 6 modulo: 10	キーワードメッセージ
↑ true	値 true を返す
Transcript show: 'hello'. Transcript cr	式セパレーター (.)
Transcript show: 'hello'; cr	メッセージカスケード (;)

Table 1.1: Pharo Syntax in a Nutshell

数，一時変数の先頭の文字は小文字にします. つまり，プライベートなスコープを持つ変数の先頭を小文字にするのです.

共有変数 の先頭は大文字にします．グローバル変数， クラス変数， プール変数， ディクショナリ，もしくはクラス名を表します．Transcript はグローバル変数であり，クラス TranscriptStream のインスタンスです.

The receiver. self is a keyword that refers to the object inside which the current method is executing. We call it "the receiver" because this object will normally have received the message that caused the method to execute. self is called a "pseudo-variable" since we cannot assign to it.

Integers. In addition to ordinary decimal integers like 42, Pharo also provides a radix notation. 2r101 is 101 in radix 2 (*i.e.,* binary), which is equal to decimal 5.

Floating point numbers can be specified with their base-ten exponent: 2.4e7 is 2.4×10^7.

Characters. A dollar sign introduces a literal character: $a is the literal for 'a'. Instances of non-printing characters can be obtained by sending appropriately named messages to the Character class, such as Character space and Character tab.

Strings. Single quotes are used to define a literal string. If you want a string with a quote inside, just double the quote, as in 'G"day'.

Symbols are like Strings, in that they contain a sequence of characters. However, unlike a string, a literal symbol is guaranteed to be globally unique. There is only one Symbol object #Hello but there may be multiple String objects with the value 'Hello'.

Compile-time arrays are defined by #( ), surrounding space-separated literals. Everything within the parentheses must be a compile-time constant. For example, #(27 (true false) abc) is a literal array of three elements: the integer 27, the compile-time array containing the two booleans, and the symbol #abc. (Note that this is the same as #(27 #(true false) #abc).)

Run-time arrays. Curly braces { } define a (dynamic) array at run-time. Elements are expressions separated by periods. So { 1. 2. 1+2 } defines an array with elements 1, 2, and the result of evaluating 1+2. (The curly-brace notation is peculiar to the Pharo and Squeak dialects of Smalltalk! In other Smalltalks you must build up dynamic arrays explicitly.)

Comments are enclosed in double quotes. *"hello"* is a comment, not a string, and is ignored by the Pharo compiler. Comments may span multiple lines.

Local variable definitions. Vertical bars | | enclose the declaration of one or more local variables in a method (and also in a block).

Assignment. := assigns an object to a variable.

Blocks. Square brackets [ ] define a block, also known as a block closure or a lexical closure, which is a first-class object representing a function. As we shall see, blocks may take arguments and can have local variables.

Primitives. <primitive: ...> denotes an invocation of a virtual machine primitive. (<primitive: 1> is the VM primitive for SmallInteger»+.) Any code following the primitive is executed only if the primitive fails. The same syntax is also used for method annotations.

Unary messages consist of a single word (like factorial) sent to a receiver (like 3).