

Lang Syntax Reference

Randy Henry

`pivotsallit@gmail.com`

April 24, 2020

Contents

1	Introduction	2
2	Miscellaneous	2
2.1	Commenting	2
2.2	Documentation comment	2
2.3	Tokens	2
2.4	Variable assignment/declaration	3
2.5	Grouping	3
2.6	Comparisons	4
3	Functions	4
3.1	Function calls	4
3.2	Function definitions	5
3.3	Composition	5
4	Control Flow	5
4.1	Sequencing	5
4.2	If ... then	6
4.3	Pattern matching	6
4.4	Looping	6
5	Types	7
6	Classes / Traits	7
7	Modules	7

1 Introduction

This paper walks through the "Syntax Across Languages" [Pixel, 2008] page, describing how each phenomenon it references would be translated into valid LANG syntax.

This is meant not as an introductory guide to LANG, but as a sort of formal reference for how common design patterns would appear in LANG code out in the wild.

2 Miscellaneous

2.1 Commenting

Until end of line	<code>-- This is a comment.</code>
-------------------	------------------------------------

Nestable	<code>--/ This is a comment. /--</code>
----------	---

2.2 Documentation comment

Until end of line	<code>--/ This is a comment.</code>
-------------------	-------------------------------------

Nestable	<code>--// This is a comment. //--</code>
----------	---

2.3 Tokens

Case-sensitive	<code>x != X.</code>
----------------	----------------------

kebab-case-variables	<code>avogadro's-number := 6.022e23.</code>
----------------------	---

Upper-Kebab-Types	<code>type 2D-Point := { x: Int, y: Int. }.</code>
-------------------	--

2.4 Variable assignment/declaration

	<pre>-- separate type annotation a: Nat. a := 3.</pre>
Declaration	<pre>-- together b: Int := -4. -- inferred τ := 2 * π.</pre>
Assignment	<pre>x := 3. -- mutable x := x + 2.</pre>
Scoped Declaration	<pre>let x := expr in { ... }.</pre>

2.5 Grouping

Expressions	<pre>empty? (filter even? xs)</pre>
Statements	<pre>-- explicit (brace style) x := { y := 3, println "if then else expr", if { z := y + 2, z < 6. } then { y. } else { (y * 3) + 2. }. }. -- implicit (indentation style) x := y := 3, println "if then else expr", if z := y + 2, z < 6 then y else (y * 3) + 2.</pre>

2.6 Comparisons

Deep Equality	$\pi = \pi$, $3 \neq 4$, $3 \neq 4$.
Comparison	$x > y$, $y < x$. $a \leq b$, $b \geq a$.
Ordering (inferior, equal, or superior)	<code>compare "abc" "bac". -- LT</code>
Extreme values	<code>min [1, 2, 3], max 1 2 3.</code>

3 Functions

3.1 Function calls

Parametrized	<code>f a b,</code> <code>a.f(b). -- UFCS</code>
No parameters	<code>f.</code>
Partial application (given 1 st)	<code>map filter(even?) nested-list,</code> <code>filter(2 >) xs. -- binary infix</code>
(given 2 nd)	<code>map filter(,xs) [even?, div-by-3?],</code> <code>filter(> 2) xs.</code>

3.2 Function definitions

Typed	<pre>index: [String], Nat -> String, index xs i := { ... }. -- inline index: (xs: [String], i: ℕ) → String := ...</pre>
Inferred	<pre>index xs i := { ... }.</pre>
Anonymous	<pre>\x. { x + x. }, -- braces optional f := λx + x, -- equivalent (λx + λy) 3 4 -- 12</pre>
Var args	<pre>print-double: ...ℕ → IO, print-double ...xs := println (doubles.join " ", "), where doubles := xs.map str-dub, str-dub := String ○ (λx + x).</pre>

3.3 Composition

```
(f ○ g) x,
-- or
(f ○ g) x.
```

4 Control Flow

4.1 Sequencing

```
print x, print (x * 2), print (x × 4).
```

4.2 If ... then ...

No else	<code>pred => println f"{pred} is set".</code>
If ... else ...	<code>if pred then println f"{pred} is set" else println "How could this be???".</code>
Stacked no-elses	<code>println pred => f"{pred} is true", pred2 => f"{pred2}, but not {pred}", "Neither?? How could this be???".</code>
Guards	<code>println pred => "Pred!" pred2 => "The other one!" otherwise => "Neither??"</code>
Pythonic	<code>println "Yup" if pred else "Nope".</code>

4.3 Pattern matching

```
match val  
  v1 => ...,  
  v2 | v3 => ...,  
  _ => ....
```

4.4 Looping

While condition	<code>while c { ... }</code>
Until condition	<code>until c { ... }</code>
For value in range	<code>(1..10).map λi. ... for i in 1..10 { ... }</code>
For value in descending range	<code>(10..1).map λi. ... for i in 10..1 { ... }</code>
For value in custom range	<code>(2,4..10).map λi. ... for i in 2,4..10 { ... }</code>

5 Types

Declaration	<code>type T := ...</code>
Annotation	<code>pi: Real = 3.14</code>
Computed conversion	<code>Int 3.14, String 3.</code>

6 Classes / Traits

Declaration	<code>trait A := ...</code>
Inheritance	<code>impl A for T { ... }.</code>

7 Modules

Declaration	<code>(private public) module m.</code>
Selective export	<code>(private public) f.</code>
Import all	<code>import m.</code>
Selectively	<code>from p import name1, name2.</code>

References

[Pixel, 2008] Pixel (2008). Syntax across languages. <https://rigaux.org>.
Accessed: 2020-04-20.