Lang Syntax Reference

Randy Henry
pivotsallit@gmail.com

April 20, 2020

Contents

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	`-- This is a comment.`
Nestable	`--/ This is a comment. /--`

2.2 Documentation comment

Until end of line	`--| This is a comment.`
Nestable	`--|| This is a comment. ||--`

2.3 Tokens

Case-sensitive	`x =/= X.`
kebab-case-variables	`avogadro's-number := 6.022e23.`
Upper-Kebab-Types	`type 2D-Point := { x: Z, y: Z. }.`

2.4 Variable assignment/declaration

Declaration

```
-- separate type annotation
a: Nat.
a := 3.

-- together
b: Int := -4.

-- inferred
τ := 2 * π.
```

Assignment

```
x :- 3. -- mutable
x :- x + 2.
```

Scoped Declaration

```
let x := expr in { ... }.
```

2.5 Grouping

Expressions

```
empty? (filter even? xs)
```

Statements

```
-- 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.
```

2.6 Comparisons

Deep Equality	`π = π, 3 =/= 4, 3 ≠ 4.`

Comparison	`x > y, y < x.`
	`a <= b, b >= a.`
	`n ≥ m, m ≤ n.`

Ordering (inferior, equal, or superior)	`compare "abc" "bac". -- LT`
Extreme values	`min [1, 2, 3], max 1 2 3.`

3 Functions

3.1 Function calls

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

3.2 Function definitions

Typed

```
index: [String], Nat -> String,
index xs i := { ... }.

-- inline
        index: (xs: [String], i: ℕ) -> String := ...
```

Inferred

```
index xs i := { ... }.
```

Anonymous

```
\x. { x + x. }, -- braces optional
f := λx. λy. x + y,
f := λx y. x + y. -- equivalent
```

Var args

```
print-double: ...Num -> ().
print-double ...xs := {
    println
        (map
            (String ∘ (λx. x + x))
            xs
        ).join ", ".
}.
```

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 ...

References

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