Programming Language Reference

Adrian

January 24, 2014

1 Prelude

Hmmmmm.

2 Ternary Operator

C	a ? b : c
Lisp	if a b c
Haskell	(if a then b else c)
Erlang	case A of true -> B; false -> C end if A == true -> B; true -> C end
Python	b if a else c
Ruby	if a then b elseif d then e else c end (a && b) || c
Scala	if (a) b else c
Perl	a ? b : c

3 List Construction

Lisp	(cons 1 (cons 2 nil))
Scala	1 :: 2 :: Nil
Haskell	1 : 2 : []
Erlang	[1 | [2 | []]]
Ruby	[] << 1 << 2

4 Arrays

- C
- Java
- Scala

5 List API

5.1 Python

```
li.append(x)
li.index(x)
li.insert(i, x)
li.pop(i = -1)
li.remove(x) # void
len(li)
li.reverse()
```

5.2 Ruby

```
a + b # extend
a & b # intersection
a - b # array difference
a | b # union
li.collect { |x| block } # map
li.count
li.count(x) # occurences of x
li.delete(x) # returns x
li.delete_at(i) # returns x
li.delete_if { |item| block } # list of elements deleted
li.each { |x| block }
li.each_index { |i| block }
li.empty?
li.index(x)
li.index { |item| block }
li.drop(n) # returns last length - n elements
li.first
li.first(n) # returns first n elements
li.last(n) # returns last n elements
li.take(n) # first n elements
li.insert(i, obj...)
li.map.with_index { |x, i| block }
li.pop # end
li.push(obj, ...) # end
li.shift # front
li.unshift(obj, ...) front
li.slice
li.sort
li.sort { |a, b| block }
li.zip(arr, ...) # merges elements, creating li.size lists
```

5.3 Javascript

5.4 Java

5.5 C++

5.6 Scala

6 Slicing

Hmmmmm

7 List Comprehensions

Python	[x ** 2 for x in range(10) if x ** 2 > 3]
Ruby	(1..10). select { |x| x ** 2 > 3 }. collect { |X| 2 * x }
Scala	for (x <- 0 until 10 if x * x > 3) yield 2 * x
Erlang	[2 * X || X <- lists:seq(0, 10), X * X > 3]
Haskell	[2 * x | x <- [0..10], x ^ 2 > 3]

8 C++ Templates

Hmmmm

9 C Typedefs

Hmmmm

10 Lambdas

Javascript	function foo(x) { var y = x * 2; return y; }
Scala	(x: Int) => val y = x * 2; /*newline*/ return x;
Ruby	lambda do |x| y = x * 2; return y; end lambda { |x| y = x * 2; return y; }
Haskell	\x -> x * 2
Erlang	fun(Self, args) -> args; (_, X) -> X * 2 end

11 Y-Combinator

```
function Y(le) {
    return (function(f) {
        return f(f);
    })(function(f) {
        return le(function(x) {
            return f(f)(x);
        });
    });
}

var factorial = Y(function(recurse) {
    return function(n) {
        return n == 0 ? 1 : n * recurse(n - 1);
    };
});
```

12 Exceptions

Java, Scala, Python, Ruby, C++, Javascript, PHP

13 Objective-C Blocks

Hmmmm

14 Operator Precedence

Hmmmm

15 Iteration

Java, C++, Python, Scala, Ruby, PHP, Javascript, Erlang, Haskell, Lisp

16 Ranges

Language	Exclusive	Inclusive
Scala	0 until n	0 to n
Ruby	0... n	0.. n
Python	range(0, n)	range(0, n + 1)
Haskell	[0.. n − 1]	[0.. n]
Erlang	lists :seq(0, n − 1)	lists :seq(0, n)

17 Math

17.1 Exponentiation

C	pow(x, y)
Scala	Math.pow(x, y)
Java	Math.pow(x, y)
Javascript	Math.pow(x, y)
Erlang	math:pow(x, y)
Ruby	x ** y
Python	x ** y
Haskell	(^) :: (Num a, Integral b) => a -> b -> a (^^) :: (Fractional a, Integral b) => a -> b -> a (**) :: Floating a => a -> a -> a

17.2 Division

Family	Integer	Decimal	Truncate towards
C	a / b	(double) a / b	
Python	a // b	a / b	
Ruby	a / b	a.to_f / b	
Erlang	A div B floor(A / B)	A / B	
Haskell	quot a b div a b	a / b	
Lisp	(floor (/ a b))	(/ a b)	

17.3 Remainder

Family	Syntax	Same sign as
C	a % b	Dividend
Haskell	rem a b	Dividend
Haskell	mod a b	Divisor
Erlang	a rem b	Dividend
Python	a % b	Divisor
Ruby	a % b modulo(a, b)	Divisor
Ruby	remainder(a, b)	Dividend
Lisp	(modulo a b)	Divisor
Lisp	(remainder a b)	Dividend

18 Haskell Integer Types

Instance	Classes	Description
Int	Num, Real, Integral	
Integer	Num, Real, Integral	
Float	Num, Real, RealFrac, Floating, RealFloat	
Double	Num, Real, RealFrac, Floating, RealFloat	

Class	Extends	Description
Num		
Real	Num	
Fractional	Num	
Integral	Real	
RealFrac	Real, Fractional	

Floating	Fractional
RealFloat	RealFrac, Floating

19 Comments

Hmmmmm

20 Boolean and Logical Operators

Language	And	Or	Not	Type	True	False
Haskell	&&	||	not	Bool	True	False

21 Gotchas

- Quot truncates towards 0, and rem has the same sign as the dividend. Div truncates towards negative infinity, and mod has the same sign as the divisor

22 To learn

- perl
- pascal
- cobol
- fortran
- lua