Rules & guidelines for Scala functions

Scala provides an embarrassing variety of ways to define and call functions. A lot of the variety exists just for the writers of internal DSLs. Here are some of the pieces at our disposal.

Functions with no, 0, or 1 parameter(s)

Rules / Guidelines	Example	Notes
A **parameterless function** defines no parameters (using no parentheses). It must be called with no parentheses. Scala style says that the function should be purely functional (no side effects).	`def one = "The loneliest number"`	
A **zero-parameter function** defines zero parameters (using parentheses). It may be called with or without parentheses. Scala style says that calling the function without parentheses means it is purely functional (no side effects).	`def two() = one + " since the number one"`	
A **one-parameter** function defines one parameter (using parentheses). It must be called using parentheses.	`def ¬(value: Boolean) = !value`	

Variables, values, and functions

Rule / Guideline	Example	Notes
`val` binds a value to a name. The expression is evaluated only once, when the name is defined.	`val test_val = {println("defining test"); 3}`	
`def` defines a function. The body is evaluated every time the function is called.	`def test_def = {println("defining test"); 3}`	
lazy `val` binds a value to a name. The expression is evaluated only once, when the name is first used.	`lazy val test_val = {println("defining test"); 3}`	

By-name parameters

Rule / Guideline	Example	Notes

A **by-name** parameter (defined by placing ⇒ before the type)
 	
evaluates its argument every time the parameter is *used* (but not until the first time it's used).
	`def printThenResult(b: Boolean) = {`
	`  println("You called?")`
	`  b`
	`}`
	
	`def doAndByValue(b: Boolean) = false && b`
	
	`def doAndByName(b: ⇒Boolean) = false && b`	

A one-parameter function can be **called using braces instead of parentheses**. Scala style says we should do so only if it's a by-name parameter.	`doAndByName{true}`	

Method calls

Not part of functional programming, but it's good to know this info.

Rule / Guideline	Example	Notes

Methods can be called using **infix notation** by omitting the dot between the receiver and the method.
 	
Watch out though! This feature doesn't always play well with other features.
	`0 to(10)`
	`0 to (10)`
	`0 to 10`	

Anonymous functions

Rule / Guideline	Example	Notes

There are so many ways to define and call anonymous functions. Scala style generally says to choose the shortest way possible.	`(-5 to 5).filter( (x: Int) => x > 0 )`
	`(-5 to 5).filter( (x) => x > 0 )`
	`(-5 to 5).filter( x => x > 0 )`
	`(-5 to 5).filter( _ > 0 )`
	`(-5 to 5) filter (_ > 0)`	

Partial functions & curried functions

Rule / Guideline	Example	Notes
Partial functions: If we want to use a function as a value, we can't usually refer to its name. We have to use _ to partially apply it. *The exception is when the function is an argument to another function.*	`(-5 to 5).filter` `(-5 to 5).filter _`	
Curried functions: Functions that are meant to be called by chaining function calls.	`def sum3c(x: Int)(y: Int)(z: Int) = x + y + z` `sum3c(10)(20)(30)`	