

Internal DSLs: Control-Flow

Is this a DSL?

If This Then That (IFTTT)

The screenshot shows the IFTTT website interface with a browser window. The address bar displays <https://ifttt.com/recipes/hot>. The page features three tabs: "Recommended" (active), "DO Recipes", and "IF Recipes". Below the tabs is a grid of automation recipes, each with an "if" and "then" section and a description.

Recipe Title	Author	Followers	Likes
Remind yourself to put on sunscreen when the UV Index is high	mayallama	21k	563
Rain tomorrow? Get a notification	alexander	204k	5.0k
Fastest way to save a song	spotify	4.4k	147
Save starred Gmails to Evernote	stong	33k	1.4k
Save screenshots to a separate album	fisjon	61k	1.6k
Keep your profile pictures in sync	derickjackson	108k	3.3k
Keep a call log			
Get an email if there will be rain tomorrow			
Read the day's			

Simple techniques for adding **fluency**

Most general-purpose languages support these features.

names including Unicode	sin(θ) ASK: If the DSL supports Unicode, how will the user write programs?
whitespace	<pre>computer(); processor(); cores(2); disk(); size(150);</pre>
function composition	<pre>computer(processor(cores(2)), disk(size(150)));</pre>
method chaining	<pre>computer() .processor() .cores(2) .disk() .size(150) .end();</pre>

Techniques for **hiding the host language**

These features tend to be language-specific. Some languages support this ability more than others.

(re-)defining operators

```
set1 U set2
```

```
set1 + set2
```

Different host languages gives us different amounts of control over precedence and associativity.

infix operators

```
set1 union set2
```

```
salaries map giveRaise
```

pre- and postfix operators

```
~1
```

```
i++
```

literal extension

```
3 little pigs
```

closures

curried functions,
call-by-name,...

```
test("An empty Set should have size 0") {  
  assert(Set.empty.size == 0)  
}
```

Useful for defining new **control-flow structures**

Implicit conversions

See *Scala for the Impatient*, Chapter 21.4

The compiler looks for an implicit conversion when:

- the expected type differs from the inferred type
- an object does not contain an expected attribute

The compiler finds an implicit conversion when:

- a conversion is declared as `implicit`
- a conversion is in scope and is named with a single identifier
- a conversion is defined in the current class's *companion object*

The compiler does not look for an implicit conversion when:

- the code compiles without one
- the compiler has already performed one (for a given expression)
- it finds multiple conversions (i.e., conversion is ambiguous)

```
import scala.language.implicitConversions
```

```
:implicits [-v]
```

```
-Xprint:typer
```

Implementing new control-flow structures

```
var i = 0
while (i < 10) {
  if ( (i % 2) == 0 )
    println(i)
  i += 1
}
```



```
var i = 0
loop_until (i > 9) {
  if ( (i % 2) == 0 )
    println(i)
  i += 1
}
```

The Scala-y way to do this is:

```
(0 until 10) filter(_ % 2 == 0) map println
```

but we're feeling subversive today.

github.com/hmc-cs111-fall2016/internal-lab

The screenshot shows a GitHub repository page for 'hmc-cs111-fall2016/internal-lab'. The page includes a navigation bar with 'Pull requests', 'Issues', and 'Gist' links. Below the repository name, there are buttons for 'Watch' (1), 'Star' (0), and 'Fork' (0). A secondary navigation bar contains 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The main content area features a summary bar with '1 commit', '1 branch', '0 releases', and '1 contributor'. Below this, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. A commit history table shows a single commit by 'bwiedermann' with files 'project', 'src/main/scala/internal', 'README.md', and 'build.sbt'. The 'README.md' file is expanded, showing the title 'Lab: Subversive Scala' and an 'Overview' section. The overview text describes three puzzles for implementing control-flow structures in Scala and provides a more detailed description with a list of notes.

hmc-cs111-fall2016/internal-lab

https://github.com/hmc-cs111-fall2016/internal-lab

This repository Search Pull requests Issues Gist

hmc-cs111-fall2016 / internal-lab Watch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

No description or website provided. — Edit

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

bwiedermann initial commit Latest commit ba74904 3 minutes ago

project	initial commit	3 minutes ago
src/main/scala/internal	initial commit	3 minutes ago
README.md	initial commit	3 minutes ago
build.sbt	initial commit	3 minutes ago

README.md

Lab: Subversive Scala

Overview

In this lab, you'll face three puzzles. Each puzzle asks you to implement a new control-flow structure *for* Scala *in* Scala.

Implementing a control-flow structure is a lot like implementing a data structure: Just as you implement a new data structure by using *pre-existing* data structures, you implement a new control-flow structure by using pre-existing control-flow structures.

A more detailed description is below. Here are a few notes about the control-flow structures:

- When solving a puzzle, reverse-engineer it from the interface (i.e., what the user types). Looking at this interface, what kinds of Scala structures (e.g., functions and methods) could be used in that way?

by
read overview & set up

1:55

by
implement loop_until

2:15

by
implement repeat_until

2:20

If you thought goto was considered harmful...

```
10 COMEFROM 40
```

```
20 INPUT "WHAT IS YOUR NAME? "; A$
```

```
30 PRINT "HELLO, "; A$
```

```
40 REM
```

comes from en.wikipedia.org/wiki/COMEFROM