Lessons & Plans
# Projects

## CS 111: Domain-Specific Languages

**Fall 2015**

MW 1:15pm–2:30pm in SHAN 2475

Piazza · Syllabus · GitHub Repo · Wiki

<table>
<thead>
<tr>
<th>Monday</th>
<th>Wednesday</th>
</tr>
</thead>
<tbody>
<tr>
<td>08 · 31</td>
<td>09 · 02</td>
</tr>
<tr>
<td>(no class)</td>
<td>DSLs: What, why, and how?</td>
</tr>
<tr>
<td></td>
<td><em>writeup</em></td>
</tr>
<tr>
<td>09 · 07</td>
<td>09 · 09</td>
</tr>
<tr>
<td>What is a DSL?</td>
<td>Mini-lab: library design</td>
</tr>
<tr>
<td><em>writeup</em></td>
<td><em>board notes · lab · slides</em></td>
</tr>
<tr>
<td>09 · 14</td>
<td>09 · 16</td>
</tr>
<tr>
<td>Projects: interviews</td>
<td>Scala intro</td>
</tr>
<tr>
<td><em>writeup</em></td>
<td><em>writeup · lab</em></td>
</tr>
<tr>
<td>09 · 21</td>
<td>09 · 23</td>
</tr>
<tr>
<td>Functional Scala</td>
<td>Lab: Internal control flow</td>
</tr>
<tr>
<td><em>writeup · slides</em></td>
<td><em>writeup · lab</em></td>
</tr>
<tr>
<td>09 · 28</td>
<td>09 · 30</td>
</tr>
<tr>
<td>Object-oriented Scala</td>
<td>Extensibility: Traits</td>
</tr>
<tr>
<td>_writeup · slides · code</td>
<td><em>writeup · slides · code</em></td>
</tr>
<tr>
<td>10 · 05</td>
<td>10 · 07</td>
</tr>
<tr>
<td>Extensibility: The Expression Problem</td>
<td>Language architecture &amp; parser combinators</td>
</tr>
<tr>
<td>_writeup · slides · code</td>
<td><em>writeup · slides · lab</em></td>
</tr>
<tr>
<td>10 · 12</td>
<td>10 · 14</td>
</tr>
<tr>
<td>Models of computation</td>
<td>Lessons and plans</td>
</tr>
<tr>
<td><em>writeup · slides</em></td>
<td><em>studio</em></td>
</tr>
<tr>
<td>10 · 19</td>
<td>10 · 21</td>
</tr>
<tr>
<td>Fall Break (no class)</td>
<td><em>studio</em></td>
</tr>
<tr>
<td></td>
<td><em>Language features, part 1</em></td>
</tr>
<tr>
<td>10 · 26</td>
<td>10 · 28</td>
</tr>
<tr>
<td><em>studio</em></td>
<td><em>Language features, part 2</em></td>
</tr>
<tr>
<td>10 · 30</td>
<td>10 · 31</td>
</tr>
<tr>
<td></td>
<td><em>Project pitch</em></td>
</tr>
<tr>
<td></td>
<td><em>Studio</em></td>
</tr>
<tr>
<td>10 · 31</td>
<td>10 · 35</td>
</tr>
<tr>
<td></td>
<td><em>Project description and plan</em></td>
</tr>
<tr>
<td>What is a DSL?</td>
<td>_Submission and notebook: Sunday 11/7 at 11:59pm</td>
</tr>
<tr>
<td>Submission: Sunday 9/6 at 11:59pm</td>
<td>_Critique: Tuesday 10/07 at 11:59pm</td>
</tr>
<tr>
<td>Critique: Tuesday 9/8 at 11:59pm</td>
<td><em>Lab: Internal control flow</em></td>
</tr>
<tr>
<td>Language design</td>
<td>_Submission: Sunday 9/13 at 11:59pm</td>
</tr>
<tr>
<td><em>writeup · lab</em></td>
<td><em>Critique: Tuesday 9/15 at 11:59pm</em></td>
</tr>
<tr>
<td>Practice Scala</td>
<td>_Submission: Sunday 9/20 at 11:59pm</td>
</tr>
<tr>
<td><em>writeup · lab</em></td>
<td><em>Critique: Tuesday 9/22 at 11:59pm</em></td>
</tr>
<tr>
<td>Project proposals</td>
<td>_Submission: Saturday 9/27 at 11:59pm</td>
</tr>
<tr>
<td><em>writeup · lab</em></td>
<td><em>Critique: Tuesday 9/29 at 11:59pm</em></td>
</tr>
<tr>
<td>Internal DSL: piconot</td>
<td>_Submission: Sunday 10/14 at 11:59pm</td>
</tr>
<tr>
<td>_writeup · slides · code</td>
<td><em>Critique: Tuesday 10/16 at 11:59pm</em></td>
</tr>
<tr>
<td>External DSL: piconot</td>
<td>_Submission: Wednesday 10/14 at 11:59pm</td>
</tr>
<tr>
<td>_writeup · slides · code</td>
<td><em>Critique: Friday 10/16 at 11:59pm</em></td>
</tr>
<tr>
<td>Fall Break (no class)</td>
<td><em>studio</em></td>
</tr>
<tr>
<td><em>writeup · slides</em></td>
<td><em>Language features, part 1</em></td>
</tr>
<tr>
<td>10 · 26</td>
<td>10 · 28</td>
</tr>
<tr>
<td><em>studio</em></td>
<td><em>Language features, part 2</em></td>
</tr>
<tr>
<td>10 · 30</td>
<td>10 · 31</td>
</tr>
<tr>
<td></td>
<td><em>Project pitch</em></td>
</tr>
<tr>
<td>10 · 31</td>
<td>10 · 35</td>
</tr>
<tr>
<td></td>
<td><em>Project description and plan</em></td>
</tr>
<tr>
<td>11/7 at 11:59pm</td>
<td>_Submission and notebook: Sunday 11/7 at 11:59pm</td>
</tr>
<tr>
<td>11/9 at 11:59pm</td>
<td><em>Critique: Tuesday 10/31 at 11:59pm</em></td>
</tr>
</tbody>
</table>
Philosophies

- People first
- Programming as language design
- How does language influence thought?
- The end of PLs

Concepts

- Language design
- Parsing
- Semantic models
- Software engineering
- Extensibility
- ...

Skills

- Scala
- GitHub
- Testing
- Build system (sbt)
How did the first half go?

(1) DSLs (i.e., CS 111) is interesting.
   $1 = \text{strongly disagree}; \ 4 = \text{neither agree nor disagree}; \ 7 = \text{strongly agree}$

(2) I’m learning a lot in DSLs.
   $1 = \text{strongly disagree}; \ 4 = \text{neither agree nor disagree}; \ 7 = \text{strongly agree}$

(3) The workload seems appropriate for the course.
   $1 = \text{strongly disagree}; \ 4 = \text{neither agree nor disagree}; \ 7 = \text{strongly agree}$

(4) I know I can get help / support / advice, if I need it.
   $1 = \text{strongly disagree}; \ 4 = \text{neither agree nor disagree}; \ 7 = \text{strongly agree}$

studio model

topic selection

in-class activities

discussions

assignments
Philosophies

- People first
- Programming as language design
- How does language influence thought?
- The end of PLs

Concepts

- Language design
- Parsing
- Semantic models
- Software engineering
- Extensibility
- ...

Skills

- Scala
- GitHub
- Testing
- Build system (sbt)
A taste of metaprogramming
Python decorators

@decorator
def f():
    ...

≈
def f():
    ...
    f = decorator(f)
Something more useful

```python
def logging(f):
    def wrapper(*args, **kwargs):
        print 'Calling {0} with {1} and {2}'.format(f.func_name, args, kwargs)
        return f(*args, **kwargs)
    return wrapper

@logging
def f(x,y):
    return x+y
```
A clean integration

```
from functools import wraps

def logging(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        print 'Calling {0} with {1} and {2}'.format(f.func_name, args, kwargs)
        return f(*args, **kwargs)

    return wrapper

@logging
def f(x,y):
    return x+y
```
Classes as decorators

from functools import wraps

class logging(object):
    def __init__(self, handle=sys.stdout):
        self.handle = handle

    def __call__(self, f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            print >> self.handle, \
            'Calling {0} with {1} and {2}'.format(f.func_name, args, kwargs)
            return f(*args, **kwargs)

        return wrapper

@logging(file('log.txt', 'a'))
def g(x,y):
    return x**y
Another decorator

```python
from functools import wraps

def memoize(f):
    cache = {}

    @wraps(f)
    def wrapper(*args, **kwargs):
        key = (args, tuple(kwargs.values()))
        if key not in cache:
            cache[key] = f(*args, **kwargs)
        return cache[key]
    return wrapper

@memoize
def g(x, y):
    return x**y
```
Decorators are composable

```python
@memoize
@logging
def fib(n):
    if n==0 or n==1:
        return 1

    return fib(n-1) + fib(n-2)
```
make(p):

    if potion p is not yet in the pantry:
        foreach potion $p_i$ in p’s ingredients:
            make($p_i$)
        add p to the pantry
        proclaim “Made p!”

    fetch p from the pantry