Recap of the course so far

- 4 weeks of functional with OCaml
- 1 lecture on modularity with Aspects
- Next: 4 weeks of OO with Python
- After that: 1 week of constraint logic programming with Prolog

This stuff is for real: F#

F# = Microsoft’s Ocaml-on-steroids
- Why FP is way cool
- How FP works with Objects (C#)
- How FP allows you to write parallel code
... all with an extremely engaging speaker

News for next lecture

- We will also do KQS
- Dude, what’s KQS?
- It’s a way for you to tell us what we’re doing well on, and what we’re doing poorly on!
- So that we can improve the class

KQS

- Will handout a sheet with three questions:
  - what things do you want me to Keep doing?
  - what things do you want me to Quit doing?
  - what things do you want me to Start doing?
- You can fill out answers in prose to these questions. Start thinking early!
**OO at the highest level**

- What is OO programming?

*Answer:*
- Objects
- message sends
- dynamic dispatch

**Just to whet your appetite**

- Say we have objects, like cars, ducks, pig, cell_phones

- Say we have a message name: make_some_noise

- Each object has its own implementation for make_some_noise: these are traditionally called methods.

- car: vroom vroom, pig: oink oink, duck: quack quack

- We can send make_some_noise to any object. Depending on the actually run-time object, we’ll get a different noise!
**Oh btw...**

- What’s the difference between message and method...

**OO programming**

- Message: the name of an operation
- Method: the implementation of an operation
- Dynamic dispatch: the act of determining at based on the dynamic type which method should be run for a given message send.

- These are the core ideas of OO

**Oh btw...**

- What’s the difference between message and method...

- Message is just the name of the message, method is the implementation

- Message is the “interface”/”prototype” of the method.

**This brings us to Python...**

- We’ll use Python as our vehicle for OO programming

- Fun and useful language

- Let’s compare with OCaml along some of the dimensions we saw last time
OCaml/Python comparison

<table>
<thead>
<tr>
<th></th>
<th>ML</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL paradigm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataModel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Python

- Python has a very relaxed philosophy
  - if something "can be done" then it is allowed.

- Combination of dynamic types + everything is an object makes for very flexible, very intuitive code.

No static types

- No static type system to "prohibit" operations.
- No more of that OCaml compiler giving you hard-to-decypher error messages!
- And... No need to formally define the type system (although still need to define the dynamic semantics somehow)
No static types: but what instead?

- **Dynamic typing**
- At runtime, every "operation" is translated to a method call on the appropriate object. If the object supports the method, then the computation proceeds.
- Duck-typing: if it looks like a duck, quacks like a duck, then it is a duck!

Dynamic typing

- This loose, comfortable, free-style philosophy is at the heart of python.
- But... beware, you can get burned with this flexibility...
- Q: how many times did OCaml complain to you statically about something that was NOT a bug?

Similarities to Ocaml

- Uniform model: everything is an object, including functions
- Can pass functions around just as with objects
- Supports functional programming style with map and fold

Other cool things about Python

- A lot of stuff that you may first think is a "language feature" is actually just translated under the hood to a method call...
- Very widely used, supported.
- Has libraries for all sorts of things.
Ok, let’s start playing with Python!

- Like Perl, python is a "managed" or "interpreted" language that runs under the python environment, i.e. not compiled to machine code.
- Makes it convenient to rapidly write, check-in and test code!

Let’s fire it up!

- Ok, let’s give it a try...
- See lecture-9.py file for the rest...

Ways to run Python code

- At an interactive Python prompt: like "read-eval-print" loop of ML,
- As shell scripts,
- As stand-alone programs run from the shell.