## Key components of a lang

- Units of computation
- Types
- Memory model

### Units of computation

In OCaml

### In OCaml
In OCaml

- Expressions that evaluate to values
- **Everything** is an expression
  - int, bool, real
  - if-then-else
  - let-in
  - match
  - fun x -> x+1
  - e1 e2

In Java/Python/C++/C/…

In Java/Python/C/C++

- Store and update commands
  - \( x := 2 \times y \)
- Message sends
  - \( \text{obj}.\text{set}X(12) \)
In Java/Python/C/C++

- Store and update commands
  - $X := 2 \times y$
- Message sends
  - $\text{obj}.\text{setX}(12)$

In Prolog

- Logical facts
- Inference rules

- CAT (Carnivore)
- atoms
- $\text{Delicious}(X)$
In Prolog

- Logical facts
- Inference rules

\[
\text{Mexican(CARNITAS)} \quad \text{Food(CARNITAS)} \\
\text{Mexican(X)} \Rightarrow \text{Food(X)} \Rightarrow \text{Delicious(X)} \\
\text{Delicious(CARNITAS)}
\]
Memory/Data model

aka: what do variables refer to?

Data model in functional langs

- Environment of bindings (phonebook)

<table>
<thead>
<tr>
<th>X</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>a/b/c</td>
</tr>
<tr>
<td>z</td>
<td>[1;2;3]</td>
</tr>
</tbody>
</table>

- Never change a binding
  - add new bindings at the end of phonebook

- Variables = names in phonebook
- Most recent entry looked up during eval
Data model in functional langs

- Variables = names in phonebook
- Most recent entry looked up during eval

Closures:
- Environment “frozen” in function value
  - behavior of function cannot be changed later
  - easier reasoning, debugging

Data model in Imp/OO langs

- Variables are named cells in memory

\[ X = 5 \]

\[ X \rightarrow 100 \]
\[ Y \rightarrow 3.142 \]
\[ Z \rightarrow \text{“pumpkin”} \]
Data model in Imp/OO langs

- Variables are named cells in memory
  Can change them by assigning into them
  \[ X = 5 \]

- Variables = names of objects on the heap

\[ \begin{align*}
X & \rightarrow 100 \\
Y & \rightarrow 3.142 \\
Z & \rightarrow "pumpkin"
\end{align*} \]

Thursday, November 10, 2011
Data model in Imp/OO langs

- Variables are named cells in memory
  Can change them by assigning into them

\[ X = 5 \]

- Variables = names of objects on the heap

- \( x = x + 10 \)

\[
\begin{array}{c}
X & \rightarrow 100 \\
Y & \rightarrow 3.142 \\
Z & \rightarrow \text{“pumpkin”}
\end{array}
\]

Data model in Prolog

- Variables = unknowns to solve for

\[
\begin{array}{c}
\text{Mexican(CARNITAS)} \\
\text{Food(CARNITAS)} \\
8 \times \text{Mexican}(X) \land \text{Food}(X) \land \text{Delicious}(X) \\
\text{Delicious}(Y)\
\end{array}
\]

Q: What is delicious?
A: CARNITAS!
Types

- Used to classify things created by the programmer
- Classification used to check what can be done with/to those things

In OCaml: Static typing

- Types assigned statically (at compile time)
- Without computing values
- Rules say when expressions are type-correct

\[ e_1 : T_1 \rightarrow T_2 \quad e_2 : T_1 \]
\[ e_1 \; e_2 : T_2 \]

In OCaml: Static typing

- How to reuse code for different types?
  - parametric types: ‘a * ‘b -> ‘b * ‘a
  - implicit for-all
- Type “discovered” (inferred) automatically from code
  - less burden on the programmer
In Python: Dynamic typing

- Types assigned to values/objects as they are computed, ie: dynamically

- Before op is executed, check that operands are compatible with op

\[
def f(x):
    x. m() 
    f(a)
\]

In Python: Dynamic typing

- More programs are accepted by compiler i.e. More flexible

[1, "abc", 1.8, [ "efg", 20]]
In Python: Dynamic typing

• More programs are accepted by compiler
  i.e. More flexible

    [1, “abc”, 1.8, [ “efg”, 20]]

    let x = if b then 1 else “abc”
    let y = if b then x + 1 else x ^ “efg”

Thursday, November 10, 2011
In Python: Dynamic typing

- More programs are accepted by compiler i.e. More flexible

```python
[1, “abc”, 1.8, [ “efg”, 20]]
```

```python
let x = if b then 1 else “abc”
let y = if b then x + 1 else x ^ “efg”
```

Dynamic vs. Static, OO vs. Func

<table>
<thead>
<tr>
<th>Statically typed</th>
<th>Dynamically typed</th>
</tr>
</thead>
<tbody>
<tr>
<td>OO</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td></td>
</tr>
</tbody>
</table>

Polymorphism

- PL that is polymorphic + dynamically typed?
Polymorphism

- PL that is polymorphic + dynamically typed?

- Every dynamically typed PL is polymorphic
  - functions simply work on any datatype that can be operated on at runtime

Explicit polymorphism in statically typed PL
- assign at compile time a general polymorphic type

Final words on functional programming
What’s the point of all this?

Advantages of functional progs

- Functional programming more concise
  “one line of lisp can replace 20 lines of C” (quote from http://www.ddj.com/dept/architect/184414500?pgno=3)

- Recall reverse function:

```haskell
let reverse = fold (::) [];;
```

- How many lines in C, C++?

Better reasoning about Programs

- No side effects
  - Same inputs return same outputs

- So, can safely reorder computations

- So, can also parallelize computations

An example: Map/Reduce

- Scalable data processing:
  - MapReduce programming model
  - Hadoop [Yahoo], Hive [Facebook], Dryad [MSR]

- Programming model based on map, fold
  - apply a function on each data element (map)
  - aggregate the results (fold)

- Functional ) easy parallelism ) scalable

- For more, take Prof. Vahdat’s CSE 124
So what?

• Form the authors: “Inspired by similar primitives in LISP and other languages”

• Programmers who only know Java/C/C++ would probably not have come up with this idea

Remember

“Free your mind”
-Morpheus

Say hello to Python
Recap of the course so far

• 4+ weeks of functional with OCaml
• Next: 3 weeks of OO with Python

Thursday, November 10, 2011

OO at the highest level

• What is OO programming?

Thursday, November 10, 2011

OO at the highest level

• What is OO programming?

Answer:
– Objects
– Message sends
– Dynamic dispatch

Thursday, November 10, 2011

Just to whet your appetite

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

• Say we have a message name: make_some_noise

Thursday, November 10, 2011
Just to whet your appetite

• Each object has its own implementation for \texttt{make\_some\_noise}: these are traditionally called methods.
  – \texttt{car}: vroom vroom,
  – \texttt{pig}: oink oink,
  – \texttt{duck}: quack quack

• Can send \texttt{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...

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

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

OO programming

• Message:
  - the name of an operation

• Method:
  - the implementation of an operation

• Dynamic dispatch:
  - dynamic type of object determines
  - which method is run for given message send

• These are the core ideas of OO
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>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.

- Dynamic types + Everything is an object
  - 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-decipher error messages!
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 ML

- Uniform model
  - everything is an object, including functions

- Pass functions around
  - functions are objects!

- Supports functional programming
  - 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-10.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.