AGH

GEIST

Department of Automatics

Introduction to Python and Lisp

Sławomir Nowaczyk

Laboratorium Informatyki

Katedra Autormatyki

Akademia Górniczo-Hutnicza

January 21, 2009

AGH

GEIST

Department of Automatics

Languages of AI

- Special purpose languages (SQL, XSL, awk, ...)
  - PROLOG
  - PDDL: Planning Domain Definition Language
  - Bertrand: Constraint Programming
  - KQML, Phantom, SAIL, ARL, Brahms, CEL, AgentSpeak, MAML, ...
- General purpose languages
  - exploratory programming
  - mutable specifications
  - tricky problems
  - limited system integration
  - expert users

A G H

GEIST

Department
of Automatics

Powerful Language

- Expressive abstraction mechanisms
  - group *similar* concepts together
- Ways to avoid code duplication
  - specify small *differences* efficiently
- Express complex concepts cleanly
  - clear mapping into *“programmer’s head”*
- Programming skills & learning curve
  - will bad programmer blow things up?
  - handling idiosyncrasies within a project
- Mental effort
  - is coding more like puzzle-solving?

Abstraction Mechanisms

- Functions

```
if(action.equals("+"))
{
  b = Integer.valueOf(stack.removeLast());
  a = Integer.valueOf(stack.removeLast());
  result = a+b;
  stack.addLast(String.valueOf(result));
  updateDisplay();
  System.out.println(String.valueOf(a) +
    "+" + String.valueOf(b) + "=" +
    String.valueOf(result));
}
```

GEIST

Department of Automatics

AGH

Functional Abstraction

```
interface Operation
{ int apply(int a, int b); }
class AddOperation implements Operation
{ int apply(int a, int b) {return a+b;} }

void doComputation(Operation operation) {
  b = Integer.valueOf(stack.removeLast());
  a = Integer.valueOf(stack.removeLast());
  result = operation.apply(a,b);
  stack.addLast(String.valueOf(result));
  updateDisplay();
  System.out.println(String.valueOf(a) +
    "+" + String.valueOf(b) + "=" +
    String.valueOf(result));
}
```

GEIST

Department of Automatics

AGH

Functional Abstraction

```
interface ZeroDiv
{ bool isValid(int a, int b) return true; }
class ZeroDivReal implements ZeroDiv
{ int isValid(int a, int b) return b!=0; }

void doComputation(Operation operation,
                   ZeroDiv zerodiv) {
  b = Integer.valueOf(stack.removeLast());
  a = Integer.valueOf(stack.removeLast());
  if (zerodiv.isValid()) {
    result = operation.apply(a,b);
    stack.addLast(String.valueOf(result));
    updateDisplay();
    System.out.println(...);
  } }
```

GEIST

Department of Automatics

AGH

Functional Abstraction

```
void doComputation(Operation operation,
                   ZeroDiv zerodiv,
                   StackItemSelector sel,
                   ArgumentsOrder order,
                   Associativity assoc,
                   Display display,
                   ...)

{
  ...
}
```

GEIST

Department of Automatics

AGH

File Handling

GEIST

Department of Automatics

```
FileOutputStream out;
PrintStream p;

try
{
  out = new FileOutputStream("myfile.txt");
  p = new PrintStream( out );
  p.println("This is written to a file");
  p.close();
}
catch (Exception e)
{
  System.err.println ("Error");
}
```

AGH

GEIST

Department of Automatics

File Handling

```
FileOutputStream out;
PrintStream p;

try
{
  out = new FileOutputStream("myfile.txt");
  p = new PrintStream( out );
  p.println( getResult() );
  p.close();
}
catch (Exception e)
{
  System.err.println ("Error");
}
```

AGH

File Handling

```
FileOutputStream out;
PrintStream p;

try
{
  out = new FileOutputStream("myfile.txt");
  p = new PrintStream( out );
  p.println( getResult() );
}
finally
  { p.close(); }
catch (Exception e)
  { System.err.println("Error"); }
```

GEIST

Department
of Automatics

AGH

Resource Acquisition Is Initialization

GEIST

Department
of Automatics

```
void main() {
  file myfile("myfile.txt");
  myfile.write( getResult() );
}

class file {
  FILE* file_;
  file (const char* filename)
        : file_(fopen(filename, "w+")) {
    if (!file_)
      throw runtime_error("failure");
  }
  ~file() { fclose(file_) }
}
```

A G H

Programming Languages Evolution

- General-purpose programming languages are becoming more and more *Lisp-like*
  - Lisp is only 1 year younger than Fortran
  - 14 years older than C
  - 33 years older than Python
  - 37 years older than Java
- Code readability
- Dynamism
- Compiled to a byte code
- Interpreted by virtual machine
- Garbage collection

GEIST

Department of Automatics

AGH

GEIST

Department of Automatics

Basic Python Features

- Interactive *shell*
  - incredibly useful for debugging
- Rich container types
- Functions, classes, modules
- Namespaces
- Exceptions
- Portable
- Powerful introspection
- Plenty of dynamic features
- Extremely rich standard library

GEIST

Department
of Automatics

Python Applications

- Industrial Light & Magic
- ForecastWatch.com
- Frequentis TAPtools
- AstraZeneca
- MayaVi
- YouTube.com
- Google
- Journyx
- EZTrip.com
- Firaxis Games
- ...

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

GEIST

Department of Automatics

AGH

Python Design

- Readability
  - “Python is executable pseudocode”
  - whitespace is significant (no brackets)
  - use English keywords instead of punctuation
- Multi-paradigm programming language
- Strongly dynamically typed language
  - variables are not declared
  - any value can be assigned
  - values “carry their own types”
    - `a = 1; a = "x"` # OK
    - `1 == "1"` # False

GEIST

Department of Automatics

AGH

Basic Python Features

GEIST

Department
of Automatics

- Numeric types: long, complex & rational
- Both “normal” and unicode strings
  - immutable
- Dictionaries as fundamental data type
  - hash tables with immutable keys
  - internally storing fields & methods of classes
- Lists and tuples
  - flexible arrays (*not* linked lists)
  - indexing, slices & other basic operations
- Assignment manipulates references
  - `x = y` does not make a copy

A G H

Basic Python Features

- “Batteries-included” approach to standard library
  - one of Python’s greatest strengths
  - makes Python a powerful *glue language*
- Numerous high-level data types
  - tuple
  - set & frozenset
  - datetime, calendar
  - heapq
  - queue, deque
  - defaultdict
  - namedtuple
  - weakref

GEIST

Department of Automatics

A G H

Standard Library

- Core
  - os, sys, string, getopt, struct, pickle, re,
- Internet
  - socket, rfc822, httplib, htmllib, ftplib, smtplib,
- Data types
  - datetime, calendar, sets, mutex, weakref,
- Operating system
  - threading, select, mmap, ctypes, platform,
- Miscellaneous
  - pdb, profile, Tkinter, audio, dbm, xml, distutils, zipfile

GEIST

Department of Automatics

AGH

Control Structures

```
if x < 0:
  print "negative"
elif x > 0:
  print "positive"
else:
  print "zero"

for x in argv:
  print x

while True:
  if not done:
    continue
  break
```

GEIST

Department
of Automatics

AGH

GEIST

Department
of Automatics

Functions

```
def add(x, y): return x+y
def add(x=0, y=0):
  "documentation string"
  return x+y

def add(*args):      # add(1,2,3,4,5)
  res = 0
  for x in args:  res += x
  return res

def display(**arg):  # display(a=1,b=2,c=3)
  for i in arg:
    print "%s: %s" % (i, arg[i])
```

GEIST

Department of Automatics

Lists

```
s = [1,2,"a","b",[1,2],{1:2,3:4}]
s[i] = x
s[i:j] = t
del s[i:j]
s.append(x)
s.extend(x)
s.count(x)
s.index(x[, i[, j]])
s.insert(i, x)
s.pop([i])
s.remove(x)
s.reverse()
s.sort([cmp[, key[, reverse]]])
```

AGH

Some More Exotic Features

- Iterators
  - `for line in file("fname"):`
- Anonymous functions

```
>>> sorted(["a","B","c"])
['B', 'a', 'c']
>>> sorted(["a","B","c"],
             key=lambda x: x.lower())
['a', 'B', 'c']
```

- “else” clause in loops

```
for x in [1,2,3,4]:
   if x==5: break
else: print "Not found"
```

GEIST

Department of Automatics

AGH

Generators

- Functions with “resume” capability

```
def Fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b
```

- Usable as iterators

```
for i in Fibonacci():
    print i
```

- Generator expressions

```
sum( i*i for i in range(10) )
```

GEIST

Department of Automatics

Resource Acquisition

- Context managers

```
with open("/etc/passwd", "r") as File:
    File.write( getResult() )
```

- More or less equivalent to

```
__manager = open("/etc/passwd", "r")
File = __manager.__enter__()
try:
    File.write( getResult() )
except Exception, e:
    if not __manager.__exit__(e):
        raise e
```

GEIST

Department
of Automatics

AGH

GEIST

Department of Automatics

Properties

```
class myclass(object):
    def __init__(self):
        self.__x = None

    def getx(self):            return self.__x
    def setx(self, value): self.__x = value
    def delx(self):            del self.__x
    x = property(getx, setx, delx,
                 "I’m the ’x’ property.")


a = myclass()
a.x = 5     # Set
print a.x   # Get
del a.x     # Del
```

AGH

GEIST

Department of Automatics

Some Code Examples

- Decode a base64 encoded file

```
import base64, sys
fin = open(sys.argv[1], "rb")
fout = open(sys.argv[2], "wb")
base64.decode(fin,fout)
```

- Download a web page

```
import urllib2
f = urllib2.urlopen('http://www.python.org/')
print f.read(1024)
```

Some Examples

- Count *bytes-transferred* from apache log

```
81.107... "GET /ply/ HTTP/1.1" 200 7587
81.107... "GET /fav.ico HTTP/1.1" 404 133
```

```
wwwlogfile = open("access-log")
bytecolumn = (line.rsplit(None,1)[1]
                for line in wwwlog)
bytescount = (int(x) for x in bytecolumn
                if x != ’-’)
print "Total", sum(bytes)
```

GEIST

Department of Automatics

AGH

GEIST

Department of Automatics

Some Examples

- Get all links in a web page

```
import HTMLParser, urllib
class linkParser(HTMLParser.HTMLParser):
  def __init__(self):
    HTMLParser.HTMLParser.__init__(self)
    self.links = []
  def handle_starttag(self, tag, attrs):
    if tag=='a':
      self.links.append(dict(attrs)['href'])
htmlSource = urllib.urlopen(
            "http://www.python.org/").read()
p = linkParser().feed(htmlSource)
for link in p.links:
  print link
```

AGH

Some Examples

- Fetch, read and parse RSS

```
import urllib, sys, xml.dom.minidom
address = 'http://www.example.org/rss'
txt = urllib.urlopen(address)
doc = xml.dom.minidom.parse(txt)
for item in doc.getElementsByTagName('item'):
    title = item.getElementsByTagName(
                 'title')[0].firstChild.data
    print "Title:",
            title.encode('latin-1','replace')
```

GEIST

Department
of Automatics

AGH

GEIST

Department of Automatics

Metaclasses

- Very powerful mechanism
    - not for the faint of heart

```
def class_with_method(func):
  class klass: pass
  setattr(klass, func.__name__, func)
  return klass

def say_foo(self): print 'foo'

Foo = class_with_method(say_foo)
foo = Foo()
foo.say_foo()
```

AGH

GEIST

Department of Automatics

Metaclasses, cont.

- Metaclasses: a solution looking for a problem?

Metaclasses are deeper magic than 99% of users should ever worry about. If you wonder whether you need them, you don't (the people who actually need them know with certainty that they need them, and don't need an explanation about why).
— Python Guru Tim Peters

AGH

GEIST

Department of Automatics

Metaclasses Example

```
class Printable(type):
  def whoami(cls):
    print "I am a", cls.__name__

Foo = Printable('Foo',(),{})
Foo.whoami()

class Bar():
    __metaclass__ = Printable
    def who(self): print "Who?"
Bar().who()
Bar.whoami()
```

AGH

Aspect-Oriented Programming

```
if len(sys.argv) > 2:
  mod, mc  = sys.argv[1:3]
  m = __import__(mod,globals(),locals(),[mc])
  __metaclass__ = getattr(m, mc)

class Data:
  def __init__(self):
    self.num, self.str = 38, "spam"
    self.lst = [’a’,’b’,’c’]
  def dumps(s): return str((s.__dict__))

print Data()

{’lst’:[’a’,’b’,’c’],’num’:38,’str’:’spam’}
```

GEIST

Department
of Automatics

Aspect-Oriented Programming

```
% dump.py gnosis.magic MetaXMLPickler
<?xml version="1.0"?>
<!DOCTYPE PyObject SYSTEM "PyObjects.dtd">
<PyObject module="__main__" class="Data" id="
<attr name="lst" type="list" id="980012" >
  <item type="string" value="a" />
  <item type="string" value="b" />
  <item type="string" value="c" />
</attr>
<attr name="num" type="numeric" value="38" />
<attr name="str" type="string" value="spam" /
</PyObject>
```

GEIST

Department of Automatics

Aspect-Oriented Programming

```
class MetaPickler(type):
  "Metaclass for gnosis.xml serialization"
  def __init__(cls, name, bases, dict):
    from gnosis.xml.pickle import dumps
    super(MetaPickler, cls).__init__(
                      name, bases, dict)
    setattr(cls, 'dumps', dumps)
```

GEIST

Department
of Automatics

A G H

GEIST

Department
of Automatics

Dynamic language

```
py> from ctypes import *
py> class X:
...   def __init__(s, foo): s.foo = foo
py> x = X(123)
py> x.foo
123
py> p = cast(c_char_p("foo"),POINTER(c_char))
py> p[0],p[1],p[2] = "b","a","r"
py> x.foo
AttributeError: X has no attribute ’foo’
py> dir(x)
[’__doc__’, ’__init__’, ’__module__’, ’bar’]
py> x.bar
AttributeError: X has no attribute ’bar’
```

A G H

Django

- High-level Python Web framework
  - rapid development
  - clean design
- Object-relational mapper
  - automatically manage DB
  - rich API available
- Automatically generated admin interface
- Flexible URL configuration
- Rich template definition language
- Customisable caching framework
- Syndication-feed-generating framework

GEIST

Department
of Automatics

Django Framework

```
#> django-admin.py startproject mysite
mysite/
    __init__.py
    manage.py
    settings.py
    urls.py

#> python manage.py runserver
Validating models...
0 errors found.

Django version 1.0, using settings ’mysite.se
Development server is running at http://127.0
```

GEIST

Department of Automatics

AGH

Django Framework

```
#> vi settings.py
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': 'mydatabase'
} }
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'mysite.polls' )
#> python manage.py syncdb
```

GEIST

Department
of Automatics

AGH

Django Framework

```
#> python manage.py startapp polls
#> vi polls/models.py

from django.db import models

class Poll(models.Model):
  question = models.CharField(max_length=200)
  pub_date = models.DateTimeField(
                           ’date published’)

class Choice(models.Model):
  poll = models.ForeignKey(Poll)
  choice = models.CharField(max_length=200)
  votes = models.IntegerField()
```

GEIST

Department
of Automatics

AGH

Lisp

- LISt Processing language
  - primary data structure is a linked list
  - `(elem1 elem2 elem3 elem4)`
  - source code is organised in the same fashion
  - readily-available *abstract syntax tree*
- Code and data are interchangeable
  - basic syntax of the language is very simple
  - heavily customisable using *macros*
- Read-Eval-Print Loop
  - read *s-expression*
  - evaluate resulting *Lisp form*

GEIST

Department
of Automatics

AGH

Lambda Calculus

- Lambda terms
  - variable x
  - $\lambda x.t$
  - ts
- Identity function $\lambda x.x$
- Constant function $\lambda x.y$
- Function application $(\lambda x.x)y$

GEIST

Department of Automatics

A G H

Lisp Applications

- Emacs
  - *the extensible, customizable, self-documenting, real-time display editor*
  - large portion of code written in Lisp
  - Lisp is the extension language
  - by far the best IDE in existence
- AutoCAD
- Script-Fu plugins for GIMP
- Remote Agent (NASA Deep Space 1, 1998)
- ITA Software's airline engine
- Yahoo Store
- ...

GEIST

Department
of Automatics

AGH

GEIST

Department of Automatics

Language Evolution

- Paul Graham, The Roots of Lisp, May 2001:

It seems to me that there have been two really clean, consistent models of programming so far: the C model and the Lisp model. These two seem points of high ground, with swampy lowlands between them. As computers have grown more powerful, the new languages being developed have been moving steadily toward the Lisp model. A popular recipe for new programming languages in the past 20 years has been to take the C model of computing and add to it, piecemeal, parts taken from the Lisp model, like runtime typing and garbage collection.

A G H

First Language to Have...

- Conditionals: if-then-else constructs
- A function type
- Recursion
- Typed values rather than typed variables
- Dynamic memory allocation
- Garbage collection
- Incremental compilation
- Built-in extensibility
- The whole language always available
  - programs are capable of constructing and executing other programs on the fly

GEIST

Department of Automatics

Learning Lisp

Lisp is worth learning for a different reason: the profound enlightenment experience you will have when you finally get it. That experience will make you a better programmer for the rest of your days, even if you never actually use Lisp itself a lot.

— Eric Raymond, the author of The Cathedral and the Bazaar, The Art of Unix Programming, The New Hacker's Dictionary

GEIST

Department of Automatics

AGH

Lisp Design

- Every Lisp object is either an atom or a list
- Empty list is both: `()`, `nil`
- `(a . (b . NIL)) == (a b)`
- `(+ 1 2) ==> 3`
- `'(+ 1 2) ==> (+ 1 2)`

```
(defun fib (n)
  (if (< n 3)
      1
      (+ (fib (- n 1) (fib (- n 2))))))

(format t "Hello, world!")
```

GEIST

Department
of Automatics

AGH

Lisp Basics

GEIST

Department of Automatics

```
(defun hello-world ()
  (format t "hello, world"))

CL-USER> (hello-world)
hello, world
NIL


(defvar *db* nil)
(push (list :name "A" :count 2 :value 3) *db*)
(push (list :name "B" :count 3 :value 8) *db*)
(defun dump-db ()
  (dolist (cd *db*)
    (format t "~{~a:~10t~a~%~}~%" cd)))
```

AGH

Lisp Basics

```
(defun save-db (filename)
  (with-open-file (out filename
                       :direction :output
                       :if-exists :supersede)
    (with-standard-io-syntax
      (print *db* out))))


(defun load-db (filename)
  (with-open-file (in filename)
    (with-standard-io-syntax
      (setf *db* (read in)))))
```

GEIST

Department
of Automatics

GEIST

Department of Automatics

Lisp Examples

```
CL-USER> (setf *print-circle* t)
CL-USER> (setq x '(a b c))
(A B C)
CL-USER> (setq y '(d e f))
(D E F)
CL-USER> (nconc x y)
(A B C D E F)
CL-USER> x
(A B C D E F)
CL-USER> y
(D E F)
CL-USER> (nconc x y)
(A B C . #1=(D E F . #1#))
```

Lisp Examples

```
((a 1) (b 2) (c 3))
==>
((1 a) (2 b) (c 3))

(defun reverse-inner (lsts)
  (mapcar #'reverse lsts))
```

GEIST

Department
of Automatics

AGH

Lisp Examples

```
CL-USER> (setq x
    ’((pear 3) (potatoe 7) (leek 5)))
((PEAR 3) (POTATOE 7) (LEEK 5))

CL-USER> (reduce #’+ x :key #’second)
15
```

GEIST

Department
of Automatics

Lisp Examples

```
CL-USER>
  (let ((seq '(1 2 3 4 5)))
    (list
      (position (reduce #'min seq) seq)
      (position (reduce #'max seq) seq)))
(0 4)
```

GEIST

Department of Automatics

AGH

GEIST

Department
of Automatics

Binary Tree in Lisp

```
(defun make-bin-tree-leaf (E)
    "Create a leaf."
    (list E))
(defun make-bin-tree-node (E B1 B2)
    "Create a node with element K, left
      subtree B1 and right subtree B2."
    (list E B1 B2))
(defun bin-tree-leaf-element (L)
    "Retrieve the element of a leaf L."
    (first L))
(defun bin-tree-node-element (N)
    "Retrieve the element of a node N."
    (first N))
```

Binary Tree in Lisp

```
(defun bin-tree-node-left (N)
    "Retrieve the left subtree of a node N."
    (second N))
(defun bin-tree-node-right (N)
    "Retrieve the right subtree of a node N."
    (third N))
(defun bin-tree-leaf-p (B)
    "Test if binary tree B is a leaf."
    (and (listp B) (= (list-length B) 1)))
(defun bin-tree-node-p (B)
    "Test if binary tree B is a node."
    (and (listp B) (= (list-length B) 3)))
```

GEIST

Department
of Automatics

AGH

GEIST

Department
of Automatics

Binary Tree in Lisp

```
CL-USER> (make-bin-tree-node '*
                   (make-bin-tree-node '+
                       (make-bin-tree-leaf 2)
                       (make-bin-tree-leaf 3))
                   (make-bin-tree-node '-
                       (make-bin-tree-leaf 7)
                       (make-bin-tree-leaf 8)))

(* (+ (2) (3)) (- (7) (8)))
```

A G H

Binary Tree in Lisp

```
(defun bin-tree-member-p (B E)
  "Test if E is an element in binary tree B."
  (if (bin-tree-leaf-p B)
      (equal E (bin-tree-leaf-element B))
    (let
        ((elmt  (bin-tree-node-element B))
         (left  (bin-tree-node-left    B))
         (right (bin-tree-node-right   B)))
      (or (equal E elmt)
          (bin-tree-member-p left E)
          (bin-tree-member-p right E)))))
```

GEIST

Department of Automatics

AGH

GEIST

Department of Automatics

Binary Tree in Lisp

```
(defun bin-tree-preorder (B)
  "Create a list containing keys
    of B in preorder."
  (if (bin-tree-leaf-p B)
      (list (bin-tree-leaf-element B))
    (let
        ((elmt  (bin-tree-node-element B))
         (left  (bin-tree-node-left    B))
         (right (bin-tree-node-right   B)))
      (cons elmt
         (append (bin-tree-preorder left)
                 (bin-tree-preorder right))))))
```

AGH

GEIST

Department of Automatics

Macros

- *The* most powerful feature of any language
  - code which is used to create more code
  - full-fledged *code generation* system
  - metaprogramming technique
- Macro takes *unevaluated Lisp code* as argument
  - and returns a new *Lisp form* to be evaluated

```
(defun backwards (expr) (reverse expr))
(backwards '(1 2 3))  ==> (3 2 1)
...
(defmacro backwards (expr) (reverse expr))
...
(backwards ("Hello, world" t format))
```

AGH

GEIST

Department of Automatics

Macros

- *The* most powerful feature of any language
  - code which is used to create more code
  - full-fledged *code generation* system
  - metaprogramming technique
- Macro takes *unevaluated Lisp code* as argument
  - and returns a new *Lisp form* to be evaluated

```
(defmacro backwards (expr) (reverse expr))
(backwards ("Hello, world" t format))
Hello, world
...
(macroexpand ’(backwards ("Hi" t format)))
(FORMAT T "Hi")
```

AGH

GEIST

Department of Automatics

Macros Explained

- Macros look like functions
  - `(defun add-f (a b) (+ a b))`
  - `(defmacro add-m (a b) (+ a b))`
- Macro returns a *form*, not a value
  - `(add-m 1 2) <==> 3`
- Macro gets *expanded* during *compilation*
  - `(macroexpand '(add-m 1 2)) ==> 3`
  - `(let ((a 1)) (add-m a 5))` is an error
  - at *compile-time*, since a is not a number

```
(defmacro add-m (a b) '(+ a b))
(macroexpand '(add-m 1 2))  ==> (+ A B)
(let ((a 1) (b 2)) (add-m a b)) ==> 3
```

Programmable Programming Language

Common Lisp follows the philosophy that what's good for the language's designer is good for the language's users. Thus, when you're programming in Common Lisp, you almost never find yourself wishing the language supported some feature that would make your program easier to write, because, as you'll see throughout this book, you can just add the feature yourself.

GEIST

Department of Automatics

AGH

GEIST

Department of Automatics

Programmable Programming Language

For instance, the original implementation of the Common Lisp Object System (CLOS), Common Lisp's powerful object system, was as a library written in portable Common Lisp. This allowed Lisp programmers to gain actual experience with the facilities it provided before it was officially incorporated into the language.
—Practical Common Lisp, Peter Seibel

Useful Macros

```
(dolist (x ’(a b c))
  (print x))
(dolist (var list &optional result)
  &body body)
```

DOLIST is similar to Perl’s foreach or Python’s for. Java added a similar kind of loop construct with the "enhanced" for loop in Java 1.5, as part of JSR-201. Notice what a difference macros make. A Lisp programmer who notices a common pattern in their code can write a macro to give themselves a source-level abstraction of that pattern. A Java programmer who notices the same pattern has to convince Sun that this particular abstraction is worth adding to the language. Then Sun has to publish a JSR and convene an industry-wide "expert group" to hash everything out. That process–according to Sun–takes an average of 18 months. After that, the compiler writers all have to go upgrade their compilers to support the new feature. And even once the Java programmer’s favorite compiler supports the new version of Java, they probably still can’t use the new feature until they’re allowed to break source compatibility with older versions of Java. So an annoyance that Common Lisp programmers can resolve for themselves within five minutes plagues Java programmers for years.

AGH

Basic Control Structures

```
(if (spam-p current-message)
    (progn
      (file-in-spam-folder current-message)
      (update-spam-database current-message))

(defmacro when (condition &rest body)
  ‘(if ,condition (progn ,@body)))

(when (spam-p current-message)
  (file-in-spam-folder current-message)
  (update-spam-database current-message))

(defmacro unless (condition &rest body)
  ‘(if (not ,condition) (progn ,@body)))
```

GEIST

Department of Automatics

DOLIST Macro

```
(macroexpand-1
  '(dolist (x '(a b c) ) (print x)))

(DO* ((#1=#:LIST-3526 '(A B C) (CDR #1#))
      (X NIL))
     ((ENDP #1#) NIL) (DECLARE (LIST #1#))
  (SETQ X (CAR #1#)) (PRINT X))

(do* (variable-definition*)
     (end-test-form result-form*)
  statement*)
```

AGH

GEIST

Department
of Automatics

Loop Macro

```
(loop for x in ’(a b c d e)
      for y in ’(1 2 3 4 5)
      collect (list x y) )

(loop for x in ’(a b c d e 1 2 3 4)
      until (numberp x)
      do (print x)
      collect (list x ’foo))

(let ((s "alpha45"))
  (loop for i from 0 below (length s)
     for ch =  (char s i)
     when (find ch "0123456789" :test #’eql)
     return ch) )
```

AGH

GEIST

Department of Automatics

Until Macro

```
(defmacro until (test &body body)
  (let ((start-tag (gensym "START"))
        (end-tag   (gensym "END")))
    ‘(tagbody ,start-tag
              (when ,test (go ,end-tag))
              (progn ,@body)
              (go ,start-tag)
              ,end-tag)))
```

GEIST

Department
of Automatics

Collectors Macro

```
(with-collectors (collectors &body))

(with-collectors (a b)
(dotimes (i 10)
(if (oddp i)
  (a i)
(b i))))
  => (1 3 5 7 9) (0 2 4 6 8)
```

Resources Macro

```
(defmacro with-resource ((name) &body body)
  ‘(let ((,name (allocator-for-resource)))
  (unwind-protect
    (progn
     ,@body)
  (deallocate-my-scarce-resource r))))

(with-resource (r)
  (foo)
  (bar))
```

GEIST

Department
of Automatics

AGH

GEIST

Department of Automatics

Dialects

- Common Lisp
  - rich, multi-paradigm version
  - CLOS is an object system that supports multimethods and method combinations
- Scheme
  - much smaller and more functional
  - focus on usefulness as teaching language
- Elisp
  - outdated as a language
  - but probably most practically useful

AGH

GEIST

Department of Automatics

Take-Away Message

- Programming languages *are* different
  - but which one is better depends on skills of people who are supposed to use them
- Python *is* useful for doing real work
  - honestly, it is *a lot* better than Java in virtually every respect
- Lisp *is* a way to improve your programming skills
  - limited practical use, but it is a different way of looking at computational problems
- Dozens of cool languages appear every year
  - keep expanding your horizons
  - remember about COBOL & FORTRAN

AGH

Learn More

- Python
  - htpp://www.python.org
  - Tutorial
  - Library Reference
  - Google
- Lisp
  - *Practical Common Lisp* by Peter Seibel http://www.gigamonkeys.com/book/
  - *Structure and Interpretation of Computer Programs* by Harold Abelson, Gerald Jay Sussman and Julie Sussman

http://mitpress.mit.edu/sicp/full-text/book/book.html

GEIST

Department of Automatics

AGH

GEIST

Department of Automatics

Questions?