AGH

CSLAB

Katedra
Automatyki

Introduction to Python and Lisp

Sławomir Nowaczyk

Laboratorium Informatyki
Katedra Autormatyki
Akademia Górniczo-Hutnicza

January 21, 2009

AGH

CSLAB

Katedra
Automatyki

Languages of AI

- Special purpose languages
  - PROLOG
  - Planning Domain Definition Language PDDL
  - Bertrand — Constraint Programming
  - ...
- General purpose languages
  - exploratory programming
  - mutable specifications
  - tricky problems
  - limited system integration
  - expert users

Simple Calculator by AGH Students

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

AGH

CSLAB

Katedra
Automatyki

Simple Calculator by AGH Students

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

AGH

CSLAB

Katedra
Automatyki

Simple Calculator by AGH Students

```
if(x.equals("/"))
{
  b = Integer.valueOf(stos.removeLast());
  a = Integer.valueOf(stos.removeLast());
  if (b==0) { return DivisionByZero; }
  result = a/b;
  stos.addLast(String.valueOf(result));
  updateDisplay();
  System.out.println(String.valueOf(a) +
    "/" + String.valueOf(b) + "=" +
    String.valueOf(result));
}
```

AGH

CSLAB

Katedra
Automatyki

AGH

CSLAB

Katedra
Automatyki

Simple Calculator in JAVA

```
interface calc { int do(int a, int b); }
class plus implements calc {
  int do(int a, int b) { return a+b; } }

if(x.equals("+")) {
  doWork(new plus(),"+",false);
}
if(x.equals("-")) {
  doWork(new minus(),"-",false);
}
if(x.equals("/")) {
  doWork(new divide(),"/",true);
}
```

AGH

CSLAB

Katedra
Automatyki

Powerful Language

- Expressive abstraction mechanisms
  - group *similar* concepts together
- Ways to avoid code duplication
  - specify subtle *differences*
- 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?

AGH

CSLAB

Katedra
Automatyki

C versus Python versus Lisp

- 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
- Code readability
- Dynamism
- Compiled to a byte code
- Interpreted by virtual machine
- Garbage collection
- Extremely powerful

AGH

Basic Python Features

- Interactive “shell”
- Rich container types
- Functions, classes, modules
- Namespaces
- Exceptions
- Portable
- Powerful introspection
- Plenty of dynamic features
- Extremely rich standard library

CSLAB

Katedra
Automatyki

AGH

CSLAB

Katedra
Automatyki

Python Applications

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

AGH

CSLAB

Katedra
Automatyki

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!

AGH

CSLAB

Katedra
Automatyki

Python Design

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

AGH

CSLAB

Katedra
Automatyki

Basic Python Features

- Numeric types: long, complex & rational
- Both “normal” and unicode strings
  - immutable
- Lists as essential data type
  - flexible arrays (*not* linked lists)
  - indexing, slices, basic operations
- Dictionaries as fundamental data type
  - hash tables with immutable keys
- “Batteries-included” approach to standard library
  - one of Python’s greatest strengths
  - makes Python a powerful *glue language*

AGH

CSLAB

Katedra
Automatyki

Some More Exotic Features

- Good support for iterators
  - `for line in file("fname"):`
- Resource management

```
with open("/etc/passwd", "r") as File:
    raise Exception
```

- Generators
  -
```
def Fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b
```
- Multi-paradigm programming language

AGH

CSLAB

Katedra
Automatyki

Basic 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
```

AGH

CSLAB

Katedra
Automatyki

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])
```

AGH

CSLAB

Katedra
Automatyki

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

AGH

CSLAB

Katedra
Automatyki

Some 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)
```

AGH

CSLAB

Katedra
Automatyki

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)
```

AGH

CSLAB

Katedra
Automatyki

Metaclasses

- Very powerful mechanism
  - not for 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

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

CSLAB

Katedra
Automatyki

AGH

CSLAB

Katedra
Automatyki

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

CSLAB

Katedra
Automatyki

Lisp

- LISt Processing language
  - primary data structure is a linked list
  - `(elem1 elem2 elem3 elem4)`
  - source is organised in this fashion as well
  - 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*

AGH

CSLAB

Katedra
Automatyki

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

AGH

CSLAB

Katedra
Automatyki

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.

AGH

CSLAB

Katedra
Automatyki

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 can construct and execute other programs on the fly

AGH

CSLAB

Katedra Automatyki

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

AGH

CSLAB

Katedra
Automatyki

Lisp Design

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

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

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

AGH

CSLAB

Katedra
Automatyki

Lisp Basics

```
(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

CSLAB

Katedra
Automatyki

Lisp Basics, part 2

```
(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)))))
```

AGH

CSLAB

Katedra
Automatyki

Lisp Basics, part 3

```
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#))
```

AGH

CSLAB

Katedra Automatyki

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

CSLAB

Katedra Automatyki

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

CSLAB

Katedra Automatyki

Macros Explained

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

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

AGH

CSLAB

Katedra
Automatyki

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.

AGH

CSLAB

Katedra
Automatyki

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

AGH

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.

CSLAB

Katedra Automatyki

AGH

CSLAB

Katedra
Automatyki

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)))
```

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

CSLAB

Katedra
Automatyki

AGH

CSLAB

Katedra
Automatyki

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

CSLAB

Katedra
Automatyki

Testing Framework

- Simple unit testing framework
  - Organise, run and report tests
  - Not full-fledged, but with some cool features
- Incremental development
  - simplifying code
  - adding functionality
- 26 lines of code

AGH

CSLAB

Katedra
Automatyki

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

CSLAB

Katedra
Automatyki

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

CSLAB

Katedra
Automatyki

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

AGH

CSLAB

Katedra
Automatyki

Questions?