

Advanced Python (for Biologists)

Paul-Michael Agapow

Institute of Animal Health / VieDigitale Ltd

pma@viedigitale.com / paul-michael.agapow@bbsrc.ac.uk

Aims

- To be able to program what you need to program
- Pragmatic
- Not to know, but to know where to look
- Just enough theory
- Jargon

Contents

- Python
- Software engineering
- BioPython
- Other Python libraries

Python

- Revision
- I/O
- Modules
- Installation
- Iterators
- Strings & string manipulation
- Objects

Software engineering

- Design
- Debugging
- Defensive programming
- Structured data (XML etc.)
- Documentation

BioPython

- Sequences & SeqRecords
- Clustering
- Alignment
- Database access

Other Python libraries

- Matplotlib & basemap
- SciPy
- Numerical computing & NumPy
- Others?

Assessment

- 15%
- Daily worksheets (save as transcripts, or source)
- Projects (suggestions?)

Today

- Theory: programming languages, translation & jargon
- Calling Python
- IPython
- Where you're at

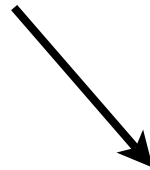
What is programming?

(What is computation?)

- *(stupid physics analogy)*
- Not binary, not maths
- Symbol manipulation
- A problem of translation

Different languages

print "Hello, World"



```
; This program displays "Hello, World!" dosseg
model small
.stack 100h .data
hello_message db 'Hello, World!',0dh,0ah,'$' .code
main proc
mov ax,@data
mov ds,ax mov ah,9
mov dx,offset hello_message
int 21h mov ax,4C00h
int 21h
main endp
end main
```



```
100101010010101001011101010110
10010100100101010010010110011110
01100101001111010111010101001010
11101110111010101001010001000100
10100100100011001001100101010111
01101010100101010111100001010100
00110101000101010001100110011001
10010101001010100101110101011010
10010010101001001011001111001100
10011110101110101010010101110111
```

Levels of programming language

- Low Level (C, C++, Pascal, Ada, Modula-2)
“programs”, translated (compiled) once before use, fine detail, slow to write but fast to run
- High Level (Python, Perl, Lisp, Tcl)
“scripts”, translated (interpreted) every time when run, coarse detail, fast to write but slow to run

How / when is Python called?

- On the commandline
- As a result of script being executed
- The interactive prompt

On the commandline

- `home> python extract_concensus.py`
- `home> python23 extract_concensus.py`
- `home> /usr/bin/local/python23
extract_concensus.py`
- `home> /usr/bin/local/python23
projects/biopy/
extract_concensus.py`
- `-V` for version

What is `__main__`?

- The script called defines its name as `__main__`
- Can use to make script behave two ways

```
if __name__ == "__main__":  
    print "Running as main"  
else:  
    print "Not main, probably  
imported"
```

Commandline arguments

```
if __name__ == "__main__":  
    import sys  
    for item in sys.argv:  
        print "Argument:", item
```

```
# sys.argv contains 4 arguments  
python checkargs.py inseq.fasta true strip_spaces=true
```

```
# sys.argv contains 6 arguments  
python checkargs.py inseq.fasta true strip_spaces = true
```

Raw code on the commandline

- Pass code straight to Python
- Delimit statements with ;
- Calculator?

```
home> python -c "x=3; y=4; print x*y"  
12
```

Execute scripts directly

- (In Unix) make file executable with

```
chmod +x myscript.py
```

- First line of script is

```
#!/usr/bin/env python
```

(or `#!/python etc.`)

- Just like calling `python myscript.py`
etc.

The interactive prompt

- Call with `python`
- Can write and manipulate code on the fly (advantage of interpreted language)
- `python -i myscript.py` runs the script and then dumps you into the prompt
- Awkward (IPython!)

IPython

- An enhanced Python prompt
- Change working (current) directories and list contents from within Python using Unix commands `cd` and `ls`.

```
Documents/Projects/MyPython/Casimir> python
>>> import xmltools
ImportError: no module named xmltools
>>> import Documents/Projects/MyPython/xmltools
SyntaxError: invalid syntax
>>> import "Documents/Projects/MyPython/xmltools"
SyntaxError: invalid syntax
```

Tab completion

- Prompts with list of matching objects
- Includes contents of objects
- Only works on loaded objects (of course)

```
In [1]: o<TAB>
or      oct      ord      open      object
In [2]: import os
In [3]: o<TAB>
or      oct      ord      open      object      os
In [2]: os.ch<TAB>
os.chdir  os.chmod
```

Introspection

- Introspection
- Run programs
- Confirmed quit
- Much, much more

```
In [2]: import string
In [2]: ?string
Type:          module
Base Class:    <type 'module'>
String Form:   <module 'string' from 'C:\Python24\lib\string.pyc'>
Namespace:    Interactive
File:         c:\python24\lib\string.py
Docstring:
    A collection of string operations
    (most are no longer used).
    [...]
```