

# You Need To Know Python

Russel Winder

email: [russel@russel.org.uk](mailto:russel@russel.org.uk)

xmpp: [russel@russel.org.uk](xmpp:russel@russel.org.uk)

twitter: [russel\\_winder](https://twitter.com/russel_winder)

# *Aims, Goals and Objects*

- Investigate the factors that make Python such a cool language.
- Show that people programming in any other language benefit from knowing Python.
- *Arrive at pub in good time to have a drink and good conversation – and then some food.*

# *Structure*

- A beginning.
- A middle.
- The other middle.
- An end.

- 
- *Networking*

# *Protocol*

- Questions or short comments during the session are entirely in order.
- Let me know you have an interjection by raising your hand, and when I come to an appropriate pause, I'll pass you the token.

*Questions, answers and comments need to be of appropriate length.*

# A Beginning

**I need to know Python**

*This is an “In the Brain”  
session after all.*



# *A Bit of Language History (of RW)*

- FORTRAN

---

  - More FORTRAN

---

  - Further FORTRAN
  - *JCL and CLists*
  - Algol 68
  - Pascal

---

  - C
  - Sh, Csh, Bash

---

  - FORTRAN77
  - C++
- Python*
- Scheme
  - Miranda
  - Java
  - Haskell

---

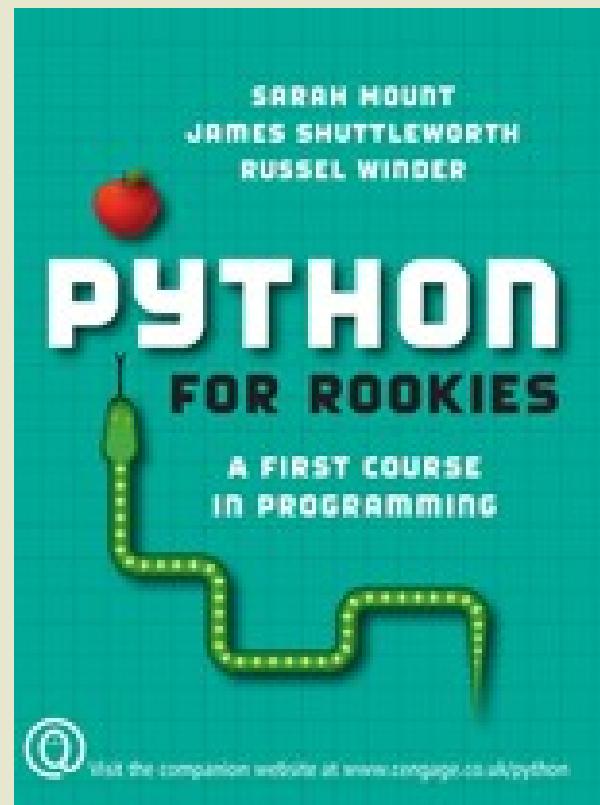
  - Fortran
  - ***Python***

---

  - Groovy
  - Ruby
  - Scala
  - Fantom
  - Clojure
  - Go
- Solve
  - UC++
  - EC++ aka | | C++
  - KC++
  - ***GPars***
  - ***Python-CSP***
  - ***Python-Concurrent***

# *They Say that . . .*

- . . . the fastest way to learn something is to write a book about it:



“They” are definitely wrong, to write about something you need to know about it.

# *Psychology of Programming*

Programmers who can program at least reasonably in multiple languages, where these involve multiple computational paradigms, are better programmers.

# *Python is . . .*

- . . . a multi-paradigm language, and so . . .
- . . . knowing Python contributes to programming expertise. Hence:

***You need to know Python***

Not entirely convincing, as Python isn't really as multi-paradigm as people sometimes pretend.

# A Middle

You *need* to know Python

# *The Paradigms of Programming*

- Imperative
  - Procedural
  - Object-oriented
- Declarative
  - Functional
  - Logic

# *The Paradigm Wars*

- 1950s–1980s: Structured vs. Spaghetti – goto.
- 1980s–1990s: Functional vs. Object-oriented.
- 1990s–present: Static vs. Dynamic.
- 2000s-present: Functional vs. Object-oriented.

# *The Language Wars*

- Early period:
  - FORTRAN, COBOL, Lisp
- Late-Early period:
  - Algol, Pascal
  - BCPL, C
- Early-Middle:
  - Smalltalk, C++
- Middle-Middle:
  - Java, Haskell, Lisp
- Late-Middle period:
  - Python, Ruby
  - Groovy, Clojure
  - Scala
  - D, C++ (resurgent?)
  - Go

# *And the Winner Was . . .*

- C++ appeared to win the language wars in the 1990s for, and on behalf of, object orientation.
  - And yet, C++ isn't really an object-oriented language.
  - Object orientation is about objects passing messages to each other:
    - C++ sort of does this but requires explicit programmer doublethink.
    - Shared memory multi-threads is no substitute for active objects passing messages.

## *And then . . .*

- . . . Java stole the crown of C++ . . .
- . . . and the wars continued unabated.
  - And Java isn't really object-oriented either – far too much shared memory multi-threading, and nowhere near enough message passing between active objects.

# *VM Wars*

- Should software be virtual machine based or native?
- Context:
  - Client or server.
  - Platform portability.
- Layers of virtual machines:
  - Language: JVM, **PVM**, Parrot, . . .
  - Clusters: PVM, MPI, . . .
  - Platforms: VMWare, KVM, VirtualBox, . . .

# *Cloud Wars*

Forget it, its all just too *nebulous*.



Who knows what will happen regarding programming languages in the Modern Period?

No-one.

We'll find out when it happens.  
In the mean time . . .

# *The Hardware Effect*

- Moore's Law:
  - Transistor count per processor chip doubles every 18–24 months.
- Side issue:
  - Processor speed doubles ever 18–24 months.

# *The Multicore Revolution*

- Processor speeds stop increasing – cannot get rid of the heat.
- Moore's Law still in effect.
- Increase core count.

# *Hardware is Going Faster*

If there are more cores in a processor then there are more instructions executed per unit time so your programs run faster.

# *Parallelism is the Norm*

Increased performance now requires  
parallelism.

***CPython is . . .***

***. . . a troubled system.***

# *The FP Imperative*

- Functional programmers have been claiming superiority of their computational model since they lost the language war.
- Functional languages are reasserting themselves. cf. Scala, Clojure.
- Not to mention that Groovy, C++, D, Go are all being used with increasing functional idioms.

# *Functional brings . . .*

- An enforcement of ideas of immutability and a lack of sharing.
- An arcane view of state.
- Monads.

# *And About Python?*

- Python is a language that:
  - was a procedural language;
  - became an object-oriented language;
  - sort of has functional language features; and
  - ignores logic programming.
- Claims to be multi-paradigm, but isn't really.
- . . . and yet . . .

# *Functional Stuff in Python*

- Higher-order functions: map, filter, reduce, list comprehensions.
- Decorators are higher-order functions:
- So we need all the functional programming idioms for using decorators and other higher-order functions.

Fibonacci

# *Functional Python*

A Monads library for Python à la  
Functional Java, Scalaz?

# Why?

Imperative language don't *need* monads,  
but they might use them anyway.

# *You Need to Know Python?*

- Yes:
  - It is one of the mainstream languages.
  - It has elements of multi-paradigm.
  - It can interwork with C, C++, Fortran.
  - It is a dynamically typed language and yet can inform how we use statically typed languages.
- *Working with statically type languages informs us how to use dynamically typed languages.*

# Interlude

# *You are a C Programmer*

- How do you unit test your code?
  - Check
  - Python with *ctypes*
- How do you system test your code?
  - Python

# The Other Middle

You need to *know* Python

# *Superficiality, A Definition*

Take an introductory course on a language



Know how to program in the language

# *Sophistication, A Definition*

Take an introductory course on a language



Plan taking an intermediate course

# *Python is Unique (?)*

- Indenting determines structure.
- Variables are entries in dictionaries.
- The parameter passing system is complicated, arcane, weird and yet so very simple and hugely efficacious
- Python has a meta-object protocol (MOP).
- Context managers.
- No parallelism.

# *Indentation*

- You have to love it *and* hate it:
  - Love:
    - Code is forcibly laid out readably.
    - Code can be made to be an art form.
  - Hate:
    - Enforcing leading space for block structuring really makes writing domain specific languages (DSLs) much harder than it should be.

# *Variables*

- Variables are not labelled boxes.
- Variables are entries in dictionaries.

# *Parameter Passing*

- Actuals:
  - Positional.
  - Defaulted positional.
  - Keyword arguments.
- Formals:
  - Positional
  - Defaulted positional
  - Other positional
  - Other keyword

# *Parameter Passing – Downside*

It is actually quite complicated

# *Parameter Passing – Upside*

- Makes working with APIs that have a penchant for large numbers of parameters very easy:
  - GTK
  - Qt
  - WxWidgets
- Becomes easy when you get used to it.
- Means absolutely no need for overloading – which is good as you can't have it in Python anyway.

*Python make sense when you **know** Python.*

# ***MOPing It Up***

- Meta-object protocol (MOP):
  - Function call is a run-time mediated activity.
  - Core to the whole dynamic language idea:
    - Lisp
    - Groovy
    - Ruby
    - ***Python***

# *Callables*

- Python MOP is about *callables*:
  - Functions.
  - Instances with `__call__` method.

# *Context Managers*

- `with` statement.
- `__entry__`, `__exit__` callables.
- RAI – resource acquisition is initialization – the greatest contribution C++ has made to software development, now available in Python.

# Builders

- (Re-)Popularized by Groovy, picked up by Ruby, now available in Python.

html.py

```
from html import XML

if __name__ == '__main__':
    x = XML ()
    with x.book as b :
        b.title ( 'Python for Rookies' ) ,
        for name in ( 'Sarah Mount' , 'James Shuttleworth' , 'Russel Winder' ) :
            b.name ( name )
        b.publisher ( 'Thomson Learning' ) ,
        b.date ( '2008' )
    print x
```

code.py

# DSLs Rock

# *The GIL*

- CPython has a global interpreter lock:
  - Only one thread can execute Python bytecode at any one time.
  - C, C++, Fortran coded extensions can release the GIL, and reclaim it when returning to run Python bytcodes.

# ***GIL and Parallelism***

**Mutually incompatible.**

# *Python and Parallelism*

- Use the *multiprocessing* package:
  - Very basic.
  - PVM per “object”.
  - Can build Actor Model, Dataflow Model, Communicating Sequential Processes (CSP) on it, but . . .
  - . . . it is intended to be a drop in replacement for the *threading* package and so is not really suitable as infrastructure for these more suitable models of parallelism.

# An Interlude

# *Python Use (Slowly) Evolves*

- Python 2 becomes Python 3.
- But everyone continues to have to work with Python 2 because that is what is there.
- We need a more concerted effort to use Python 3 and allow Python 2 to fade into history.

An End

# *You Need to Know Python*

- Yes:
  - Python is a mainstream dynamically-typed language with much to say about idioms of use.
  - Python's unique aspects teach.
  - C, C++, Fortran programmers *really* need to know Python.

# *You Need to Know Python*

- Yes:
  - To use Python reasonably you have to know it to more than a superficial level.
  - Dynamic languages require:
    - More unit and system testing.
    - More sophisticated understanding.
    - An understanding of the meta-object protocol.
  - Parallelism requires understanding of the effect of CPython's GIL and how to deal with it.

The End

# You Need To Know Python

Russel Winder

email: [russel@russel.org.uk](mailto:russel@russel.org.uk)  
xmpp: [russel@russel.org.uk](xmpp:russel@russel.org.uk)  
twitter: [russel\\_winder](https://twitter.com/russel_winder)