Rapid development in Python

presented by

Daniel J Blueman

Computer Science

Bristol University

Introduction

- today's aims
  - understand what Python brings
  - caveats/costs
- not on the menu
  - learn Python programming
- three parts with breaks

Part 1

- background
- benefits
- examples

What is it?

- newer but mature - 1991
- rapid development
- integrating experience
- emphasis on simplicity/readability
    - pseudocode
- extensively used: where?

Disadvantages

- memory footprint/speed trade
- ~~'native' library binding~~

Benefits

- Truly platform agnostic
  - write once, run anywhere
- JIT compilation
- Still being enhanced
- Built in types
  - Complex numbers
- Extensive, 'handy' standard libraries

Benefits 2

- ”batteries included” approach
  - numerical, scientific, visualisation, opengl, XML, ...
- ”best of breed”
  - OO, exception handling, named arguments, ...
- built in types actually classes
  - custom types
- ref counting mem management

Where is it used?

- Widely used
  - Youtube, Maya, ...
- MacOS X, Linux, BSD Unix

Simple example

```
#!/usr/bin/python

import os

for file in os.listdir("."):
   print "found %s" % (file)
```

Output:

```
$ ./example1.py
found graph1.py
found slider_demo.py
found histogram_demo.py
...
```

Dictionaries

- person = { ‘name’: “Robin Hood”, age: 42}
- person[‘occupation’] = “Scoundrel”

Object orientation

- core concept
- paradigm
- divide and conquer strategy
- humanistic

How is this used?

- Chess example

board object

properties

(none)

methods

create

destroy

setCell

getCell

clearCell

move

piece object

properties

colour

type

methods

create

destroy

getColour

getType

...classes!

Chess example 1.1

-
  ```
  class Piece:
      def __init__(self)
          self.pos = 0
      def getType()
      def getColour()
  ```

-
  ```
  class Board:
      def __init__(self)
      def set(self, x, y, t)
      def get(x, y)
      def move(x1, y1, x2, y2)
  ```

Chess example 1.2

-
  ```
  #!/usr/bin/python

  import chess

  Board b
  b = Piece(Piece.black, Piece.rook)
  b.setCell(1, 7, p)
  ...
  b.move(1, 7, 2, 7)
  ...
  ```

What does all this buy?

- ease of code management
- design mapping
- fewer bugs
- less maintenance

Questions

- you know you have them...

Part 2

- advanced concepts
- more examples

Library interfacing

- 'ctypes' modules (v2.5)

```
import ctypes

libc = ctypes.CDLL("libc.so.6")
print libc.strlen("Hello world!")
print libc.time(None)
```

- Demonstration!
  - including interactive shell

Complex library interfacing

- `struct passwd getpwnam(const char *login);`

-
  ```
  struct passwd {
          char    *pw_name;       /* user name */
  ...
  ```

-
  ```
  >>> class PASSWD(ctypes.Structure):
      _fields_ = [("name", ctypes.c_char_p),
  ...
  ```

-
  ```
  >>> libc.getpwnam.restype = ctypes.POINTER(PASSWD)
  >>> libc.getpwnam("daniel")
  >>> entry = libc.getpwnam("daniel")[0]
  >>> entry.uid, entry,gid
  (1500, 100)
  ```

Calling Python from C

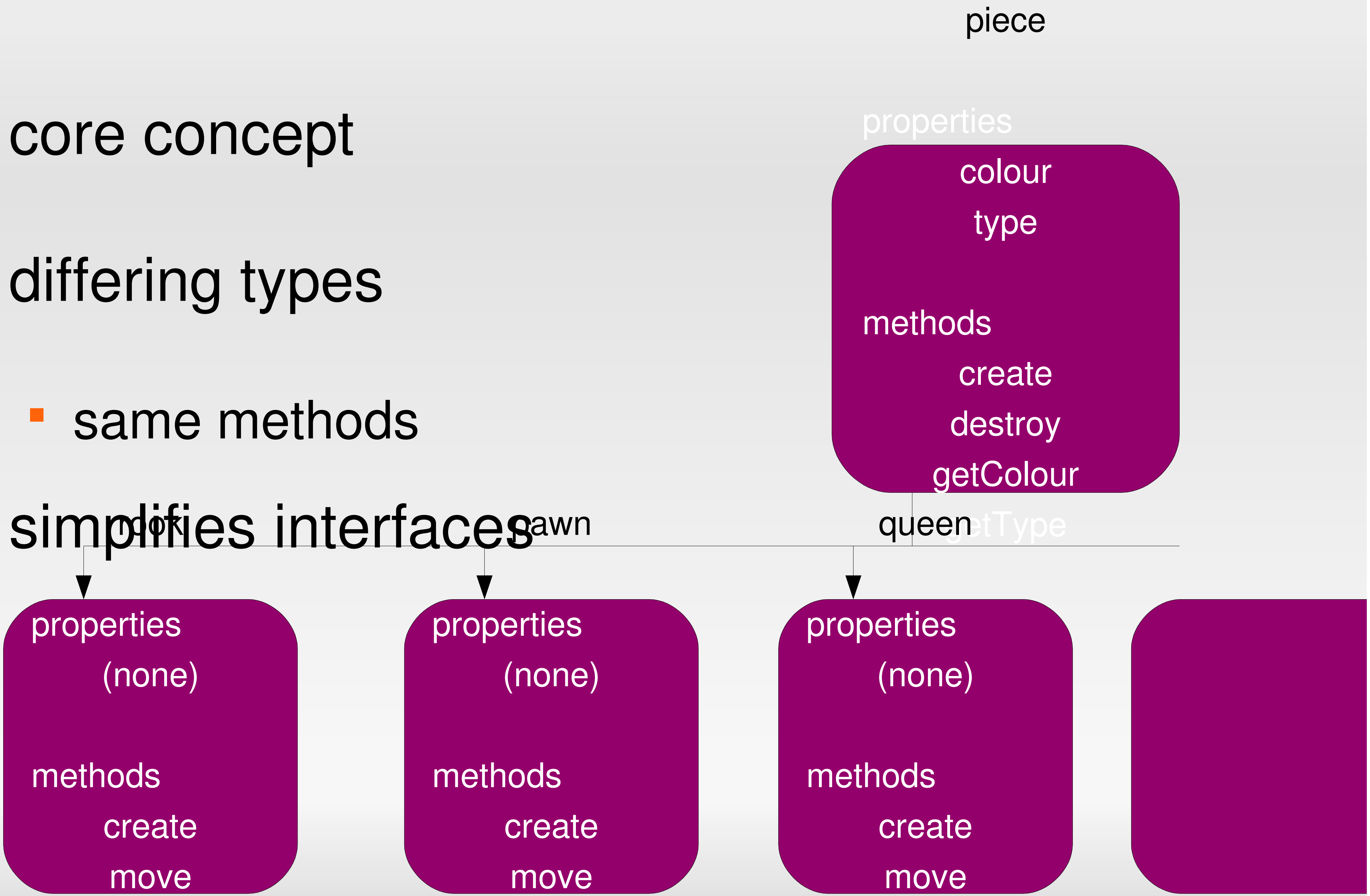
- at function-level
- library to create libraries
- covered elsewhere

Inheritance

- core concept
- base class
- derived class
- transfer of attributes, methods

Polymorphism

- core concept
- differing types
  - same methods
- simplifies interfaces



piece

properties
colour
type

methods
create
destroy
getColour
getType

rook

properties
(none)

methods
create
move

pawn

properties
(none)

methods
create
move

queen

properties
(none)

methods
create
move

Chess example 2.1

-
  ```
  class Piece:
      def __init__(self, t, c)
      def getType(self)
      def getColour(self)

      x, y
      colour
      type
  ```

-
  ```
  class Rook(Piece):
      Rook(self, c)
  ```

Chess example 2.2

-
  ```
  #!/usr/bin/python

  import chess

  Board b
  Piece p = Rook(Piece.black)
  b.setCell(1, 7, p)
  ...
  b.move(1, 7, 2, 7)
  ...
  ```

(Brain) Overloading

- core concept
- method and operator
- simplifies at one level
  - a = b + c
- complicates at another
  - what is a, b and c?
- init

Questions

- there must be one...

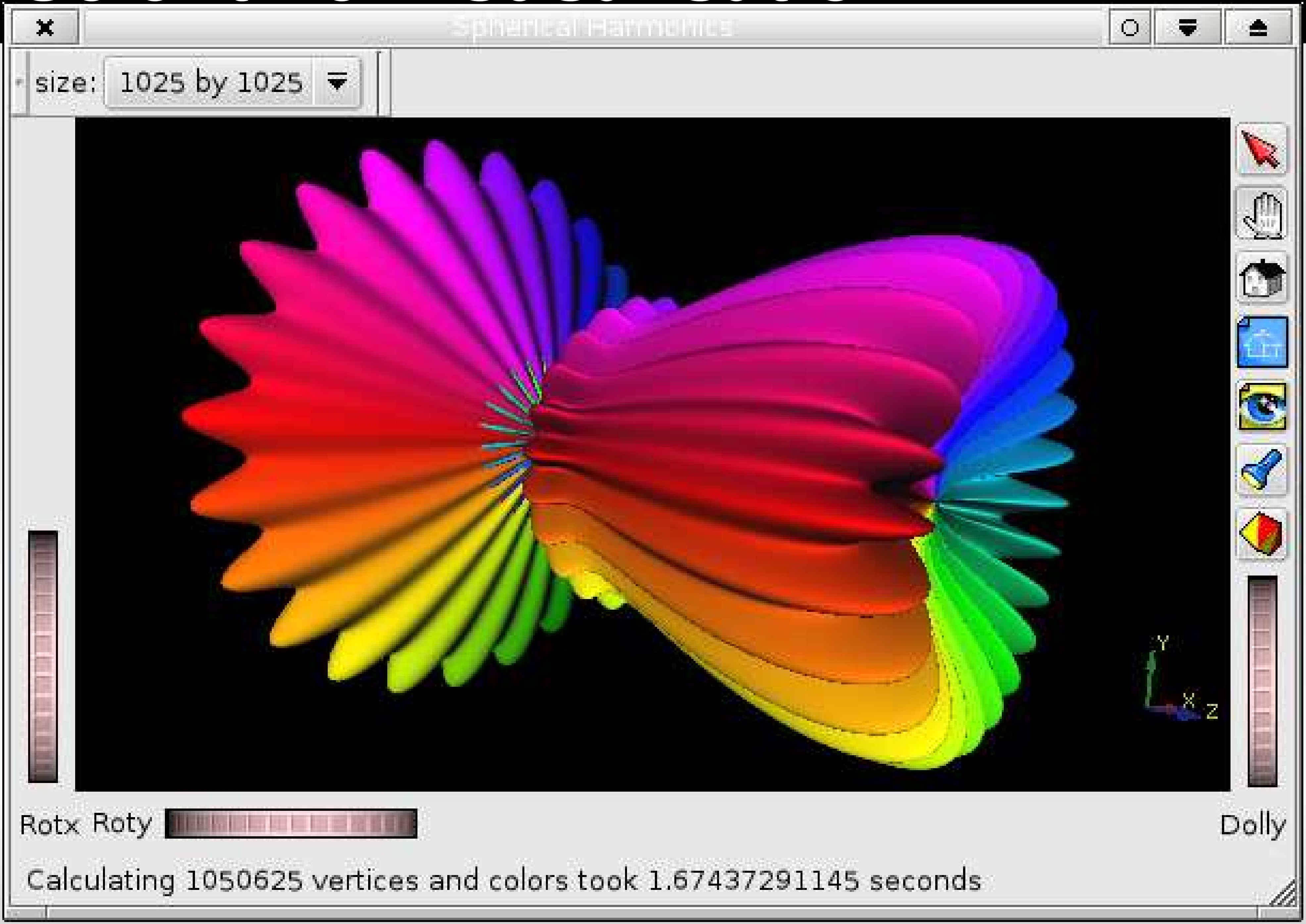
Part 3

- Fun with examples
- Programming challenge
- Questions

Scientific Visusalisation

- examples speak louder than words
  - graph1.py
  - histogram_demo.py
  - slider_demo.py
  - surface.py
  - 3d.py

Scientific Visualisation 2



size: 1025 by 1025

Calculating 1050625 vertices and colors took 1.67437291145 seconds

Applications

- examples
  - hello.py
  - testapp_ui.py (XML)

Challenge

- bubble sort program

```
#!/usr/bin/python

def bubble(list):
   # your code to go here
   return list

values = [715, 1135, 1367, 13, 17, 5135, 124, 72, 125,
63, 71, 76124, -61, 17]

result = bubble(values)
print result
```

- wget http://quora.org/bubble.py

  chmod 755 bubble.py

  ./bubble.py

Part 3: Challenge

- bubble sort program

```
#!/usr/bin/python

def bubble(list):
   # your code to go here
   for passes in range(len(list) - 1, 0, -1):
      for i in range(passes):
         if list[i] > list[i + 1]:
            # transpose elements
            list[i], list[i + 1] = list[i + 1], list[i]

   return list

values = [715, 1135, 1367, 13, 17, 5135, 124, 72, 125,
63, 71, 76124, -61, 17]

result = bubble(values)
print result
```

Comparison

- damn good for dealing with data
- rapid devel, fewer bugs
- C/C++ for *hackers*
- matlab limits
- Is it for me?

Thanks

- last chance for questions...
- contact: daniel.blueman@gmail.com
- presentation: http://quora.org/python.pdf