Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications 177 (2007) 199–201
Computer Physics Communications
www.elsevier.com/locate/cpc

Python: a language for computational physics

P.H. Borcherds

School of Physics and Astronomy, University of Birmingham, England

Available online 12 February 2007

Abstract

Python is a relatively new computing language, created by Guido van Rossum [A.S. Tanenbaum, R. van Renesse, H. van Staveren, G.J. Sharp, S.J. Mullender, A.J. Jansen, G. van Rossum, Experiences with the Amoeba distributed operating system, Communications of the ACM 33 (1990) 46–63; also on-line at http://www.cs.vu.nl/pub/amoeba/. [6]], which is particularly suitable for teaching a course in computational physics. There are two questions to be considered: (i) For whom is the course intended? (ii) What are the criteria for a suitable language, and why choose Python? The criteria include the nature of the application. High performance computing requires a compiled language, e.g., FORTRAN. For some applications a computer algebra, e.g., Maple, is appropriate. For teaching, and for program development, an interpreted language has considerable advantages: Python appears particularly suitable. Python's attractions include (i) its system of modules which makes it easy to extend, (ii) its excellent graphics (VPython module), (iii) its excellent on line documentation, (iv) it is free and can be downloaded from the web. Python and VPython will be described briefly, and some programs demonstrated.

Keywords: Education; Graphics; Computational physics; Computation; Python

1. Introduction

This paper discusses why Python is a suitable language for the teaching of computational physics. It provides an overview of Python, but it is not an introduction to programming Python. Python is an object-oriented language, but that will not be explicitly discussed. There is no discussion of other computing languages, but see Donaldson: Python as a First Programming Language for Everyone [3].

Raw Python, that is Python by itself, is a language with limited capabilities, but it can be extended by importing one or more of the many available modules. VPython, which has been developed by Bruce Sherwood, Ruth Chabay, David Scherer and colleagues at the University of North Carolina, is a module particularly suitable for teaching computational physics [2].

Current versions of Python and of VPython can be downloaded from the web, and each is free. Each comes with excellent on-line documentation, including a tutorial. VPython comes with a folder of examples: of which ColorSliders and Stonehenge are particularly attractive and show different features of VPython.

Another example in the folder is Bounce.py, which is listed in Table 1 to show the structure of a Python program. Its first statement is `from visual import *` which makes available the visual module (VPython). Note that there are no end statements. In a block statement, the first line of the block ends with a colon (:) and subsequent lines are indented by one tab space. The end of the block is indicated by the end of the indentation.

`ball` and `floor` are visual objects, with properties such as `ball.velocity`.

The program shows a ball bouncing, but when it runs it has an irritating feature: the scale of the picture alters with the height of the ball: there is nothing explicit in the program to determine the scale. The VPython autoscale default is that the scale is chosen to match the size of the whole picture to the size of the window.

When writing a program with a graphical output, a programmer must pay attention to scale factors, which can be tedious, since it is necessary to know the screen resolution, etc., and how the computer program interacts with the display. Having a varying scale factor, as in this example is a minor irritation, particularly since it is easily overcome, by inserting `scene.autoscale = False` immediately before the `while` statement.

E-mail address: p.h.borcherds@bham.ac.uk.

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2007.02.019

Table 1
bounce.py

```
#bounce.py
from visual import *          # VPython
floor = box(length=4, height=0.5, width=4,
color=color.blue)
ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity =vector(0,-1,0)
dt = 0.01
while True:
    rate(100)
    ball.pos += ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y *= (-1)
    else:
        ball.velocity.y -= 9.8*dt
```

2. For whom?

All physics graduates should have some knowledge of a high level computing language and of computational physics, of what it can do, and of its limitations, as well as its strengths [1]. An introductory course in computational physics has two aspects:

- an introduction to numerical methods, e.g., there is more to solving a differential equation than the Euler method;
- an introduction to computer programming, e.g., input and output, conditional statements, loops and arrays. (Experience has shown students have difficulty with these.)

An introductory course needs to be made attractive to all students, not only to computer enthusiasts. The language used must be easy to run, easy to program and must have graphics.

For such a course the advantages of an interpreted language such as Python (with an immediate mode) are overwhelming: students can get results immediately, and can find and rectify errors quickly.

Python is widely used in program development by large organisations: see the Python website. An interpreted program runs much more slowly than a compiled one, but program development is faster, and some parts of a program do not need to run fast, e.g., those involving human interaction. Furthermore, most computer languages are sufficiently similar that translating from one to another is straightforward.

3. Getting started with Python/VPython

Both Python and VPython must be installed. If not already installed, find the websites (Python.org and VPython.org) and follow the instructions.

VPython has an excellent Integrated Development System [IDE] called 'IDLE' whose use is strongly recommended: it organises the formatting of the program being written. To open the IDLE Window, double click on the VPython IDLE Icon on the desktop. At the top of the IDLE Window is a menu: 'Run'. Click on it, and double click on the option 'Python Shell', opening a new window called 'Python shell' which displays the Python prompt '>>>'.

The computer is now ready for your first Python session, in which you write your first Python program, using immediate mode: the Hello World Program.

At the prompt (>>>) in the Python Shell Window, type "Hello World" (including the quotation marks) and press the return key. The computer prints out the message 'Hello World' and you have written and run your first program: it seems almost too easy.

To write a program, such as bounce.py shown above, return to the IDLE Window, and type the listing. IDLE handles the formatting automatically. The file has to be saved before it can be run, so save it with a suitable name, with the extension '.py', in the default directory (IDLE looks after that) then press F5 and the program will run, or will stop and report where it has found errors.

4. Programming aids

Python has a number of built-in aids to programming

- 'Help' on the window menu bar takes you to the Python documentation and to documentation on Visual if it has been imported. The Python documentation includes a tutorial by Guido van Rossum, the originator of Python. The VPython documentation also includes a tutorial.
- Two useful functions are dir() and help(). dir() on its own lists the currently accessible objects, while dir(object) gives the properties of the named object. help() gives further information on objects.
- There is a large amount of documentation on the web, e.g., a very useful text book (Downey, 'How to think like a computer scientist' [4]) available either free on line or as a book.
- Several textbooks are available, e.g., [5]. Many are more suitable for computer science than for computational science.

5. Programming and graphics

An introductory course in computational physics should aim to be based on familiar problems in physics, preferably with known analytic solutions. We mention two topics: (1) ordinary differential equations, (2) the solution of nonlinear equations. For each, a graphical display enhances understanding.

One of the strengths of an analytic solution of differential equations is that it can yield insight; for example, Kepler's Laws, which we discuss below; while a purely numerical solution is unlikely to yield a similar level of insight. This does not mean that we should not carry out purely arithmetic solutions, e.g., the Apollo missions depended critically on the numerical solutions of the equations of motion.

A purely arithmetic solution is unlikely to yield insight. How then can one obtain insight from a computational calculation? We may be able to obtain insight through displaying the results graphically. Our ability to recognise patterns is highly developed, and when we observe an unfamiliar pattern we examine it closely, as we can see from Figs. 1 and 2.

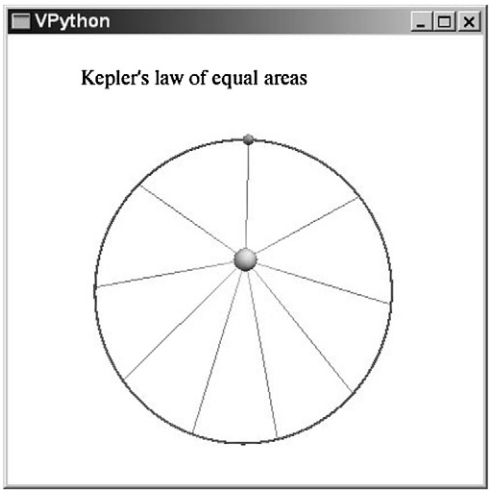

Fig. 1. Kepler's Law of Equal Areas.

Fig. 2. Newton–Raphson method: basins of attraction of cube roots of unity (real root at right edge).

In computational physics the appearance of an unexpected pattern may be nothing more than an artefact of the calculations, e.g., too large a step size in the solution of a differential equation or if the method being used is inappropriate, such as using the Euler method for the gravitational two body problem, one may obtain what appear to be very exciting results but which are of value only in warning the user how badly things can go wrong in computational science.

In the gravitational two body problem, the Euler method can be shown to be unconditionally unstable, and a fourth-order Runge–Kutta method is used: Python's ability to manipulate vectors and arrays simplifies the program writing. On looking at a graphical display of the solution of this problem, it is very satisfying to rediscover Kepler's Law of Equal Areas, as is shown in Fig. 1.

The author still remembers with pleasure his "discovery" of the law of equal areas when he was teaching himself how to make use of the graphics display on an Apple II computer. At that time the speed of computers was sufficiently slow that the development of the trajectory could be followed. On a modern computer, the speed is such that the whole trajectory may appear almost instantaneously. Python has the rate() function which permits the display to be slowed down so that one can observe the development of the system being studied: this helps one obtain better insight into its dynamics.

The solution of nonlinear equations by the Newton–Raphson method is familiar, but it holds a surprise which can be appreciated with a graphical display.

The equation $x^3 = 1$ has one real root and two complex roots. Python's ability to manipulate complex numbers simplifies program writing.

The Newton–Raphson method is iterative. If an equation has several solutions, which solution is found depends upon the initial trial value. All the initial values which lead to a particular root are said to lie in the basin of attraction of that root. This equation has three roots, and three basins of attraction, shown in Fig. 2. The basins have fractal boundaries, with the property that to get from one basin to another, you have to pass through the third basin.

6. Summary

Python is a very attractive language, particularly suitable for teaching computational physics, but also widely used by many organisations for program development.

Python has many features not discussed, e.g., classes, which make it a very powerful language, fully supporting object oriented programming if required.

That Python is available free makes it particularly suitable for use in developing countries.

References

[1] P.H. Borcherds, Computational physics, Physics Education 21 (1986) 238–253.
[2] D. Scherer, P. DuBois, B. Sherwood, Scientific computing: VPython, Computing in Science and Engineering 2 (5) (October 2000) 56–62.
[3] T. Donaldson, Python as a first programming language for everyone, www.cs.ubc.ca/wccce/Program03/papers/Toby.html.
[4] A. Downey, J. Elkner, C. Meyers, How to think like a computer scientist: learning with Python, www.greenteapress.com/thinkpython.
[5] M. Lutz, D. Ascher, Learning Python, O'Reilly, ISBN 1-56592-464-9, 1999.
[6] A.S. Tanenbaum, R. van Renesse, H. van Staveren, G.J. Sharp, S.J. Mullender, A.J. Jansen, G. van Rossum, Experiences with the Amoeba distributed operating system, Communications of the ACM 33 (1990) 46–63; also on-line at http://www.cs.vu.nl/pub/amoeba/.