Volume 2 (2002)

Interface: The Journal of Education, Community and Values

3-1-2002

Introduction to using Python

Matt Ernst
Pacific University

Introduction to using Python

Introduction to using Python

Posted on **March 1, 2002** by **Editor**

By Matt Ernst <erns0637@pacificu.edu>
Junior Computer Science Major
Pacific University

Introduction

The most general and powerful solution to any problem on a computer is a programmatic solution. No other method has the flexibility and precision of code. Few other methods require so little in the way of computer power or initial expense. The downsize, of course, is that non-specialists rarely have any formal training in programming and may be intimidated by the prospect of learning to program independently. This impression is easily reinforced by existing resources. Type “programming” into a search engine and you’ll get millions of hits. There’s programming in C, C++, Java, Perl, Pascal, Fortran, and… the list could go on for some time. There’s graphics programming, game programming, network programming, sound programming, embedded programming, systems programming – again, the list is almost inexhaustible.

A considerable body of work in the form of books with titles like “Learn [some language] in 24 Hours” or “[some language] for Dummies” reinforces two different false notions: that becoming a proficient user of a particular language takes little time, and that programming in general is reserved for “smarties”; ordinary “dummies” need patronizing learning materials. In reality, it takes a good deal of time to master programming, and little to grasp the basics. Also, many titles make the false assumption that that the programming neophyte already knows what language or environment is an appropriate choice for their situation. Books will start out with an introduction touting the myriad benefits of a particular language or environment without considering the alternatives and criteria for selection. New programmers who hope to quickly accomplish useful tasks may be sorely disappointed, depending on what language/learning resource they chose.

Enter Python

The first language students are exposed to in Pacific University’s computer science program is C++ – a powerful, hard to debug, compiled language. A couple of semesters later the emphasis

switches to C, which is similar to C++ but with fewer conveniences. Nothing encourages the ability to swim like the prospect of drowning; nevertheless, non-specialists should probably avoid these languages for an introduction to programming. My own introduction to programming was in BASIC on the Commodore 64 – a fairly forgiving combination that encouraged illegible and messy programs.

For the most part, over the last two years I have personally avoided programmatic solutions to computer tasks outside of the context of computer science courses, because the languages I knew how to use (C++ and C) were just too painful to use quickly and practically. They were unforgiving, and could not accomplish much without additional libraries, often tersely documented. The information I found on the Internet generally suggested that Perl was a good language for convenient programs that were easier to write. Sadly, my brief experimentation with Perl never really went anywhere. I found the language hopelessly messy and even painful to look at (no offense intended to Perl lovers). Earlier this year, one computer science course offered students a choice of languages for an assignment; C was not a requirement. I decided that I would learn to use a new language. I picked Python since I'd heard good things about it and the web site seemed to have plenty of documentation. After doing one assignment in this language that I had never used before, I was ready to abandon my old languages altogether. It truly made programming enjoyable again, allowed me to write code that was far more compact, and allowed me to concentrate on conceptual issues instead of implementation details. I can also say that it would have made a far better introductory programming language than BASIC or C++. It strikes me as an equally good language for experienced programmers, people who don't yet program but who would like to, and people who just want to automate operations once in a while.

If you want to enjoy the Python experience you will first need to visit its homepage and obtain the appropriate distribution files for your platform. The Macintosh, Windows, and Unix are all supported. If you are uncomfortable or unauthorized to install software on the machine you wish to use, ask an administrator or technician for help. The Python examples that are to follow assume a Unix environment such as Linux or Mac OS X; however, they should run with minor modifications on other platforms as well. Once Python is properly installed, you'll want to pick an editor capable of complementing Python. However, you can always use a generic text editor, minus the extra conveniences.

The classic "Hello, world!" program is easy to write in Python.

```
#!/usr/bin/env python
print "Hello, world!"
```

This should be written in a text editor and then saved with an appropriate name, such as hello.py. Make the program executable; in a Unix shell this is accomplished by typing

```
chmod +x hello.py
```

on the command line. Now you can execute your program by typing

```
./hello.py
```

in the shell. This classic first example really doesn't teach much about programming, though. Before going further, I should heed my own indignant writing: Python is not the best tool for every task. No language is. Python is an interpreted language, as opposed to a compiled one. This means that in many or most cases Python code will run more slowly than equivalent code in a compiled language like Fortran or C. In many cases this performance disadvantage does not matter, because sheer execution speed is not the bottleneck for the particular application, but it is something to keep in mind. Additionally, while Python code is compact, the Python interpreter and its associated files weigh in at several megabytes, and they must be present on your system for Python code to run. So Python code is not suitable where storage space is at a premium, such as on an emergency boot disk or a PDA.

Programming fundamentals

Each year millions of middle and high school students learn algebra. If you're one of the many who passed algebra, you already have a handle on the fundamentals of programming, though you may not know it. If you're not confident about your algebra skills, don't fear – you can still pick up programming.

A program consists of an ordered sequence of operations. The operations may be performed on **literals** - values directly written into the program, such as `11`, `3.14159`, `'A'`, `'Northwest'`, and `['june', 'july', 'august']`. The operations may also be performed on **variables** - symbolic names that represent something else, such as `n`, `x`, `y`,`current_temperature`, and `PI`.

```
#!/usr/bin/env python
V="variable"
print "This is a " + "constant"
print "This is a " + V
```

The above program accomplishes similar things in two differnent ways. The first print statement combines two literal pieces of text (called **strings** in Python) and prints the result to the screen. The second print statement combines a literal piece of a text with a previously assigned string variable,`V`, to accomplish a similar effect. You've already learned your first Python operators. The **=** is the **assignment operator**. As its name implies, the assignment operator assigns values to variables. Variables can receive values from literals and other variables.

```
#!/usr/bin/env python
a=10
b=20
```

```
c=a
d=b+c
print a, b, c, d
```

The astute reader may notice that in this program the + symbol performed addition on two variables, while in the previous program it combined two strings. What's going on? It turns out that Python uses **dynamic typing**. What this means is that Python can recognize the difference between `9465`and `"New Mexico"` and behave accordingly. When it sees the + symbol it will combine strings and add numbers. This is a great time saver when it comes to writing programs. It may come as a surprise if you don't have prior programming experience, but C and C++ do **not** have this sort of intelligent variable handling and will blissfully do arithmetic on the internal representations of `"New Mexico"` and `9465`, leaving the programmer puzzled when the output appears garbled. Python has the standard selection of arithmetic operators: `+`, `-`, `/` (division), `*` (multiplication), and `%`(modulus – returns the remainder after whole-number division). They work basically how you would expect on numerical literals and variables, with one caveat.

```
#!/usr/bin/env python
A=10
B=4
C=4.0
print A/B, A/C, A/float(B)
```

If Python sees only whole numbers, with no decimal portion, it will perform integer division when it encounters the `/` symbol. That is, it will act as if you were doing long division on whole numbers and ignoring the remainder. To force it to consider the fractional portion of the answer, you can either add a `.0` after your number in the assignment statement, or force the type of the variable to be floating point by writing `float(your_variable)`. Now that we have assignment statements, variables, and arithmetic operations, it's possible to do a number of things. But we're still missing some key tools.

Comparison operators compare values – literals and/or variables – and return a value: 1 for true, 0 for false.

```
#!/usr/bin/env python
a=3
b=12
the_biggest=400
the_smallest=1
print a==3, a < b, a > b, the_biggest==the_smallest, 2 <= a,
the_smallest >= a
```

If you want to see if two values are the same, use the comparison operator`==`, not the

assignment operator =. The rest of the operators should look fairly familiar to anyone who's had algebra. They are < (less than), <= (less than or equal to), > (greater than), and >= (greater than or equal to).

Closely related to the idea of comparison operators are **flow control statements**. Earlier I mentioned that a program is an ordered sequence of operations. The only order we've seen thus far is that statements are executed one line after another, with no line going twice and no line being skipped. Something more is clearly needed. For one thing, there needs to be a way to execute certain instructions repeatedly without typing them out dozens of times. One way to accomplish this is with a `while` **loop**.

```
#!/usr/bin/env python
n=0
while n < 10:
   n=n+1
   print n
```

This prints the numbers 1 through 10. The extra indentation on the two lines below the `while` statement is not just for readability, although it helps there too. Indentation – whitespace – is actually a part of of Python's syntax. Modify the whitespace and you modify the program.

```
#!/usr/bin/env python
n=0
while n < 10:
   n=n+1
print n
```

This program will only print out 10, since the print statement has been moved out of the **body** of the loop by moving it back to the left. The body of the loop consists of all the indented statements following the `while`. At the top of the loop, a comparison is made to see if the condition is still true. Is n still smaller than 10? If so, the statements in the body of the loop are executed. When the end of the loop is reached (indicated by the end of indentation) the program jumps back up to the place where the loop started. When the condition at the top of the loop is finally false, the loop is left and the program resumes with the first line of code following the body of the loop. Note that it is up to the programmer to make sure that the loop can be exited. Failing to do so leads to a common error condition known as an**infinite loop**.

```
#!/usr/bin/env python
n=0
while n < 10:
   print "This loop will never exit; press CTRL-c to interrupt."
n=n+1
```

Since this program doesn't increase the size of n inside the loop, the loop will never exit. You will have to manually force the program to quit. Now that we've seen how to execute some statements repeatedly, let's see how a program can selectively choose which statements to execute.

```
#!/usr/bin/env python
n=0
while n < 10:
   n=n+1
   if n % 2 == 0:
     print n, "is even."
   else:
     print n, "is odd."
```

A number is even if it is evenly divisible by two. If it is evenly divisible by two, there will be no remainder (modulus) after integer division. So the program checks each time whether `n % 2` is equal to zero. If it is, it prints that the number is even. Otherwise (the `else` clause) it prints that the number is odd. The correct lines of code are not executed unless the pertinent condition is true. As with the `while` loop, Python uses indentation to keep track of which statements are associated with which control flow operations.

It's appropriate at this point to introduce a very useful Python data type – the`list`. Lists are collections of multiple values. Lists can contain constants or previously declared variables. Any member of a list can be accessed by using brackets `[which_list_member]` and an **index value** telling you which individual member of the list you want. Python, like most programming languages, begins counting things at zero instead of one. Since in everyday life people generally count starting with one, this can be hard to remember at first.

```
#!/usr/bin/env python
name = "Saint Brendan"
age = 63
test_person = [name, age, "male"]
print test_person[0], "is a", test_person[2]
print test_person[0], "had a birthday today."
test_person[1] = test_person[1] - 1
print test_person[0], "was", test_person[1], "years old yesterday."
test_person[1] = test_person[1] + 1
print test_person[0], "is", test_person[1], "years old today."
```

If a Python program tries to access a list entry that does not exist – like using`test_person[3]` to see whether Saint Brendan was a male – an error message will appear and the broken portion of the program will need to be fixed. By contrast, a C program would not give an error message, but would instead return garbage or crash with a cryptic

message like

```
Segmentation fault
core dumped
```

Now it's time to put things together and see what sort of useful tasks can be accomplished. Rather than coming up with another short example, let's examine some real (if not too fancy) code.

An example from real life

Last November I found, sitting on the shelves of Pacific's library, a large two-volume encyclopedia of chemistry dating from 1860. Since I love both old books and chemistry, I was fascinated. There was information on virtually every facet of mid-19th century chemical technology. Since the book was so old, I knew it was well into the public domain. I searched on the web to see if anyone else had put this book online. Nobody had, so I decided to undertake the task myself. Since I had no desire to manually correct OCR errors in more than 2000 pages of text, and didn't have an OCR package lying around anyway, I decided to put the book online as a series of scanned images in PNG format. After many hours of tedious scanning and cropping in Photoshop, the images needed to move to the Web.

Each image (corresponding to each page in the book) needed its own HTML file so there would be "Index", "Back", and "Next" links associated with each page. It also needed a master index detailing what was in each volume. Finally, with the exception of a few pages, I needed to drastically reduce the storage space and bandwidth requirements by resizing the images, reducing the number of colors, and optimizing the PNG compression.

A suite of programs known as ImageMagick could do the first half of the task. The "convert" program from this suite would handle resizing and color-reducing the images. To get the best compression possible for each individual image I used the program pngcrush. I needed to apply the "convert" and "pngcrush" programs to ~2000 image files, create HTML pages for each one of those files, and build an index that would hold all of the pages for each volume. It was clearly not a job to be done by hand. I wrote 3 related Python programs to assist me.

Program 1

`pics` was a variable corresponding to a file I opened in read mode. The file was called "infile" and it was a list of every image file in the current directory. I generated it by typing

```
ls *.png > infile
```

at the Unix command prompt. The line of code`piclist=pics.readlines()` loaded the entire contents of infile into a list called piclist, creating a new list entry whenever it found the end

of line indicator (so that there would be one image file name per list entry). The loop repeats until the entire list has been processed (`len(piclist)`returns the number of entries in piclist). You may notice that string.split() is repeatedly used. This is a very useful function from Python's string module that can split a string into multiple chunks based on a user-specified separator. The file names were of the format c-001.png, c-002.png, …, c-327.png, and so on. At various points the program stripped the file extension off (separating with '.'), obtained only the numerical portion of the file name (separating with '-'), and removed the newline character from the file name (separating with '\n', which represents the newline).

A large number of file operations happen in the loop. Near the beginning of the loop body, the program opens a new writable file with an .html extension based on the name of the image file in the current index position. Various HTML tags and text are written to the file using `tempfile.write()`. When I needed to write out quotation marks I used an **escape** (\") to protect Python from interpreting the quotation mark as the end of the string, and instead realize that it was to be a literal portion of the output. At the bottom of the loop body the program closes the file it's just written, adds one to the loop counter, and repeats until every image has an HTML page.

Program 2

The second program uses the external programs "convert" and "pngcrush" to reduce the size of all the images in the input list. The string module is used again. This time the os (operating system) module is also used, since it allows a program to start another program as if it were a user sitting at the keyboard. First the "convert" program is given a file name from the list to read from; it writes to a temporary output file. Then the "pngcrush" program uses the temporary file as input and writes to the original file name. The procedure repeats until the list of input files is exhausted. Before calling`os.system`, the command used in each case is printed out so that progress can be monitored.

Program 3

The third program is the least exciting of all, and should look familiar after the first two. This time the program reads a list of all the HTML files in the current directory, then makes an appropriate index page to sit one directory level up and link to all the individual pages. After running this program it was still necessary to edit the index by hand to give an appropriate title to each page (which is a long and ongoing process), but the program at least gave me an easy to modify template.

Conclusion

Python is a powerful and friendly language, suited for a variety of tasks. Experienced programmers and novices alike should be able to find something to love about it. The large number of modules that come with Python (the surface of which we have only scratched)

enables the quick, elegant, and easy completion of a vast variety of programming tasks with a minimum of “reinventing the wheel.”

To learn more about programming, particularly in Python, I once again heartily recommend that you visit the Python homepage. To learn more about the different modules available in Python and what they can do, have a look at the Python library reference. For a lengthier, more thorough Python introduction visit the Python.org Python tutorial. There is much more information out there, both on Python.org and elsewhere on the internet. To see the work in progress (as of March 7, 2002) which I refer to in the last section of this document, visit the James Sheridan Muspratt page.

```
#################################################################
#Program 1 - generates HTML pages for each image
#!/usr/bin/env python
import string

pics=open("infile", "r")
piclist=pics.readlines()
pics.close()

title_a="<TITLE>Muspratt Chemistry Vol. 1 pg. "
title_b="</TITLE>\n"
k=0
while k < len(piclist):
        fpart = string.split(piclist[k], '.')[0]
        tempfile=open(fpart + ".html", "w")
        tempfile.write("<HTML>\n")
        tempfile.write(title_a)
        tempfile.write(string.split(fpart, '-')[1])
        tempfile.write(title_b)
        if k + 1 < len(piclist):
                tempfile.write("<A HREF = \"" + string.split(picli
                tempfile.write("Next</A>\n")
        if k - 1 >= 0:
                tempfile.write("<A HREF = \"" + string.split(picli
                tempfile.write("Back</A>\n")
        tempfile.write("<P>")
        tempfile.write("<IMG SRC = \"" + string.split(piclist[k],
        tempfile.write("<P>")
        if k + 1 < len(piclist):
                tempfile.write("<A HREF = \"" + string.split(picli
                tempfile.write("Next</A>\n")
        if k - 1 >= 0:
                tempfile.write("<A HREF = \"" + string.split(picli
```

```
                tempfile.write("Back</A>\n")
        tempfile.write("</HTML>")
        tempfile.close()
        k=k+1

#####################################################################
#Program 2 - reduces space consumption for each image
#!/usr/bin/env python
import string
import os

pics=open("infile", "r")
piclist=pics.readlines()
pics.close()

k=0
while k < len(piclist):
        cfirst = "convert +dither -normalize -colors 8 -filter sir
            -geometry 75% " + string.split(piclist[k], '\n')[0] +
        print cfirst
        os.system(cfirst)
        csecond = "pngcrush tmp.png " + piclist[k]
        print csecond
        os.system(csecond)
        k+=1

#####################################################################
#Program 3 - make a clickable index linking all the current volume
#!/usr/bin/env python
import string
import os

pics=open("infile", "r")
piclist=pics.readlines()
pics.close()
k=0
outfile=open("volume2.html", "w")
outfile.write("<HTML>\n<TITLE>Muspratt Volume 1 index</TITLE>\n")
while k < len(piclist):
        outfile.write("<A HREF=\"muspratt1/" + string.split(piclis
        outfile.write("<br />\n")
        k+=1
outfile.write("</HTML>")
outfile.close()
```

This entry was posted in Uncategorized by **Editor**. Bookmark the **permalink [http://bcis.pacificu.edu/interface/?p=2365]** .

10 THOUGHTS ON "INTRODUCTION TO USING PYTHON"

zycie gwiazd
on **January 30, 2014 at 7:12 AM** said:

Attractive section of content. I just stumbled upon your web site and in accession capital to assert that I get in fact enjoyed account your blog posts. Any way I am going to be subscribing for the augment and even I achievement you access consistently rapidly.

nigeria
on **January 30, 2014 at 1:57 PM** said:

"Great, thanks for sharing this article post.Appreciate it Once again and again. Terrific."

plotka
on **February 1, 2014 at 1:57 AM** said:

thank u … these kinds of a fascinating topic

social network
on **February 3, 2014 at 1:48 AM** said:

Pretty section of content. I simply stumbled upon your website and in accession capital to say that I get in fact enjoyed account your blog posts. Any way I will be subscribing in your augment and even I success you access persistently rapidly.

temat

on **February 3, 2014 at 1:57 AM** said:

Hurrah, which is what I was seeking for, what a information! present the following at this website, thanks admin of this website.

nigeria social network
on **February 4, 2014 at 10:15 AM** said:

“Great, thanks for sharing this article post.Appreciate it Once more and again. Terrific.”

nigeria entertainment news
on **February 4, 2014 at 10:27 AM** said:

Hi there! I know this really is kinda off topic nonetheless I’d figured I’d ask. Would you be interested in trading links or perhaps guest writing a blog article or vice-versa? My site addresses plenty of the same subjects as yours and I believe we could greatly benefit from each other. Should you happen to be interested be my guest to send me an e-mail. I look forward to hearing from you! Terrific blog by the way!

nigeria entertainment news
on **February 4, 2014 at 10:35 AM** said:

Good work…

nigeria dating
on **February 5, 2014 at 12:18 AM** said:

What a funny blog! I really enjoyed watching this comic video with my household as well as such as my mates.

cork board ideas
on **February 5, 2014 at 11:41 AM** said:

There is certainly a great deal to find out about this subject. I like all of the points you've made.