BOSTON
TRAINING ACADEMY

Data Analysis With Python

Introduction

What is Python?

Python is a high-level scripting language which can be used for a wide variety of text processing, system administration and internet-related tasks. Unlike many similar languages, it's core language is very small and easy to mas- ter, while allowing the addition of modules to perform a virtually limitless variety of tasks. Python is a true object-oriented language, and is available on a wide variety of platforms. There's even a python interpreter written entirely in Java, further enhancing python's position as an excellent solution for internet-based problems.

Python was developed in the early 1990's by Guido van Rossum, then at CWI in Amsterdam, and currently at CNRI in Virginia. In some ways, python grew out of a project to design a computer language which would be easy for beginners to learn, yet would be powerful enough for even advanced users. This heritage is reflected in python's small, clean syntax and the thor- oughness of the implementation of ideas like object-oriented programming, without eliminating the ability to program in a more traditional style. So python is an excellent choice as a first programming language without sacri- ficing the power and advanced capabilities that users will eventually need.

Although pictures of snakes often appear on python books and websites, the name is derived from Guido van Rossum's favorite TV show, "Monty Python's Flying Circus". For this reason, lots of online and print documen- tation for the language has a light and humorous touch. Interestingly, many experienced programmers report that python has brought back a lot of the fun they used to have programming, so van Rossum's inspiration may be well expressed in the language itself.

The very Basics of Python

There are a few features of python which are different than other program- ming languages, and which should be mentioned early on so that subsequent examples don't seem confusing. Further information on all of these features will be provided later, when the topics are covered in depth.

Python statements do not need to end with a special character – the python interpreter knows that you are done with an individual statement by the presence of a newline, which will be generated when you press the "Return" key of your keyboard. If a statement spans more than one line, the safest course of action is to use a backslash (\) at the end of the line to let python know that you are going to continue the statement on the next line; you can continue using backslashes on additional continuation lines. (There are situations where the backslashes are not needed which will be discussed later.)

Python provides you with a certain level of freedom when composing a program, but there are some rules which must always be obeyed. One of these rules, which some people find very surprising, is that python uses in- dentation (that is, the amount of white space before the statement itself) to indicate the presence of loops, instead of using delimiters like curly braces ({}) or keywords (like "begin" and "end") as in many other languages. The amount of indentation you use is not important, but it must be consistent within a given depth of a loop, and statements which are not indented must begin in the first column. Most python programmers prefer to use an edi- tor like emacs, which automatically provides consistent indentation; you will probably find it easier to maintain your programs if you use consistent in- dentation in every loop, at all depths, and an intelligent editor is very useful in achieving this.

Invoking Python

There are three ways to invoke python, each with its' own uses. The first way is to type "python" at the shell command prompt. This brings up the

INVOKING PYTHON

python interpreter with a message similar to this one:

```
Python 2.2.1 (#2, Aug 27 2002, 09:01:47)
[GCC 2.95.4 20011002 (Debian prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

The three greater-than signs (>>>) represent python's prompt; you type your commands after the prompt, and hit return for python to execute them. If you've typed an executable statement, python will execute it immediately and display the results of the statement on the screen. For example, if I use python's print statement to print the famous "Hello, world" greeting, I'll immediately see a response:

```
>>> print 'hello,world' hello,world
```

The print statement automatically adds a newline at the end of the printed string. This is true regardless of how python is invoked. (You can suppress the newline by following the string to be printed with a comma.)
When using the python interpreter this way, it executes statements im- mediately, and, unless the value of an expression is assigned to a variable (See Section 6.1), python will display the value of that expression as soon as it's typed. This makes python a very handy calculator:

```
>>> cost = 27.00
>>> taxrate = .075
>>> cost * taxrate 2.025
>>> 16 + 25 + 92 * 3
317
```

When you use python interactively and wish to use a loop, you must, as always, indent the body of the loop consistently when you type your statements. Python can't execute your statements until the completion of the loop, and as a reminder, it changes its prompt from greater-than signs to periods. Here's a trivial loop that prints each letter of a word on a separate line — notice the change in the prompt, and that python doesn't respond until you enter a completely blank line.

```
>>> word = 'python'
>>> for i in word:
...     print i
...
p y t h o n
```

The need for a completely blank line is peculiar to the interactive use of python. In other settings, simply returning to the previous level of indenta- tion informs python that you're closing the loop.

You can terminate an interactive session by entering the end-of-file charac- ter appropriate to your system (control-Z for Windows, control-D for Unix), or by entering

```
import sys sys.exit()
```

or

```
raise  SystemExit
```

at the python prompt.

For longer programs, you can compose your python code in the editor of your choice, and execute the program by either typing “python”, followed by the name of the file containing your program, or by clicking on the file’s icon, if you’ve associated the suffix of your python file with the python in- terpreter. The file extension most commonly used for python files is “.py”. Under UNIX systems, a standard technique for running programs written in languages like python is to include a specially formed comment as the first line of the file, informing the shell where to find the interpreter for your program. Suppose that python is installed as /usr/local/bin/python on your system. (The UNIX command “which python” should tell you where python is installed if it’s not in /usr/local/bin.) Then the first line of your python program, starting in column 1, should look like this:

```
#!/usr/local/bin/python
```

After creating a file, say myprogram.py, which contains the special comment as its first line, you would make the file executable (through the UNIX com- mand “chmod +x myprogram.py”), and then you could execute your pro- gram by simply typing “myprogram.py” at the UNIX prompt.

When you’re running python interactively, you can instruct python to ex- ecute files containing python programs with the execfile function. Suppose that you are using python interactively, and wish to run the program you’ve stored in the file myprog.py. You could enter the following statement:

```
execfile("myprog.py")
```

The file name, since it is not an internal python symbol (like a variable name or keyword), must be surrounded by quotes.

Basic Principles of Python

Python has many features that usually are found only in languages which are much more complex to learn and use. These features were designed into python from its very first beginnings, rather than being accumulated into an end result, as is the case with many other scripting languages. If you’re new to programming, even the basic descriptions which follow may seem intimidating. But don’t worry – all of these ideas will be made clearer in the chapters which follow. The idea of presenting these concepts now is to make you aware of how python works, and the general philosophy behind python programming. If some of the concepts that are introduced here seem abstract or overly complex, just try to get a general feel for the idea, and the details will be fleshed out later

Basic Core Language

Python is designed so that there really isn’t that much to learn in the basic language. For example, there is only one basic structure for conditional pro- gramming (if/else/elif), two looping commands (while and for), and a consistent method of handling errors (try/except) which apply to all python programs. This doesn’t mean that the language is not flexible and powerful, however. It simply means that you’re not confronted with an overwhelming choice of options at every turn, which can make programming a much simpler task.

Modules

Python relies on modules, that is, self-contained programs which define a variety of functions and data types, that you can call in order to do tasks be- yond the scope of the basic core language by using the import command. For example, the core distribution of python contains modules for processing files, accessing your computer's operating system and the internet, writing CGI scripts (which handle communicating with pages displayed in web browsers), string handling and many other tasks. Optional modules, available on the Python web site (http://www.python.org), can be used to create graphical user interfaces, communicate with data bases, process image files, and so on. This structure makes it easy to get started with python, learning specific skills only as you need them, as well as making python run more efficiently by not always including every capability in every program.

Object Oriented Programming

Python is a true object-oriented language. The term "object oriented" has become quite a popular buzzword; such high profile languages as C++ and Java are both object oriented by design. Many other languages add some object-oriented capabilities, but were not designed to be object oriented from the ground up as python was. Why is this feature important? Object ori- ented program allows you to focus on the data you're interested in, whether it's employee information, the results of a scientific experiment or survey, setlists for your favorite band, the contents of your CD collection, informa- tion entered by an internet user into a search form or shopping cart, and to develop methods to deal efficiently with your data. A basic concept of object oriented programming is encapsulation, the ability to define an object that contains your data and all the information a program needs to operate on that data. In this way, when you call a function (known as a method in object-oriented lingo), you don't need to specify a lot of details about your data, because your data object "knows" all about itself. In addition, objects can inherit from other objects, so if you or someone else has designed an ob- ject that's very close to one you're interested in, you only have to construct those methods which differ from the existing object, allowing you to save a lot of work.

Another nice feature of object oriented programs is operator overloading. What this means is that the same operator can have different meanings

when used with different types of data. For example, in python, when you're dealing with numbers, the plus sign (+) has its usual obvious meaning of addition. But when you're dealing with strings, the plus sign means to join the two strings together. In addition to being able to use overloading for built-in types (like numbers and strings), python also allows you to define what operators mean for the data types you create yourself.

Perhaps the nicest feature of object-oriented programming in python is that you can use as much or as little of it as you want. Until you get comfortable with the ideas behind object-oriented programming, you can write more traditional programs in python without any problems.

Namespaces and Variable Scoping

When you type the name of a variable inside a script or interactive python session, python needs to figure out exactly what variable you're using. To prevent variables you create from overwriting or interfering with variables in python itself or in the modules you use, python uses the concept of multiple namespaces. Basically, this means that the same variable name can be used in different parts of a program without fear of destroying the value of a variable you're not concerned with.

To keep its bookkeeping in order, python enforces what is known as the LGB rule. First, the local namespace is searched, then the global names- pace, then the namespace of python built-in functions and variables. A local namespace is automatically created whenever you write a function, or a module containing any of functions, class definitions, or methods.

The global namespace consists primarily of the variables you create as part of the "top-level" program, like a script or an interactive session. Finally, the built-in namespace consists of the objects which are part of python's core. You can see the contents of any of the namespaces by using the dir command:

```
>>> dir()
[' builtins ', ' doc ', ' name ']
>>> dir( builtins )
['ArithmeticError', 'AssertionError', 'AttributeError', 'EOFError', 'Ellipsis', 'Exception', 'FloatingPointError', 'IOError', 'ImportError', 'IndexError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'OverflowError', 'RuntimeError', 'StandardError', 'SyntaxError', 'SystemError', 'SystemExit', 'TypeError', 'ValueError', 'ZeroDivisionError', 'repr', 'round', 'setattr', 'slice', 'str', 'tuple', 'type', 'vars', 'xrange']
```

The builtins namespace contains all the functions, variables and ex- ceptions which are part of python's core.

To give controlled access to other namespaces, python uses the import statement. There are three ways to use this statement. In its simplest form, you import the name of a module; this allows you to specify the various objects defined in that module by using a two level name, with the mod- ule's name and the object's name separated by a period. For example, the string module (Section 8.4) provides many functions useful for dealing with character strings. Suppose we want to use the split function of the string module to break up a sentence into a list containing separate words. We could use the following sequence of statements:

```
>>> import string
>>> string.split('Welcome to the Ministry of Silly Walks') ['Welcome', 'to', 'the', 'Ministry', 'of', 'Silly', 'Walks']
```

If we had tried to refer to this function as simply "split", python would not be able to find it. That's because we have only imported the string module into the local namespace, not all of the objects defined in the module. (See below for details of how to do that.)

The second form of the import statement is more specific; it specifies the individual objects from a module whose names we want imported into the local namespace. For example, if we only needed the two functions split and join for use in a program, we could import just those two names directly into the local namespace, allowing us to dispense with the string. prefix:

```
>>> from string import split,join
>>> split('Welcome to the Ministry of Silly Walks') ['Welcome', 'to', 'the', 'Ministry', 'of', 'Silly', 'Walks']
```

This technique reduces the amount of typing we need to do, and is an efficient way to bring just a few outside objects into the local environment.

Finally, some modules are designed so that you're expected to have top- level access to all of the functions in the module without having to use the module name as a prefix. In cases like this you can use a statement like:

```
>>> from string import *
```

Now all of the objects defined in the string module are available directly in the top-level environment, with no need for a prefix. You should use this technique with caution, because certain commonly used names from the module may override the names of your variables. In addition, it introduces lots of names into the local namespace, which could adversely affect python's efficiency.