A Quick Tour of Python

Perry Greenfield & Richard L. White

May 2, 2002

Space Telescope Science Institute
Email: help@stsci.edu

Abstract

This document serves as a whirlwind overview of the basics of Python. We hope it will give adventurous PyRAF users a better idea of what is going on in the Python environment.

Contents

1 Python is Dynamic

Python is dynamically typed. You do not need to declare variables. You may simply assign to them, which creates the variable. Variables may hold simple types (integers, floats, etc.), functions, and objects among other things.

```
x = 1
name = "sample string"
name2 = 'another sample string'
name3 = """a multiline
  string example"""
y = 3.14
longint = 1000000000000L
z = None
```

Note the last example. Python has a special value called `None`. It is generally used to represent a null value. Variable names are case sensitive. Variables can change type, simply by assigning them a new value of a different type.

```
x = 1
x = "string value"
```

Typing a variable name by itself at the interactive prompt results in its value or information about it being printed out (unless you are in PyRAF and type the name of an IRAF task, in which case CL emulation mode is entered and the task runs with no command-line arguments). Unlike the IRAF CL, no equal-sign (=) is needed to inspect a Python variab9.5()4.2(e)1dTf 8xa4a3Tc [(TD [(longintT>>>line)]TJ15.6("string)-669.16.1(t)-5.4(c)16.1(.)e)-296longintTsandF16.1(t) t=eein iL6

The indices effectively mark the gaps between the items in the string or list. Specifying `2:4` means everything between 2 and 4. Python sequences (including strings) can be indexed in reverse order as well. The index -1 represents the last element, -2 the penultimate element, and so forth. If the first index in a slice is omitted, it defaults to the beginning of the string; if the last is omitted, it defaults to the end of the string. So `s[-4:]` contains the last 4 elements of the string (`'ring'` in the above example).

Strings can be concatenated using the addition operator

```
>>> print "hello" + " world"
'hello world'
```

And they can be replicated using the multiplication operator

```
>>> print "hello"*5
'hellohellohellohellohello'
```

There is also a string module (see below for more on modules) in the standard Python library. It provides many additional operations on strings. In the latest version of Python (2.0), strings also have methods that can be used for further manipulations. For example `s.find('abc')` returns the (zero-based) index of the first occurence of the substring `'abc'` in string `s` (-1 if it is not found.) The string module equivalent is `string.find(s,'abc')`.

3.2 Lists

One can view lists as generalized, dynamically sized arrays. A list may contain a sequence of any legitimate Python objects: numbers, strings functions, objects, and even other lists. The objects in a list do not have to be the same type. A list can be specifically constructed using square brackets and commas:

```
x = [1,4,9,"first three integer squares"]
```

Elements of lists can be accessed with subscripts and slices just like strings. E.g.,

```
>>> x[2]
9
>>> x[2:]
[9,"first three integer squares"]
```

They can have items appended to them:

```
>>> x.append('new end of list')
  or
>>> x = x + ['new end of list'] # i.e., '+' concatenates lists
>>> x
[1,4,9,'first three integer squares','new end of list']
```

(Here # is the Python comment delimiter.) Elements can be deleted and inserted:

```
>>> del x[3]
>>> x
[1, 4, 9, 'new end of list']
>>> x.insert(1,'two')
>>> x
[1, 'two', 4, 9, 'new end of list']
```

The product of a list and a constant is the result of repeatedly concatenating the list to itself:

```
>>> 2*x[0:3]
[1, 'two', 4, 1, 'two', 4]
```

```
>>> employee_id = "ricky":11,"fred":12,"ethel":15
>>> print employee_id["fred"]
12
```

Note that braces are used to enclose dictionary definitions. New items are easily added:

```
>>> employee_id["lucy"] = 16
```

will create a new entry.

There are many operations available for dictionaries; for example, you can get a list of all the keys:

```
>>> print employee_id.keys()
```

```
if x = 0:
   print "did I really mean to write a never used print statement?"
```

But this is legal:

```
if x == 0:
   print "much better"
```

5 Control Constructs

Python has if, for, and while control statements. The for statement is different than that in most other languages; it is more like the `foreach` loop in `csh`. It takes the following form:

```
for item in itemlist:
  print item
```

The loop body is executed with a new value for the variable `item` for each iteration. The values are taken from a sequence-like object (such as a string, list, or tuple ... but other possibilities exist), and `item` is set to each of the values in the sequence.

Note the colon after the statement – all statements that control the execution of a following block of statements (including for, if, while, etc.) end with a colon.

To get the common loop over consecutive integers, use the built-in `range` function:

```
for i in range(100): print i
```

`range(100)` constructs a list of values from 0 to 99 (yes, 99!) and the for loop repeats 100 times with values of `i` starting at 0 and ending at 99. The argument to `range` is the number of elements in the returned list, not the maximum value. There are additional arguments to `range` if you want a loop that starts at a value other than zero or that increments by a value other than 1.

6 Modules and Namespaces

There are different ways of loading modules (the Python equivauaoc

You can import using an alternate form:

```
>>> from string import *
>>> print capitalize(s)
```

Note that with this form of import, you do not prepend the module name to the function name, which makes using the `capitalize` function a bit more convenient. But this approach does have drawbacks. All the string module names appear in the user namespace. Importing many modules this way greatly increases the possibility of name collisions. If you import a module you are developing and want to reload an edited version, importing this way makes it very difficult to reload (it's possible, but usually too tedious to be worthwhile). So, **when debugging Python modules or scripts interactively, don't use `from mmm import *`!**

A better way to use the `from ... import ...` form of the import statement is to specify explicitly which names you want to import:

```
>>> from string import capitalize
>>> print capitalize(s)
```

This avoids cluttering your namespace with functions and variables that you do not use. Namespaces in general are a large and important topic in Python but are mainly beyond the scope of this quick overview. They are ubiquitous – there are namespaces associated with functions, modules, and class instances, and each such object has a local and a global namespace.

7 Objects, Classes and What They Mean to You

If you are an object-oriented programmer, you'll find Python a pleasure to use: it provides a well-integrated object model with pretty much all of the tools you expect in an object-oriented language. But nothing forces you to write classes in Python. You can write traditional procedural code just fine if you think OO stuff is for namby pamby dweebs. Nonetheless, it is helpful to know a bit about it since many of the the standard libraries are written with objects in mind. Learning to use objects is much easier than learning how to write good classes.

Typically an object is created by calling a function (or at least something that looks like a function) and assigning the return value to a variable. Thereafter you may access and modify attributes of the object (its local variables, in effect). You can perform operations on the object (or have it perform actions) by calling its 'methods'. File handles provide a good example of what we mean.

```
>>> fin = open('input.txt','r') # open file in read mode and return file handle
>>> lines = fin.readlines()     # call a method that returns a list of lines
>>> for line in lines: print line
>>> print fin.name              # print name of the file for the fin object
                                # 'name' is an attribute of the file object
>>> fin.close()                 # close the file
```

This approach is used by many Python libraries.

8 Defining Functions

Defining functions is easy:

```
def factorial(n):
    """a simple factorial function with no overflow protection"""
    if n:
        return n*factorial(n-1)
    else:
        return 1
```

Note that, like other code blocks, indentation is used to decide what belongs in the function body. Also note that Python allows recursive functions.

The string at the beginning of the function is called a “doc string” and contains documentation information on the function. It is available as the `__doc__` attribute of the function, so `print factorial.__doc__` will print the

- *Quick Python*