Python 101 (part 5): Snake Oil For The Soul

By Vikram Vaswani

Table of Contents

Couch Potato

Yeah, yeah, I know. So far, all we've been doing in this tutorial is mucking around with Python's data structures, dumb creatures with names like "tuples" and "lists", and messing about with twisty things like the "for" and "while" loops. And you're fed up.

All these concepts and structures, beautifully−designed and wonderfully−elegant though they may be, leave you cold. You just don't see the point of this exercise. As a matter of fact, you're thinking, now is probably a good time to drop this thing and get on with something more worthwhile − "Ally McBeal", perhaps?

I hear you.

Over the next few pages, we're going to leave the rarefied world of Python data structures and enter something a lot more tangible − the filesystem. I'm going to be showing you how to use Python to interact with files on your filesystem, opening them, reading them and writing to them.

No longer will you be playing abstract games with list slices and dictionary keys − by the end of this article, you'll be able to watch as your Python code creates new files in front of your eyes, or rub your hands in glee as you read other people's private documents. You will have the power to wipe out each and every file on your hard drive, or selectively nuke only the ones you don't like. You will be respected...and, more importantly, you will be feared.

Even "Ally McBeal" can't top that!

Air In A Bottle

Like all widely−used programming languages, Python has the very useful ability to read data from, and write data to, files on your system. It accomplishes this via "file handles" − a programming construct that allows Python scripts to communicate with data structures like files, directories and other Python scripts.

For my first example, I'll assume that I have a text file called "snakeoil.txt", containing the following commercial:

```
After years of research, scientists at AB Labs have come up
with a radical
new product, once that promises to improve the quality of life
for humans
and animals everywhere.

We call it Air In A Bottle(tm), and we firmly consider it to
be one of our
best inventions ever.

Carry it around with you (the bottle is titanium, the same
material used in
rocket ships, and the air is, well, air) and inhale from it
whenever you
need some air. Or give it to your friends as a present (after
all, they
need air too!)

Air In A Bottle(tm). Get some today!
```

Now, in order to read this data into a Python program, I need to open the file and assign it a file handle − this file handle can then be used to interact with the data.

```
Python 1.5.2 (#1, Aug 25 2000, 09:33:37) [GCC 2.96 20000731
(experimental)] on
linux−i386
Copyright 1991−1995 Stichting Mathematisch Centrum, Amsterdam
>>> data = open("snakeoil.txt")
>>> data.read()
'After years of research, scientists at AB Labs have come up
with a radical
new
```

```
product, once that promises to improve the quality of life for
humans and
animals everywhere.\012 \012We call it Air In A Bottle(tm),
and we firmly
consider it to be one of our best inventions
ever.\012\012Carry it around
with you (the bottle is titanium, the same material used in
rocket ships,
and the air is, well, air) and inhale from it whenever you
need some air.
Or give it to your friends as a present (after all, they need
air
too!)\012\012Air In A Bottle(tm). Get some
today!\012'
>>> data.close()
>>>
```

In this case, I've first used the open() function to open the file "snakeoil.txt" and assign it to the file handle "data". Next, I've used the read() function to read everything in the file, and the close() function to close the handle after I'm done.

Since this is the command−line interpreter, a read() call will display the contents of the file on the console, complete with line breaks (in case you're wondering, that's what \012 is.) The read() function reads the entire file into a single string.

Treading The Right Path

You can specify a full path to the file as well:

```
>>> data = open("/tmp/snakeoil.txt")
>>>
```

You could write a Python program that accomplishes the same thing:

```
#!/usr/bin/python

# open file
data = open("snakeoil.txt")

# read file
dump = data.read()
print dump, "-- EOF -- EOF --"

# clean up
data.close()
```

And when you run this script, Python should return the contents of the file "snakeoil.txt", with a message at the end.

```
After years of research, scientists at AB Labs have come up
with a radical
new product, once that promises to improve the quality of life
for humans
and animals everywhere.

We call it Air In A Bottle(tm), and we firmly consider it to
be one of our
best inventions ever.

Carry it around with you (the bottle is titanium, the same
material used in
rocket ships, and the air is, well, air) and inhale from it
```

```
whenever you
need some air. Or give it to your friends as a present (after
all, they
need air too!)

Air In A Bottle(tm). Get some today!
-- EOF -- EOF --
```

Notice that since this is being run as a script and not via the command−line interpreter, the \012s are automatically converted into line breaks.

And look what happens if the file doesn't exist:

```
>>> data = open("/tmp/nosuchfile.txt")
Traceback (innermost last):
File "", line 1, in ?
IOError: [Errno 2] No such file or directory:
'/tmp/nosuchfile.txt'
>>>
```

Standing In Line

There's also another method of reading data from a file − the readline() method call, used to read a single line at a time.

```
>>> data = open("snakeoil.txt")
>>> data.readline()
'After years of research, scientists at AB Labs have come up
with a radical
new product, once that promises to improve the quality of life
for humans
and animals everywhere.\012'
>>> data.readline()
' \012'
>>> data.readline()
'We call it Air In A Bottle(tm), and we firmly consider it to
be one of our
best inventions ever.\012'
>>>
```

You can combine this with a loop that will run through the file, printing one line after another:

```
#!/usr/bin/python

# open file
data = open("snakeoil.txt")

# read first line
line = data.readline()

# loop until EOF
while line != "":
print line,
line = data.readline()
else:
print "-- all done --"

# clean up
data.close()
```

Well, it works − but how about making it a little more efficient? Instead of reading a file line by line, you can also suck the entire thing straight into your program via a list − definitely more likely to impress the girls!

```
#!/usr/bin/python

# open file
data = open("snakeoil.txt")

# read file into list
lines = data.readlines()

# loop until EOF
for x in lines:
print x,
else:
print "-- all done --"

# clean up
data.close()
```

In this case, the readlines() method is used to read the file into a list. Each element of the list now corresponds to a single line from the file.

```
>>> data.readlines()
['After years of research, scientists at AB Labs have come up
with a
radical new product, once that promises to improve the quality
of life for
humans and animals everywhere.\012', ' \012', 'We call it Air
In A
Bottle(tm), and we firmly consider it to be one of our best
inventions
ever.\012', '\012', 'Carry it around
with you (the bottle is titanium, the same material used in
rocket ships,
and the air is, well, air) and inhale from it whenever you
need some air.
Or give it to your friends as a present (after all, they need
air
too!)\012', '\012', 'Air In A Bottle(tm). Get some
today!\012']
>>>
```

Once this has been done, it's a simple matter to run through the list and display its contents with the "for" loop.

Learning To Write

Obviously, reading a file is no great shakes − but how about writing to a file?

Well, it isn't all that hard either − you just need to open the file with an optional argument indicating that you would like to write to it.

```
>>> data = open("snakeoil.txt", "w")
>>>
```

The optional argument specifies the mode in which the file should be opened, and may be either one of "r" (read only), "w" (write) or "a" (append). Opening a file with "w" is akin to creating a new, empty file − the previous contents of the file are erased. Opening a file with "a" retains the previous contents of the file and appends new material to the end of the file.

You can add the "+" operator to the three modes above to indicate that both read and write operations should be allowed.

Just as Python has the read() and readlines() methods for file read, it has corresponding write() and writelines() functions for file output. The write() function writes a single string to the file, as demonstrated in the following example:

```
>>> data = open("snakeoil.txt", "a")
>>> data.write("-- end of commercial --")
>>> data.close()
>>>
```

```
$ cat snakeoil.txt
After years of research, scientists at AB Labs have come up
with a radical
new product, once that promises to improve the quality of life
for humans
and animals everywhere.

We call it Air In A Bottle(tm), and we firmly consider it to
```

```
be one of our
best inventions ever.

Carry it around with you (the bottle is titanium, the same
material used in
rocket ships, and the air is, well, air) and inhale from it
whenever you
need some air. Or give it to your friends as a present (after
all, they
need air too!)

Air In A Bottle(tm). Get some today!
-- end of commercial --$
```

It should be noted that write() does not automatically add a line break to your string; this needs to be done manually.

The writelines() method takes a list and writes each element of the list to the file sequentially. The following Python program builds a list from user input, and then writes the list to the file.

```
#!/usr/bin/python

# empty list
comments = []

# ask for three comments and build list
for x in range (0,3):
rawInput = raw_input("Enter a comment: ")
comments.append("User comment:" + rawInput + "\n")

# open file
data = open("snakeoil.txt", "a")

# write list
data.writelines(comments)

# clean up
data.close()
```

Here's what it looks like:

```
Enter a comment: This product sucks!
Enter a comment: How stupid do you think we are??????
Enter a comment: Can I have some more?

$ cat snakeoil.txt
After years of research, scientists at AB Labs have come up
with a radical
new product, once that promises to improve the quality of life
for humans
and animals everywhere.

We call it Air In A Bottle(tm), and we firmly consider it to
be one of our
best inventions ever.

Carry it around with you (the bottle is titanium, the same
material used in
rocket ships, and the air is, well, air) and inhale from it
whenever you
need some air. Or give it to your friends as a present (after
all, they
need air too!)

Air In A Bottle(tm). Get some today!
User comment:This product sucks!
User comment:How stupid do you think we are??????
User comment:Can I have some more?
```

As you can see, the comments entered at the command line have been appended to the end of the file.

Desperately Seeking Python

There are a few other miscellaneous methods you should know about.

The seek() method moves the internal file pointer to a specific byte location, while the tell() method returns the current location of the file pointer.

```
>>> data = open("snakeoil.txt")
>>> data.seek(20)
>>> data.tell()
20
>>> data.readline()
'rch, scientists at AB Labs have come up with a radical new
product, once
that promises to improve the quality of life for humans and
animals
everywhere.\012'
>>> data.tell()
173
>>>
```

The truncate() method is used to truncate a file to a specific byte size.

```
>>> data = open("snakeoil.txt", "a")
>>> data.truncate(175)
>>>
```

And there are also a couple of interesting file properties you might like to use in your Python programs. The "mode" and "name" properties return the file mode and file name respectively, while the "closed" property returns a Boolean value indicating whether or not the file is closed.

```
>>> data = open("snakeoil.txt", "a")
>>> data.mode
'a'
>>> data.name
'snakeoil.txt'
>>> data.closed
```

```
0
>>> data.close()
>>> data.closed
1
>>>
```

Python also comes with an "os" module, which adds a bunch of other file methods and properties to the built−in ones described above...but I'll leave that to you to experiment with. In case you don't know what a module is, don't worry − all will be explained soon!

String Theory

Now, in addition to the various methods already explained in this and previous articles, Python 2.0 comes with a whole bunch of new methods for the various data types you've studied. Over the next couple of pages, I'd like to briefly illustrate these with some examples.

A number of string methods are available to convert and check string formats − take a look:

```
>>> # string methods
>>> str = "jackfrost"
>>> # check for lowercase
>>> str.islower()
1
>>> # check for uppercase
>>> str.isupper()
0
>>> # capitalize first character
>>> str.capitalize()
'Jackfrost'
>>> # convert to uppercase
>>> str.upper()
'JACKFROST'
>>> # convert to lowercase
>>> str.lower()
'jackfrost'
>>> # let's try another string
>>> str = "Jack Frost"
>>> # check if this is a title
>>> str.istitle()
1
>>> # split string and return sections as list
>>> str.split(" ")
['Jack', 'Frost']
>>> str.swapcase()
'jACK fROST'
>>> str = "Mummy"
>>> # count number of occurrences within a string
>>> str.count("m")
2
>>>
```

Incidentally, the count() method also works with lists.

```
>>> list = ["red", "green", "blue", "red", "yellow", "blue",
"black",
"white", "green"]
>>> list.count("green")
2
>>>
```

Pop() Goes The Weasel

The pop() function allows you to remove an item from the end of a list,

```
>>> list
['red', 'green', 'blue', 'red', 'yellow', 'blue', 'black',
'white', 'green']
>>> len(list)
9
>>> list.pop()
'green'
>>> len(list)
8
>>>
```

and the str() function converts a list into a printable string.

```
>>> list
['red', 'green', 'blue', 'red', 'yellow', 'blue', 'black',
'white', 'green']
>>> str(list)
"['red', 'green', 'blue', 'red', 'yellow', 'blue', 'black',
'white',
'green']"
>>>
```

Finally, the max() and min() methods return the largest and smallest item within a string, list or tuple.

```
>>> # list
>>> friends = ["Rachel", "Ross", "Joey", "Phoebe", "Monica",
"Chandler"]
>>> max(friends)
'Ross'
>>> min(friends)
'Chandler'
>>> # string
>>> alphabet = "abcdefghijklmnopqrstuvwxyz"
```

```
>>> min(alphabet)
'a'
>>> max(alphabet)
'z'
>>> # tuple
>>> num = (1,2,3)
>>> max(num)
3
>>> min(num)
1
>>>
```

And that's about it for the moment. In this article, you learnt how to open a connection between your Python program and a file residing on your filesystem. You then learnt how to use built−in Python methods to read data from and to that file, together with some additional methods and properties that might come in useful. Finally, you've also added to your store of Python knowledge with a look at some of the new features available in Python 2.x.

Over the next few article, we'll be elevating our view a little, with a look at the Python framework for software abstractions like functions, classes and modules. Specifically, in the next article, I'll be showing you how to build your own functions, and also discussing return values and arguments in the context of a Python program. Make sure you don't miss that one!