Finance 5350 - Fall 2016

Computational Financial Modeling

These notes are based off the material by Jake VanderPlas on his GitHub repository. A big thanks goes to Dr. VanderPlas for making these notebooks available under the "No Rights Reserved" CC0 license!

"No Rights Reserved" CC0

These notes form essentially the first chapter of his forthcoming book, which looks fantastic:

> A Whirlwind Tour of Python by Jake VanderPlas (O'Reilly). Copyright 2016 O'Reilly Media, Inc., 978-1-491-96465-1.

A Quick Tour of Python Language Syntax

Python was originally developed as a teaching language, but its ease of use and clean syntax have led it to be embraced by beginners and experts alike. The cleanliness of Python's syntax has led some to call it "executable pseudocode", and indeed my own experience has been that it is often much easier to read and understand a Python script than to read a similar script written in, say, C. Here we'll begin to discuss the main features of Python's syntax.

Syntax refers to the structure of the language (i.e., what constitutes a correctly-formed program). For the time being, we'll not focus on the semantics – the meaning of the words and symbols within the syntax – but will return to this at a later point.

Consider the following code example:

```
# set the midpoint
midpoint = 5

# make two empty lists
lower = []; upper = []

# split the numbers into lower and upper
for i in range(10):
    if (i < midpoint):
        lower.append(i)
    else:
        upper.append(i)

print("lower:", lower)
print("upper:", upper)
```

This script is a bit silly, but it compactly illustrates several of the important aspects of Python syntax. Let's walk through it and discuss some of the syntactical features of Python

Comments Are Marked by

The script starts with a comment:

```
# set the midpoint
```

Comments in Python are indicated by a pound sign (#), and anything on the line following the pound sign is ignored by the interpreter. This means, for example, that you can have stand-alone comments like the one just shown, as well as inline comments that follow a statement. For example:

```
x += 2  # shorthand for x = x + 2
```

Python does not have any syntax for multi-line comments, such as the `/* ... */` syntax used in C and C++, though multi-line strings are often used as a replacement for multi-line comments (more on this in String Manipulation and Regular Expressions).

End-of-Line Terminates a Statement

The next line in the script is

```
midpoint = 5
```

This is an assignment operation, where we've created a variable named `midpoint` and assigned it the value `5`. Notice that the end of this statement is simply marked by the end of the line. This is in contrast to languages like C and C++, where every statement must end with a semicolon (`;`).

In Python, if you'd like a statement to continue to the next line, it is possible to use the "\" marker to indicate this:

```
x = 1 + 2 + 3 + 4 +\
    5 + 6 + 7 + 8
```

It is also possible to continue expressions on the next line within parentheses, without using the "\" marker:

```
x = (1 + 2 + 3 + 4 +
     5 + 6 + 7 + 8)
```

Most Python style guides recommend the second version of line continuation (within parentheses) to the first (use of the "\" marker).

Semicolon Can Optionally Terminate a Statement

Sometimes it can be useful to put multiple statements on a single line. The next portion of the script is

```
lower = []; upper = []
```

This shows the example of how the semicolon (;) familiar in C can be used optionally in Python to put two statements on a single line. Functionally, this is entirely equivalent to writing

```
lower = []
upper = []
```

Using a semicolon to put multiple statements on a single line is generally discouraged by most Python style guides, though occasionally it proves convenient.

Indentation: Whitespace Matters!

Next, we get to the main block of code:

```
for i in range(10):
    if i < midpoint:
        lower.append(i)
    else:
        upper.append(i)
```

This is a compound control-flow statement including a loop and a conditional – we'll look at these types of statements in a moment. For now, consider that this demonstrates what is perhaps the most controversial feature of Python's syntax: whitespace is meaningful!

In programming languages, a *block* of code is a set of statements that should be treated as a unit. In C, for example, code blocks are denoted by curly braces:

```
// C/C++ code
for(int i=0; i<100; i++)
{
    // curly braces indicate code block
    total += i;
}
```

In Python, code blocks are denoted by *indentation*:

```
for i in range(100):
    # indentation indicates code block
    total += i
```

In Python, indented code blocks are always preceded by a colon (:) on the previous line.

The use of indentation helps to enforce the uniform, readable style that many find appealing in Python code. But it might be confusing to the uninitiated; for example, the following two snippets will produce different results:

```
>>> if x < 4:          >>> if x < 4:
...     y = x * 2      ...     y = x * 2
...     print(x)       ... print(x)
```

In the snippet on the left, `print(x)` is in the indented block, and will be executed only if `x` is less than `4`. In the snippet on the right `print(x)` is outside the block, and will be executed regardless of the value of `x`!

Python's use of meaningful whitespace often is surprising to progammers who are accustomed to other languages, but in practice it can lead to much more consistent and readable code than languages that do not enforce indentation of code blocks. If you find Python's use of whitespace disagreeable, I'd encourage you to give it a try: as I did, you may find that you come to appreciate it.

Finally, you should be aware that the *amount* of whitespace used for indenting code blocks is up to the user, as long as it is consistent throughout the script. By convention, most style guides recommend to indent code blocks by four spaces, and that is the convention we will follow in this report. Note that many text editors like Emacs and Vim contain Python modes that do four-space indentation automatically.

Whitespace *Within* Lines Does Not Matter

While the mantra of *meaningful whitespace* holds true for whitespace *before* lines (which indicate a code block), white space *within* lines of Python code does not matter. For example, all three of these expressions are equivalent:

```
x=1+2
x = 1 + 2
x             =         1     +                   2
```

Abusing this flexibility can lead to issues with code readibility – in fact, abusing white space is often one of the primary means of intentionally obfuscating code (which some people do for sport). Using whitespace effectively can lead to much more readable code, especially in cases where operators follow each other – compare the following two expressions for exponentiating by a negative number:

```
x=10**-2
```

to

```
x = 10 ** -2
```

I find the second version with spaces much more easily readable at a single glance. Most Python style guides recommend using a single space around bi-

nary operators, and no space around unary operators. We'll discuss Python's operators further in Basic Python Semantics: Operators.

Parentheses Are for Grouping or Calling

In the previous code snippet, we see two uses of parentheses. First, they can be used in the typical way to group statements or mathematical operations:

```
2 * (3 + 4)
```

```
14
```

They can also be used to indicate that a *function* is being called. In the next snippet, the `print()` function is used to display the contents of a variable (see the sidebar). The function call is indicated by a pair of opening and closing parentheses, with the *arguments* to the function contained within:

```
print('first value:', 1)
```

```
first value: 1
```

```
print('second value:', 2)
```

```
second value: 2
```

Some functions can be called with no arguments at all, in which case the opening and closing parentheses still must be used to indicate a function evaluation. An example of this is the `sort` method of lists:

```
L = [4,2,3,1]
L.sort()
print(L)
```

```
[1, 2, 3, 4]
```

The "`()`" after `sort` indicates that the function should be executed, and is required even if no arguments are necessary.

Aside: A Note on the `print()` Function

Above we used the example of the `print()` function. The `print()` function is one piece that has changed between Python *2.x* and Python *3.x*. In Python 2, `print` behaved as a statement: that is, you could write

```
# Python 2 only!
>> print "first value:", 1
first value: 1
```

For various reasons, the language maintainers decided that in Python 3 `print()` should become a function, so we now write

```
# Python 3 only!
>>> print("first value:", 1)
first value: 1
```

This is one of the many backward-incompatible constructs between Python 2 and 3. As of the writing of this book, it is common to find examples written in both versions of Python, and the presence of the `print` statement rather than the `print()` function is often one of the first signs that you're looking at Python 2 code.

Finishing Up and Learning More

This has been a very brief exploration of the essential features of Python syntax; its purpose is to give you a good frame of reference for when you're reading the code in later sections. Several times we've mentioned Python "style guides", which can help teams to write code in a consistent style. The most widely used style guide in Python is known as PEP8, and can be found at https://www.python.org/dev/peps/pep-0008/. As you begin to write more Python code, it would be useful to read through this! The style suggestions contain the wisdom of many Python gurus, and most suggestions go beyond simple pedantry: they are experience-based recommendations that can help avoid subtle mistakes and bugs in your code.