PYTHON PROGRAMMING

Dr Christian Hill
7–9 November 2016

The History of Python

Invented by Guido van Rossum* at the Centrum Wiskunde & Informatica in Amsterdam in the early 1990s

Named after *Monty Python's Flying Circus*

Grew in popularity throughout the 2000s, initially as a web-scripting language

Now used extensively in computing and scientific circles, e.g. by Google, YouTube, CERN, NASA, Disney, ...

* Python's Benign Dictator For Life (BDFL)

Features of Python

Free and open-source

Multi-platform

Interpreted

High-level, expressive and readable

Object-oriented

Dynamically-typed

Strongly-typed

Automatic memory-management

This Course

3 days is a short time!

The focus is on breadth, not depth

The exercises ("practicals") are where most of the learning happens

Some of the practicals are tough: it isn't necessary to complete them all...

... but solutions will be provided

It takes a long time to remember everything to do with a programming language; luckily, the Python documentation is very good.

This Course

We cover:

The Python shell: using Python as a calculator

The essential syntax of a Python program; loops, flow-control, functions, etc.

Major data structures: strings, lists, tuples, sets and dictionaries

File I/O, Errors and Exceptions, cool Python stuff

Classes and object-oriented programming (briefly)

Floating point arithmetic

an introduction to NumPy, SciPy and Matplotlib

Learning Python

C. Hill, "Learning Scientific Programming with Python", CUP (2016)

M. Lutz, "Learning Python", 5th ed., O'Reilly (2013)

M. Pilgrim, "Dive Into Python", APRESS (2004); also available online for free at: http://www.diveintopython.net/

M. Dawson, "Python Programming for the Absolute Beginner", 3rd ed., Course Technology PTR (2010)

D. M. Beazley, "Python Essential Reference", Addison Wesley (2009)

S. Oliveira and D. Stewart, "Writing Scientific Software: A Guide to Good Style", CUP (2006)

Online Python Resources

The official Python site: http://www.python.org/

The official Python documentation: http://docs.python.org/

The official Python tutorial: http://docs.python.org/tutorial/

My own scipython website: http://scipython.com

The mighty StackOverflow: http://stackoverflow.com/ (search the archive before posting your question!)

Learn Python the hard way: http://learnpythonthehardway.org/

Learn Python the very hard way: http://www.pythonchallenge.com/

The Python Shell

From a terminal (e.g. tcsh on socrates):

```
% python
Python 2.6.4 (r264:75706, Jul 20 2010, 05:25:12)
[GCC 4.4.3] on sunos5
Type "help", "copyright", "credits" or "license" for
more information.
>>>
```

Jupyter Notebook

Search for Jupyter at the Windows Start menu
Launches in a browser menu
Create a folder to store your work
Select New Python Notebook to open new window with your new Jupyter Notebook cells.
Shortcuts:

Shift-Enter: Execute the cell, showing any output, and then move the cursor onto the cell below. If there is no cell below, a new empty one will be created.
CTRL-Enter: Execute the cell in place, but keep the cursor in the current cell. Useful for quick "disposable" commands to check if a command works or for retrieving a directory listing.
Alt-Enter: Execute the cell, showing any output, and then insert a new cell immediately beneath it.

Basic Arithmetic

Basic operators:

`+` (addition)
`-` (subtraction)
`*` (multiplication)
`/` (division)
`//` (integer division)
`%` (modulus, *i.e.* remainder)
`**` (exponentiation)

Basic Arithmetic

You can use the Python shell as a calculator! Examples:

```
>>> 3 + 4
7
>>> 6 * 7
42
>>> 2**3
8
>>> 14 // 3
4
>>> 14 % 3
2
>>> 14. / 3
4.666666666666667
```

Integer division *rounds down* ("floor division")

Floating point numbers are indicated by a decimal point
The integer 3 is automatically "converted up" to float

Basic Arithmetic

A reminder about modular arithmetic

`>>> a // n` means $\texttt{floor(a/n)} = \left\lfloor \frac{a}{n} \right\rfloor$, the largest integer less than or equal to `a/n`

`>>> b = a % n` is the remainder, $\texttt{b = a} - \left\lfloor \frac{a}{n} \right\rfloor \times \texttt{n}$

For example,

```
7 // 3 = 2    since 7 = 2*3 + 1
7 % 3 = 1
```

Basic Arithmetic

Operator Precedence

Operators with the same precedence evaluate left-to-right
But "an unparenthesised sequence of power operators are evaluated right-to-left" (e.g. `3**2**3 = 3**8`, not `9**3`)
Use parentheses to over-ride precedence
In *decreasing* order of precendence:

`**`

unary `-` (*i.e.* negative numbers)

`*, /, //, %`

`+, -`

Basic Arithmetic

Operator Precedence Examples:

```
>>> 3 + 4 * 2
11
>>> (3 + 4) * 2
14
>>> -2**4
-16
>>> (-2)**4
16

>>> 2 * 14 % 3
1
>>> 14 % 3 * 2
4
```

Exponentiation has higher precedence than unary minus

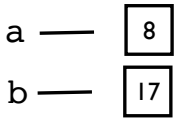
Use parentheses to raise `-2` to the fourth power.

Evaluation is left-to-right, so this is `28 % 3` ...
... but this is `2 * 2`

Variables

Variables are used store, label and manipulate data
They do not need to be "declared" before use

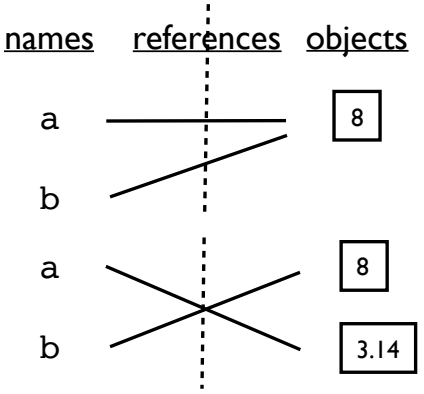
```
>>> a = 8
>>> b = 17
>>> a
8
>>> a - b
-9
>>> a / 4 + b
19
>>> c = b - 7
>>> c
10
```



b - 7 = 17 - 7 = 10 — c

Variables

```
>>> a = 8
>>> b = a
```




```
>>> a = 8
>>> b = a
>>> a = 3.14
```

Numbers, like `8` and `3.14` are *immutable*: the line `a = 3.14` creates a new object and labels it `a`. Note that `b` is not changed.

Dynamic typing

```
>>> a = 8
>>> a = a * 1.2
>>> a
9.6
```

This is an *assignment*: "multiply `a` by `1.2` and store the result in `a`", not a mathematical equation.

The multiplication changes the *type* of a from `int` to `float` (it is automatically "promoted" to the more general type needed to store the result)

An example of *duck typing*: we don't need to declare the type of `a`: "if it looks like a duck, and it walks like a duck and it quacks like a duck, then it's a duck"

A shorter way of writing:
`>>> a = a * 1.2`
is to use the idiom:
`>>> a *= 1.2`

Variable Names

Rules for variable names

Variables are *case-sensitive* (`a` and `A` are different variables)
Can contain any letter, the underscore character ('_'), and any digit (0-9) ...
... but must not start with a digit
Must not be a *reserved keyword*. In Python 3:

```
False   True    None    and     as
assert  break   class   continue  def
del     elif    else    except  finally
for     from    global  if      import
in      is      lambda  nonlocal  not
or      pass    raise   return  try
while   with    yield
```

Variable Names

Style guide for variable names

Make your variable names meaningful:
`area` is better than `a`

But not too long:
`the_area_of_the_triangle` is too unwieldy

Use lower-case names, with words separated by underscores:
`mean_height` not `MeanHeight` ("CamelCase")

Don't use `I` (upper-case i), `l` (lower case L), or `O` (upper-case o): they look too much like the digits `1` and `0`.

The variables `i`, `j`, `k` are usually expected to be integer counters

Comparison operators

Objects (such as numbers) are compared using the operators:

`==`	(equals)
`!=`	(not equal to)
`>`	(greater than)
`<`	(less than)
`>=`	(greater than or equal to)
`<=`	(less than or equal to)

The result of a comparison is a *boolean* object (of type `bool`): `True` or `False`. For example,

```
>>> 7 == 8
False
>>> 4 >= 3.14
True
```

Again, `4` is promoted up to `float` so the comparison is between objects of the same type.

Logic

The keywords `and`, `or` and `not` can be used to string together comparisons. In decreasing order of precedence:

```
not
and
or
```

For example:

```
>>> 7 > 4 and -1 <= 0
True
>>> not 7.5 < 0.9 or 4 == 4
True
>>> not (7.5 < 0.9 or 4 == 4)
False
```

i Over-ride precedence with parentheses

Logic

Truth tables:

P	not P
T	F
F	T

P	Q	P and Q
T	T	T
F	T	F
T	F	F
F	F	F

P	Q	P or Q
T	T	T
F	T	T
T	F	T
F	F	F

i `or` is 'inclusive'