APT Session 2: Python

Laurence
Tratt

KING'S College LONDON

Software Development Team
2018-10-19

What to expect from this session: Python

1. What is Python?
2. Basic Python functionality.

What to expect from this session: Python

1. What is Python?
2. Basic Python functionality.
3. Building a web spider.

Prerequisites

You should have:

1. Downloaded a text editor of your choice and familiarised yourself with its basic operation.
2. Downloaded Python 3 from https://www.python.org/downloads/release/python-370/
3. Ensured your laptop can connect to one of the College's wireless networks.

Editing text

- Programming involves editing text. You therefore need a *text editor*.

Editing text

- Programming involves editing text. You therefore need a *text editor*.
- The choice of a text editor leads to religious arguments. My advice: try several and pick whichever feels best for you.
- Editors tend to be generic (i.e. for all languages) or specific (for one language). The latter tend to be better at editing 'their' language, but then you need several text editors. I prefer to have one text editor which is OK for lots of languages.

Editing text

- Programming involves editing text. You therefore need a *text editor*.
- The choice of a text editor leads to religious arguments. My advice: try several and pick whichever feels best for you.
- Editors tend to be generic (i.e. for all languages) or specific (for one language). The latter tend to be better at editing 'their' language, but then you need several text editors. I prefer to have one text editor which is OK for lots of languages.
- I use Vim, which is probably the most common Unix editor these days; Emacs is a close #2.

Editing text

- Programming involves editing text. You therefore need a *text editor*.
- The choice of a text editor leads to religious arguments. My advice: try several and pick whichever feels best for you.
- Editors tend to be generic (i.e. for all languages) or specific (for one language). The latter tend to be better at editing 'their' language, but then you need several text editors. I prefer to have one text editor which is OK for lots of languages.
- I use Vim, which is probably the most common Unix editor these days; Emacs is a close #2.
- Other common choices: Atom, Eclipse (mostly for Java), Notepad++, Sublime, Idle (a Python editor, which is bundled with Python), TextMate (OS X only), Visual Studio Code.

What is Python?

- Java is a *statically typed* programming language.
- Python is a *dynamically typed* programming language.

What is Python?

- Java is a *statically typed* programming language.
- Python is a *dynamically typed* programming language.
- Python programs do not require as much information to be explicitly written as their Java equivalents.
- Python is more flexible at run-time than Java but provides fewer compile-time guarantees.

What is Python?

- Java is a *statically typed* programming language.
- Python is a *dynamically typed* programming language.
- Python programs do not require as much information to be explicitly written as their Java equivalents.
- Python is more flexible at run-time than Java but provides fewer compile-time guarantees.
- e.g. `2+"foo"` is rejected by `javac` but is allowed to execute by Python (leading to a run-time `TypeError` exception).

What is Python?

- Java is a *statically typed* programming language.
- Python is a *dynamically typed* programming language.
- Python programs do not require as much information to be explicitly written as their Java equivalents.
- Python is more flexible at run-time than Java but provides fewer compile-time guarantees.
- e.g. `2+"foo"` is rejected by `javac` but is allowed to execute by Python (leading to a run-time `TypeError` exception).
- Dynamic and static typing need to be part of your toolkit.

Python versions

- Two different, semi-incompatible, versions of Python:
  1. Python 2.x: the ‘old but mainstream’ option. Officially deprecated.
  2. Python 3.x: the ‘new but less common’ option. Officially supported.
- In essence, Python 3 ‘fixes’ some flaws in Python 2. Some of those fixes break old programs.

Python versions

- Two different, semi-incompatible, versions of Python:
  1. Python 2.x: the ‘old but mainstream’ option. Officially deprecated.
  2. Python 3.x: the ‘new but less common’ option. Officially supported.
- In essence, Python 3 ‘fixes’ some flaws in Python 2. Some of those fixes break old programs.
- **This version schism is deeply unfortunate.**

Python versions

- Two different, semi-incompatible, versions of Python:
  1. Python 2.x: the ‘old but mainstream’ option. Officially deprecated.
  2. Python 3.x: the ‘new but less common’ option. Officially supported.
- In essence, Python 3 ‘fixes’ some flaws in Python 2. Some of those fixes break old programs.
- **This version schism is deeply unfortunate.**
- Which to use? Python 3 is 9 years old, but only recently has it started to take over from Python 2. New code should be written for Python 3.

Python versions

- Two different, semi-incompatible, versions of Python:
  1. Python 2.x: the ‘old but mainstream’ option. Officially deprecated.
  2. Python 3.x: the ‘new but less common’ option. Officially supported.
- In essence, Python 3 ‘fixes’ some flaws in Python 2. Some of those fixes break old programs.
- **This version schism is deeply unfortunate.**
- Which to use? Python 3 is 9 years old, but only recently has it started to take over from Python 2. New code should be written for Python 3.
- Python 3 is also a (tiny) bit easier to learn.

Python versions

- Two different, semi-incompatible, versions of Python:
  1. Python 2.x: the ‘old but mainstream’ option. Officially deprecated.
  2. Python 3.x: the ‘new but less common’ option. Officially supported.
- In essence, Python 3 ‘fixes’ some flaws in Python 2. Some of those fixes break old programs.
- **This version schism is deeply unfortunate.**
- Which to use? Python 3 is 9 years old, but only recently has it started to take over from Python 2. New code should be written for Python 3.
- Python 3 is also a (tiny) bit easier to learn.
- We’ll use Python 3 today.

Python resources

The vital documentation:

- *The Python Tutorial* covers language features.
- *The Python Standard Library* covers built-in modules (the module index is particularly useful).

Python resources

The vital documentation:

- *The Python Tutorial* covers language features.
- *The Python Standard Library* covers built-in modules (the module index is particularly useful).

You will need both of these at some points today.

The REPL

One quick way of testing one-line snippets out is to use Python's REPL (Read-Eval-Print Loop). If you're on the command line, just type `python` or `python3`; if you're using the Idle editor, it starts in the REPL.

The REPL

One quick way of testing one-line snippets out is to use Python's REPL (Read-Eval-Print Loop). If you're on the command line, just type `python` or `python3`; if you're using the Idle editor, it starts in the REPL.

Exercises:

1. Evaluate the expression `2 + 3 * 4`.
2. Strings are enclosed between quotes `"this is a string"`. What happens if you multiple a string with an integer?

The basics

Differences from Java:

- Classes are not required, nor is a `main` method.
- Statements are not followed by a semi-colon.
- Types don't have to be written out.
- Code can be written at the top-level of a file (i.e. outside a class).
- `print` is a global function that prints strings to screen.

The basics

Differences from Java:

- Classes are not required, nor is a `main` method.
- Statements are not followed by a semi-colon.
- Types don't have to be written out.
- Code can be written at the top-level of a file (i.e. outside a class).
- `print` is a global function that prints strings to screen.

Exercises:

1. Write a program which prints out `Hello world!` in Python. Put it in a file `hello.py` and call it by running `python3 hello.py`.
2. Assign the string `Hello world!` to a variable then print out the contents of the variable.

Basic datatypes

- Integers look and feel similar to Java.
- Lists are nicer with dedicated syntax:
  - `lst=[1, 2, 3]` is a list with 3 integer elements.
  - First element: `lst[0]` Last element: `lst[-1]`
  - Delete an element: `del lst[0]`
  - Set an element: `lst[0] = 4`
  - Add an element to the end: `lst.append(5)`

Basic datatypes

- Integers look and feel similar to Java.
- Lists are nicer with dedicated syntax:
  - `lst=[1, 2, 3]` is a list with 3 integer elements.
  - First element: `lst[0]` Last element: `lst[-1]`
  - Delete an element: `del lst[0]`
  - Set an element: `lst[0] = 4`
  - Add an element to the end: `lst.append(5)`

Exercises:

1. Write a program which assigns the empty list to the variable `lst`.
2. Add (in order) the elements `3, 2, 1`.
3. Print out the second and (using -ve indices) last-but-one elements.
4. Sort the list (hint: read 'sequence types' in *the Python standard library*).

IO

- Print out the contents of a file by assigning a file object to `f`:

```
with open("filename") as f: # Open a file for reading
  print(f.read()) # Print out the contents of the file
```

`f` can only be read in the `with` block. Notice that brackets are not used for blocks: Python is *whitespace sensitive*.

IO

- Print out the contents of a file by assigning a file object to `f`:

```
with open("filename") as f: # Open a file for reading
  print(f.read()) # Print out the contents of the file
```

`f` can only be read in the `with` block. Notice that brackets are not used for blocks: Python is *whitespace sensitive*.

- Print out a file line-by-line:

```
with open("filename") as f:
  for line in f: # Read a line and assign it to line
    print(line)
```

IO

- Print out the contents of a file by assigning a file object to `f`:

```
with open("filename") as f: # Open a file for reading
  print(f.read()) # Print out the contents of the file
```

`f` can only be read in the `with` block. Notice that brackets are not used for blocks: Python is *whitespace sensitive*.

- Print out a file line-by-line:

```
with open("filename") as f:
  for line in f: # Read a line and assign it to line
    print(line)
```

Exercises:

1. Create a file `x.txt` with the contents `i d a c` (each on a different line).
2. Read the contents of `x.txt` into a list, sort it, then print it.
3. What does the `strip()` method do on strings? Where is it useful in your program?

Conditionals

- `if` statements have a mandatory condition, a ‘true’ clause, zero or more `elif` clauses and (optionally) an `else` clause:

```
if a < b: # True branch
  ...
elif b == 10: # elif branch
  ...
else: # else branch
  ...
```

Functions

- Functions have a name and zero or more parameters e.g:

```
def f(a):
  print(a)
  return 2 * a
```

Functions

- Functions have a name and zero or more parameters e.g:

```
def f(a):
  print(a)
  return 2 * a
```

Exercises:

1. Write a function `mymax` which takes two parameters and returns the biggest. Test it with `mymax(1, 3)` and `mymax(2, -1)`.
2. What happens if you do `mymax("2", 3)`?
3. Write a function `myadd` which takes two parameters and adds them together. What happens if you do `myadd("2", 3)`?

Classes

- Classes have a name, (optionally) an `__init__` method (similar to a Java constructor), and (optionally) other methods:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def older_than(self, other):
    return self.age > other.age

a = Person("Bob", 42)
b = Person("John", 55)
print(a.age, a.older_than(b))
```

Classes

- Classes have a name, (optionally) an `__init__` method (similar to a Java constructor), and (optionally) other methods:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def older_than(self, other):
    return self.age > other.age

a = Person("Bob", 42)
b = Person("John", 55)
print(a.age, a.older_than(b))
```

Exercises:

1. Create a class `SortedFile` which takes in a filename, reads it line by line and sorts them; and stores the filename and sorted contents as attributes. `SortedFile("path").contents` should evaluate to the sorted contents.

Modules

- `import m` makes the module `m` available in the current file.
- `import p.n` makes the module `n` (which is a module of the package `p`) available in the current file.
- `from p.n import fc` makes the function/class `fc` from the module `p.n` available in the current file.
- Once imported, modules look like normal objects.

Modules

- `import m` makes the module `m` available in the current file.
- `import p.n` makes the module `n` (which is a module of the package `p`) available in the current file.
- `from p.n import fc` makes the function/class `fc` from the module `p.n` available in the current file.
- Once imported, modules look like normal objects.

Exercises:

1. Write a program `sort.py` which imports the `sys` module, sorts its command line arguments (`sys.argv`) and prints them. Use e.g. `python3 sort.py i a e c`.
2. Use the `urlopen` function in the `urllib.request` module to read the contents of `http://www.kcl.ac.uk/` and print them to screen.

Other useful things

- You can convert objects to a string with `str(o)` (e.g. `str(1)`).

Other useful things

- You can convert objects to a string with `str(o)` (e.g. `str(1)`).
- A tuple `("a", "b")` is similar to a list `["a", "b"]` except the tuple is *immutable*.

Other useful things

- You can convert objects to a string with `str(o)` (e.g. `str(1)`).
- A tuple `("a", "b")` is similar to a list `["a", "b"]` except the tuple is *immutable*.
- Test for an element `e` in a collection `c` with `e in c` (returns `True` / `False`).

Inheritance

- `class A(B)` defines a class `A` which *inherits* from `B`. `A` contains all of `B`'s methods plus any new ones it defines.
- We can *override* methods:

```
class A(B):
  def __init__(self):
    super().__init__() # Allow the superclass to initialise
    self.a = [] # Add a new attribute
```

Inheritance

- `class A(B)` defines a class `A` which *inherits* from `B`. `A` contains all of `B`'s methods plus any new ones it defines.
- We can *override* methods:

```
class A(B):
  def __init__(self):
    super().__init__() # Allow the superclass to initialise
    self.a = [] # Add a new attribute
```

Exercises:

1. Make a class `Link_Finder` which inherits from `html.parser.HTMLParser` and overrides the `handle_starttag` method. Print out all links encountered.
2. Alter `Link_Finder` so that it stores each attribute found in an attribute `links`. Use it thus:

```
lf = Link_Finder("http://www.kcl.ac.uk/")
print(lf.links)
```

Exercise

Write a web spider which crawls over a website. For each page searched, print out the number of links in it, and recursively crawl those links. Print out the total number of pages crawled at the end.

```
$ python3 spider.py https://www.kcl.ac.uk/nms/depts/informatics/
https://www.kcl.ac.uk/nms/depts/informatics/
  <processed 225 links>
https://www.kcl.ac.uk/nms/depts/informatics/about/index.aspx
  <processed 227 links>
...
https://www.kcl.ac.uk/nms/depts/informatics/people/atoz/LoukidesG.aspx
  <processed 221 links>
Total: 287 links
```

Some hints to start you off:

- Keep a list of all the pages you haven't yet crawled. While it's not empty, you still have pages to crawl.
- Be careful not to crawl the same page twice.
- `HTMLParser` can only parse HTML (and not PNGs etc.).

Post-session exercises

Try these (roughly in order):

- Download and play with PyPy.
- Experiment with list comprehensions e.g. `[x*2 for x in [...]]`.
- Experiment with dictionaries e.g. `{"bob":42, "joe":55}`.
- Experiment with making your own modules and packages.
- Have a look at external libraries like `matplotlib` and `numpy`.