Introduction to Programming with Python

Clinton Roy

VALA

April 9, 2019

Outline

Introduction to Myself

- Technical Experience:
  - Used Python for twenty mumble years
  - Wide variety of research and commercial groups
- Organisational Experience
  - Local User group
  - Australian Python conference in Brisbane
  - Volunteer at other Open events:
    - Health Hack, Library Hack, Gov Hack
- Teaching Experience:
  - Conference Speaking and Tutorials
  - CoderDojo
  - Software Carpentry

Introduction to this Class

- Take away skills:
  - Fundamental knowledge of:
    - all programming languages
    - of the Python language
    - of the Python ecosystem
- Tried to use non-technical language
- Instant gratification, use the interpreter
- Extend the examples:
  - add, remove, change arguments!
  - don't move on till you get an error!
- Self directed, internal motivation

Design Goals of Python

- Takes care of a lot of details for you
- To be fast and easy to learn
- Low cognitive load, lets you work on your problem
- Does not lock things down
- Minimise eye strain

Python Details

- Professional programming language used all over the world in many industries
- It's Open Source, your skills are portable.
- There are lots of implementations of Python, we're only looking at one, but 99% of today is useful to all
- Comes with Linux. Older versions come bundled with Apple. Easyish to install on Windows.

Fundamentals Python Concepts

- Everything is an object
  - An object is data and related methods
- Some objects change, some objects don't
- Easy to use data structures

Fundamental Programming Concepts

- Computers run a lot of tiny steps very quickly.
  - Move this bit of memory into the cpu
  - Move this other bit of memory into the CPU
  - Add these two numbers in the CPU
  - Put the result back into memory
- Most programming comes down to organising steps:
  1. Doing one step after another
  2. Repeating steps
  3. Choosing between two steps
  4. Grouping steps
- Variables and assignment
  - A box named anything
  - Spreadsheet Cells

Helping hands

- For the most part, ignore methods that begin with double underscore

```
> s = "some string"
> print(s)
> type(s)
> help(str)
```

Example steps

- assignment

```
> angle = 30
> a, b = "a", "b"
```

- function calls

```
> min(10, 3)
```

- method calls

```
> pancake.flip()
```

- maths

```
> 10 + 3.4
```

Grouping of Steps

- functions

```
> def excited(message):
>     print(message + "!!!")
```

- classes

```
> class Pancake:
>     def flip(self):
>         self.flipped = True
```

- files
- libraries

Python Data Structures

- atoms: numbers, strings
- molecules: lists, dictionaries
- mutable or immutable

Numbers

- Immutable
- Whole numbers, floating point

```
> 123
> 3.14
```

- For more fun, Decimal and Fraction

Number Methods

```
> 1 + 1
> 3 - 4
> 4 * 2
> 2 ** 4
> 8 / 3
```

Strings

- Immutable
- Letters in between quotes

```
> 'letters in between single quotes'
> "letters in between double quotes"
> """letters in between triple quotes"""
```

String Methods

```
> "joining" + " " + "strings"
> "needle" in "a haystack"
> "one two three".index("two")
> "one to three".split()
```

Lists

- Mutable

```
> l = ["a", "b", "c"]
> l.append("d")
> ["one", "two", "three"] + [4, 5, 6]
```

List Methods

```
> l = [5, 4, 5, 3, 5, 2, 1, 5]
>
> l.sort()
>
> l.count(5)
```

Dictionaries

- Mutable
- An association between a key and a value
- Keys must be immutable

```
> d = {"key1": "value1", "key2": "value2"}
>
> d["key3"] = "value3"       # Adding an association
>
> d["key1"]                  # Asking for an association
```

Dictionary Example

```
> thesaurus = {"red" : ["scarlet", "rosy", "ruddy"],
>              "blue" : ["azure", "navy", "cobalt"]}
>
> thesaurus["red"]
```

Other data structures

- Tuples (immutable lists)
- Sets
- Queues
- Heaps
- ...

If Statement

```
> if "needle" in ["haystack"]:
>   print("found the needle!")
> else:
>   print("did not find the needle")
```

- Expressions Examples

```
> a, b = 10, 11
> a == b     # equals
> a > b      # greater than
> a < b      # less than
```

For Loops

- Loop through a data structure

```
> for element in ["a", "b", "c"]:
>     print(element)
```

- Loop through some numbers

```
> for i in range(10):
>     print(i)
```

A more complicated example

```
> upper, lower, other = [], [], []
> for element in ["one", "TWO", "three", "4"]:
>     if element.isupper():
>         upper.append(element)
>     elif element.islower():
>         lower.append(element)
>     else:
>         other.append(element)
```

Other loops

- While

```
> a = 0
> while a < 10:
>     print(a)
>     a = a + 1
```

Functions

- Let you reuse a block of code

```
> def even_stevens(number):
>     if number % 2 == 0:
>         return True
>     else:
>         return False
```

Classes and Objects

- Lets you organise data and methods together

```
> class Pancake:
>     def __init__(self, batter_ml):
>         self.size = batter_ml
>         self.flipped = False
>
>     def flip(self):
>         self.flipped = True
>
> p = Pancake(130)
> p
> p.size
> p.flipped
> p.flip()
> p.flipped
```

Library use

```
> import random
> random.randint(1, 100)
```

Module List

```
> help() # Then "modules"
```

Resources

- These notes: Copyright © 2019 Clinton Roy

`http://github.com/clintonroy/slq2017python/tree/vala19`

- Websites
  - `http://python.org/`
  - `http://jupyter.org/`
- Books
  - Automate the Boring Stuff with Python 
    `https://automatetheboringstuff.com`
- Conferences
  - PyCon Au, PyCon NZ
  - Videos on Youtube
- Software Carpentry groups
- Podcasts
  - From Python import podcast
  - Podcast.__init__
  - Python Bytes
  - Talk Python to Me