Introduction to Programming with Python

Clinton Roy

The Edge, State Library of Queensland

October 12, 2017

Outline

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
- Self directed, internal motivation

Notes on these Notes

Introduction to Myself

- I've:
  - Used Python for twenty mumble years
  - Run the local user group for years
  - Run the Australian Python conference in Brisbane twice, helped out in other cities
  - Help out at many Open Source and Open Data events:
    - Health Hack, Library Hack, Gov Hack
- Taught at Coder Dojo and Software Carpentry

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

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

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.

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

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, numbers
- molecules: tuples, 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
```

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()
```

Tuples

- Immutable

```
> x, y, z = 5, 12
> two_dimensions = (x, y)
> three_dimensions = (x, y, z)
> ("one", 2, 3.0)
```

Lists

- Mutable

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

List Methods

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

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

- Sets
- Queues
- Heaps
- ...

If Statement

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

- Expressions Examples

```
> a = 10
> b = 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)
```

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 flipped(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

- Python.org
- Books
  - Automate the Boring Stuff with Python 🅭🅯🄏🅎 `https://automatetheboringstuff.com`
- Users Group
  - Brisbane Python Users Group
  - Humbug
- Conferences
  - PyCon Au, PyCon NZ
  - Videos on Youtube
- Software Carpentry groups at UQ, QUT, Griffith
- Podcasts
  - From Python import podcast
  - Podcast.__init__
  - Python Bytes
  - Talk Python to Me