Introduction to Professional Python

Professional Python

- Faced-paced coverage of core Python.
- Assumes you know programming principles, not necessarily in Python
- Goes deeper into the Python language than a Python-based CS1 course
- The video for each lesson is about 30 minutes.
  - Each lesson should take you 45-60 minutes if you pause the video and do the active reviews when asked.
- Each exercise should take you an hour or less.
- Projects should take you two to 10 hours.

If you do each lesson – watching the video and pausing to do the active reviews – and at least one exercise after each lesson, you will have a firm grasp of Python. Doing the projects as well will make you an even stronger Python programmer ready to join a professional team as a junior programmer.

- Altogether this course should take you 20 to 40 hours.

Python gives you wings!

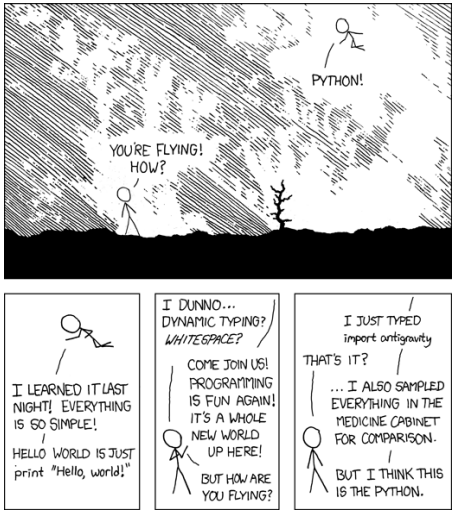

Figure 1: Python Wings

http://xkcd.com/353/

The Python Language

- Python is a general-purpose programming language, meaning you can write any kind of program in Python
  - A *domain-specific language* is designed for one application. E.g., SQL is just for manipulating relational databases.
- Python is interpreted, meaning you can run programs directly after you write them; you don't have to compile programs to some intermediate form for the operating system or a virtual machine to execute.
- Python is a great "glue" language; Python programs often bring together disparate components to do a coherent task.
  - One particular kind of glue is Python's killer feature for data science: easy to create Python bindings for libraries written in other languages
  - Data science libraries, e.g., NumPy, TensorFlow, are written high-performance languages like C and C++
  - Python provides a more comfortable way to use high-performance libraries

The coolest thing about Python ...

The Python Name

Figure 2: Flying Circus

https://en.wikipedia.org/w/index.php?curid=6130072

Python was named for Monty Python, of which Python's creator, Guido van Rossum, is a big fan. You don't have to be a fan, but it helps.

The `python3` Program

Practically speaking, Python is a program on your computer that interprets Python programs and statements.

- You can ask `python3` a question without running any Python code. For example, this is how you ask which version of Python is installed (Note: the `$` character is the command prompt in the Unix Bash shell. The Windows command prompt is `C:\>`.):

```
1 $ python3 --version
2 Python 3.8.10
```

  If you get some other response, like command not found, then you haven't properly installed Python.

Executing Python Code

Three common ways to run Python code:

1. Scripts – files containing Python code – executed on the command line:

```
$ python3 myprogram.py
```

2. Execute statements and expressions in the Python shell/interactive interpreter (commonly called a REPL for "Read-Eval-Print Loop"):

```
$ python3
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> "Hello, world!"
'Hello, world!'
```

To exit the Python shell type `exit()` and hit return, or type Ctrl-D on Linux/Unix, or Ctrl-Z on Windows.

3. In Jupyter Notebooks, which we'll use in the Data Manipulation course.

You can also run short Python code snippets on the command line using the `-c` option:

```
$ python3 -c "print(2 + 3)"
5
```

Hello, Python

Since Kernighan and Ritchie's *The C Programming Language* it's customary for your first program in a new language to be "Hello, world!" We'll keep that tradition.

Active Review

- Create a new file named `hello.py` and add the following line to it, and save it:

```
1 print("Hello, world!")
```

- Then open your OS command shell (terminal – not a Python REPL), go to the directory where you saved `hello.py` and enter:

```
1 $ python3 hello.py
```

Hello, world! will be printed to the console on the next line.

The Python REPL

Invoke the Python interactive shell by entering `python3` at your command shell's prompt without any arguments:

```
$ python3
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

`>>>` is the command prompt for the Python REPL.

- REPL stands for *R*ead *E*val *P*rint *L*oop:
  1. *Read* an expression or statement at the command prompt,
  2. *Evaluate* the expression or execute the statement,
  3. *Print* the result to the console, and
  4. *Loop* back to *Read* step

We'll spend a lot of time in the REPL, but since this course is intended as a fast-paced introduction to Python for professional programmers, we'll use the iPython REPL.

iPython

Two modes:

1. Interactive shell
   - Replacement for `python3` REPL
2. Jupyter notebook kernel
   - Interactive web-based documents mixing text, executable code, graphics

In this course we'll only use iPython as a REPL. Since iPython is a third-party package, we need to install it before we can use it. Enter this on your OS shell's command line (not Python REPL):

```
1 pip3 install ipython
```

We'll learn about `pip3` in the lesson on modules and programs.

iPython Shell History

Active Review

In your OS command shell, run `ipython` and type in the following (on the `In` lines) to get a feel for using iPython.

```
In [1]: ['Sage', 'Thyme', 'Oregano', 'Posh']
Out[1]: ['Sage', 'Thyme', 'Oregano', 'Posh']

In [2]: type(In[1])
Out[2]: str

In [3]: type(Out[1])
Out[3]: list

In [4]: spices = Out[1]

In [5]: spices
Out[5]: ['Sage', 'Thyme', 'Oregano', 'Posh']

In [6]: spices is Out[1]
Out[6]: True
```

Notice that every input is contained in the `In` list, and every output is contained in the `Out` dictionary.

iPython Help

Single `?` gives abbreviated version of python's `help`

```
In [7]: def add(a, b):
   ...:     """Return the result of + operation on a and b"""
   ...:     return a + b
   ...:
In [8]: add?
Signature: add(a, b)
Docstring: Return the result of + operation on a and b
File:      ~/cs2316/<ipython-input-7-af5293282e78>
Type:      function
```

Double `??` gives source code, if available.

```
In [9]: add??
Signature: add(a, b)
Source:
def add(a, b):
    """Return the result of + operation on a and b"""
    return a + b
File:      ~/cs2316/<ipython-input-7-af5293282e78>
Type:      function
```

iPython Magic Commands

Special commands provided by iPython, prepended by `%`.

- Run a Python script from within iPython:

```
In [35]: %run people.py
[<Stan, 2008-08-13, 150cm, 45kg>,
 <Kyle, 2008-02-25, 160cm, 50kg>,
 <Cartman, 2008-05-26, 140cm, 100kg>,
 <Kenny, 2009-07-30, 130cm, 40kg>]
```

- Get help with a magic command with `?`

```
In [2]: %cd?
Docstring:
Change the current working directory.

(content elided)

Usage:

  cd 'dir': changes to directory 'dir'.
(additional output elided)
```

Get a list of all magic commands with `%lsmagic`

iPython Shell Commands

Run shell commands by prepending with a `!`

```
In [27]: !ls *.py
fun.py       grades.py    maths.py     people.py    pp.py

In [28]: pyscripts = !ls *.py

In [29]: pyscripts
Out[29]: ['fun.py', 'grades.py', 'maths.py', 'people.py', 'pp.py']
```

iPython provides magic commands for most common shell commands.

iPython Directory Bookmarking

Great time saving feature.

```
In [1]: pwd
Out[1]: '/Users/chris/vcs/github.com/drcscodes/drcs.codes-solutions'

In [2]: %bookmark drcs.codes-solutions
    '/Users/chris/vcs/github.com/drcscodes/drcs.codes-solutions'

In [3]: cd
/Users/chris

In [4]: cd drcs.codes-solutions
(bookmark:drcs.codes-solutions) ->
    /Users/chris/vcs/github.com/drcscodes/drcs.codes-solutions
/Users/chris/vcs/github.com/drcscodes/drcs.codes-solutions
```

iPython Automagic commands

With `automagic` turned on, some shell commands can be run as if they were built into iPython:

```
In [22]: pwd
Out[22]: '/Users/chris/cs2316'

In [23]: ls *.py
fun.py      grades.py   maths.py    people.py   pp.py
```

- Toggle automagic on and off with `%automagic`.
- These commands work with automagic:
  - `%cd`, `%cat`, `%cp`, `%env`, `%ls`, `%man`, `%mkdir`, `%more`, `%mv`, `%pwd`, `%rm`, and `%rmdir`

iPython is nicer than the Python.org REPL, but doctests use the Python.org REPL prompt. For writing doctest examples, iPython offers the `%doctest_mode` magic.

```
In [93]: def dubbel(x: int) -> int:
    ...:     return x * 2
    ...:

In [94]: %doctest_mode
Exception reporting mode: Plain
Doctest mode is: ON
>>> dubbel(3)
6
>>> %doctest_mode
Exception reporting mode: Context
Doctest mode is: OFF

In [97]:
```

Conclusion

- Python is an interpreted general purpose language.
- Python code can be run as programs or interactively in a Python REPL.
- Python is a great glue language.
- Python is fun!