Introduction to Python

Intensive 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

Python gives you wings!

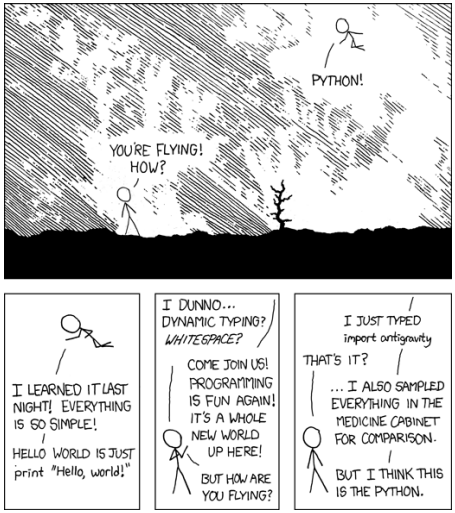

Figure 1: Python Wings

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.

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

- You can run a Python program, which has a .py extension by convention:

```
$ python3 myprogram.py
```

- Or you can invoke the interactive Python shell (sometimes called 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.
>>>
```

To exit the Python shell type Ctrl-D on Linux/Unix, or Ctrl-Z on Windows.

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!"

- Open your text editor, paste the following code into a buffer (or tab or window or whatever your editor calls it), and save it as `hello.py`:

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

- Then open your command shell (terminal on Unix or CMD.exe on Windows), 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.

Interpreting Python Programs

What happens when we enter `python3 hello.py` at an operating system command shell prompt?

1. `python3` tells the OS to load the Python interpreter into memory and run it. `python` is the name of an executable file on your hard disk which your OS can find because its directory is on the `PATH`
2. We invoke `python` with a *command line argument*, which `python3` reads after it starts running
3. Since the command line argument was the name of a file (`hello.py`), the `python3` loads the file and executes the Python code in it.

A Python program, or script, is just a sequence of Python statements and expressions.

The Python REPL

Invoke the Python interactive shell by entering python at your command shell's prompt without any arguments and type in the same line we put in hello.py:

```
$ 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 data analytics, we'll use the iPython REPL.

iPython

Two modes:

- Interactive shell
  - Replacement for `python` REPL
- Jupyter notebook
  - Interactive web-based documents mixing text, executable code, graphics

Before we proceed, make sure your computer is ready (OS shell):

```
1 $ pip3 install ipython
```

iPython Shell History

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

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

`In` is a list, `Out` is a dict.

iPython Help

Single `?` gives abbeviated 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 [3]: %pwd
Out[3]: '/home/chris/vcs/github.com/cs2316/cs2316.github.io/code'

In [4]: %cd
/home/chris

In [5]: %bookmark cs2316code
    ~/chris/vcs/github.com/cs2316/cs2316.github.io/code

In [6]: cd cs2316code
(bookmark:cs2316code) -> ~/chris/vcs/github.com/cs2316/cs2316.github.io/code
/home/chris/vcs/github.com/cs2316/cs2316.github.io/code
```

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 superior to 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!