Introduction to Python

Global Knowledge®

Objectives

- Learn the history of Python
- See the differences between Python 2 and Python 3
- Install and configure Python on your OS
- Become proficient with the interactive shell
- Learn about different implementations of Python
- Choose an IDE
- Discover the Zen of Python

What is Python?

- High-level programming language
- Interpreted (sometimes JIT compiled)
- Object-oriented (especially Python 3)
- Strongly-typed with dynamic semantics
- Syntax emphasizes readability
- Supports modules and packages
- Batteries included (large standard library [1])

A brief history of Python

Guido van Rossum begins development on Python — 1989

Python 2.0 released — 2000

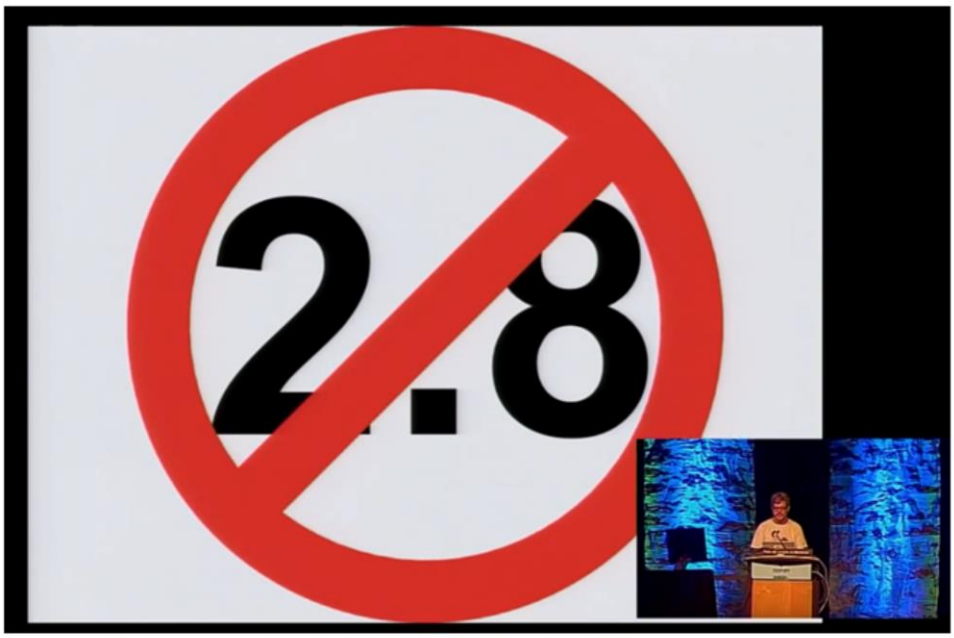
Python 3.0 released — 2008

Get the full history from Guido
http://www.youtube.com/watch?v=ugqu10JV7dk

Python 2 vs. Python 3

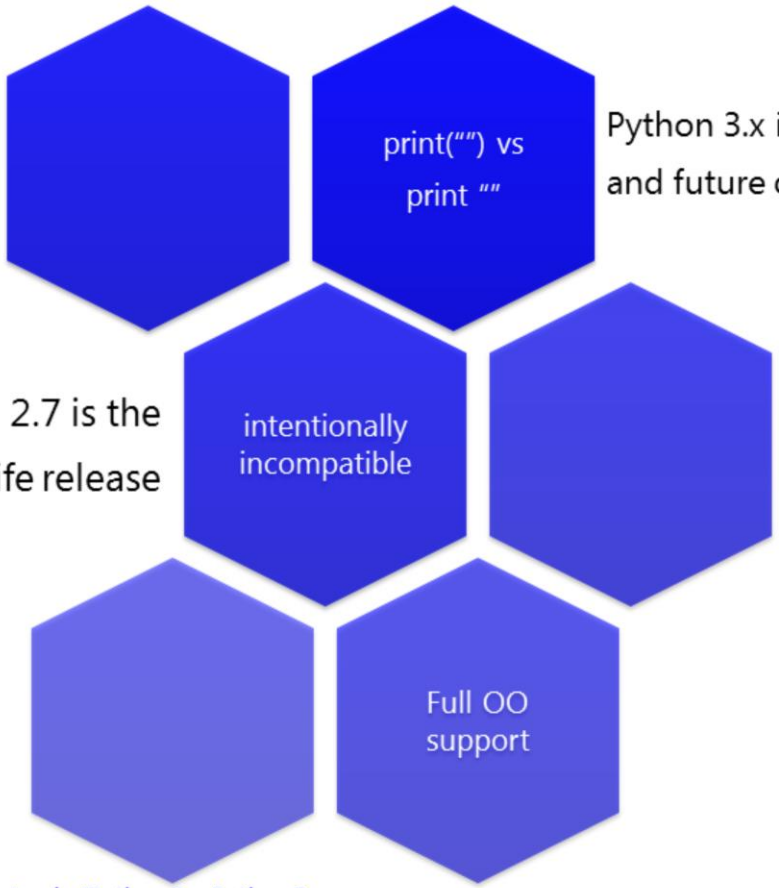
This course focuses on Python 3 because it literally is the future:

- Guido has announced that there will NOT be new features in Python 2
- The syntax / library differences are not major, we discuss some during the course



https://www.youtube.com/watch?v=0Ef9GudbxXY

Python 2 vs. Python 3



https://wiki.python.org/moin/Python2orPython3

Python 3 highlights

- Iterators everywhere
- range() is an iterator (used in for loops with indexes)
- Python 3 uses full OO for classes (all derive from object)
- Different integer math
- All text is Unicode
- Changes to collection classes
- Many features of `__future__` now built-in.
- Old modules removed and some renamed
- New string formatting (via `"somestring".format()`)
- Changes to exception requirements

http://docs.python.org/3/whatsnew/3.0.html

In Python you can add some features from future versions into the current version. For example, to get print as a function into Python 2.7:
from __future__ import print_function

To get Python 3's behavior for division:
from __future__ import division

To get Python 3's unicode literals:
from __future__ import unicode_literals

Getting started

- Installing and configuring Python
  - http://www.python.org/download/
  - Some versions Python come pre-configured on some OS's

Overview of applications

- Python can be used to build a wide variety of applications
  - **Web applications** (e.g. YouTube, Pinterest, Disqus [1])
    - Django
    - Pyramid
    - Flask
    - Web2Py
  - **Glue applications** (infrastructure)
  - **Scripting Extensions** (Civilization 4)
  - **GUI applications**
    - PyQt / PySide
    - TkInter
  - **Cloud**
    - Windows Azure, Google App Engine
    - PiCloud
  - **Scientific Applications**
    - NumPy / SciPy

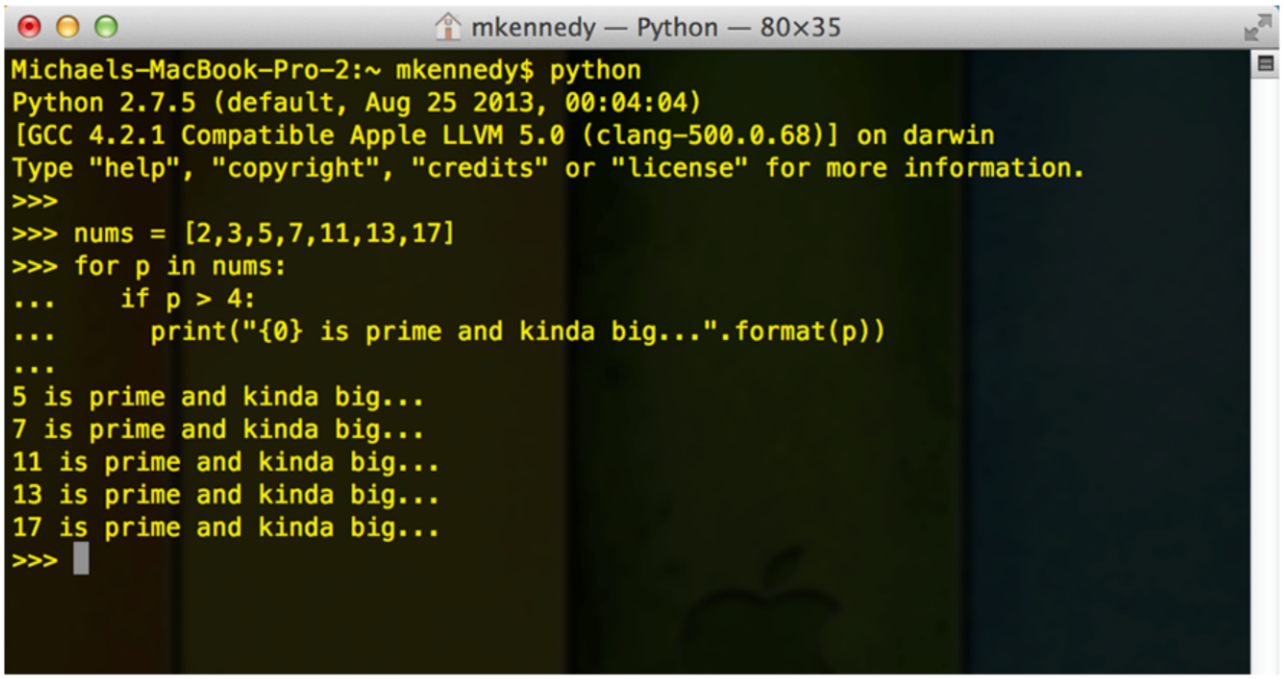
Exploring the standard library

- There are many modules in the standard library. Here are the major functionality areas from

  - Built-in Functions
  - Built-in Types
  - Text Processing Services
  - Data Types
  - Mathematical Modules
  - Functional Programming Modules
  - File and Directory Access
  - Data Persistence
  - Compression and Archiving
  - Common File Formats
  - Cryptographic Services
  - Operating System Services
  - Concurrent Execution
  - Networking
  - Internet Data Handling
  - Structured Markup Processing Tools
  - Internet Protocols and Support
  - Multimedia Services
  - Internationalization
  - Program Frameworks
  - Graphical User Interfaces with Tk
  - Unit testing and mocking
  - Debugging and Profiling
  - Python Runtime Services
  - Custom Python Interpreters
  - Importing Modules
  - Python Language Services
  - Windows Specific Services
  - Unix Specific Services

Using the interactive shell

- Python comes with REPL (read–eval–print loop) interactive language shell.



Using the interpreter [tips]

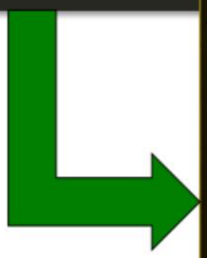
- Language shell is good for experimenting
  - For real programs we use scripts and maybe an IDE
- Tips:
  - Modules and scripts can be imported (e.g. `import pymongo`)
  - Single line expressions and methods can be run
  - Multi line expressions can be entered (... implies more input)
  - Don't forget the spaces for multi lines.

```
>>> for p in nums:
... print(p)
  File "<stdin>", line 2
    print(p)
        ^
IndentationError: expected an indented block
>>>
```

Using the interpreter [text editor]

- For more complex code, you can use a text editor and then paste multiple lines (or use 'real' scripts of course)

```
1  import sys
2
3  def echoMan():
4      print("What do you want to say? ")
5      msg = sys.stdin.readline()
6      print("oh sweet, agreed: {0}".format(msg))
7
8  for i in range(1,5):
9      echoMan()
10
```



```
>>> import sys
>>>
>>> def echoMan():
...     print("What do you want to say? ")
...     msg = sys.stdin.readline()
...     print("oh sweet, agreed: {0}".format(msg))
...
>>> for i in range(1,5):
...     echoMan()
...
What do you want to say?
Python is cool!
oh sweet, agreed: Python is cool!
```

Python implementation

IronPython

Pyston
(from DropBox)

Many more at https://wiki.python.org/moin/PythonImplementations

Python implementation [CPython]

- CPython
  - the reference implementation of Python
  - written in C
  - compiles Python code to intermediate bytecode
  - bytecode is then **interpreted** by a virtual machine
  - required for any packages that rely on C extensions

http://www.python.org/getit/

Python implementation [PyPy]

- PyPy
  - Python compiles (via JIT) to native code
  - can be over 5 times faster than CPython
  - restricted statically-typed subset of the Python language called RPython
  - aims for maximum compatibility with the reference CPython

http://pypy.org/

Python implementation [IronPython]

- IronPython
  - an implementation of Python for the .NET framework
  - access to standard library
  - access to .NET base class library
  - expose Python types and methods to .NET applications
  - Python Tools for Visual Studio integrates IronPython directly into the Visual Studio
  - Python Tools for Visual Studio also supports 'pure' Python

IronPython

http://ironpython.net/

Python implementation [Jython]

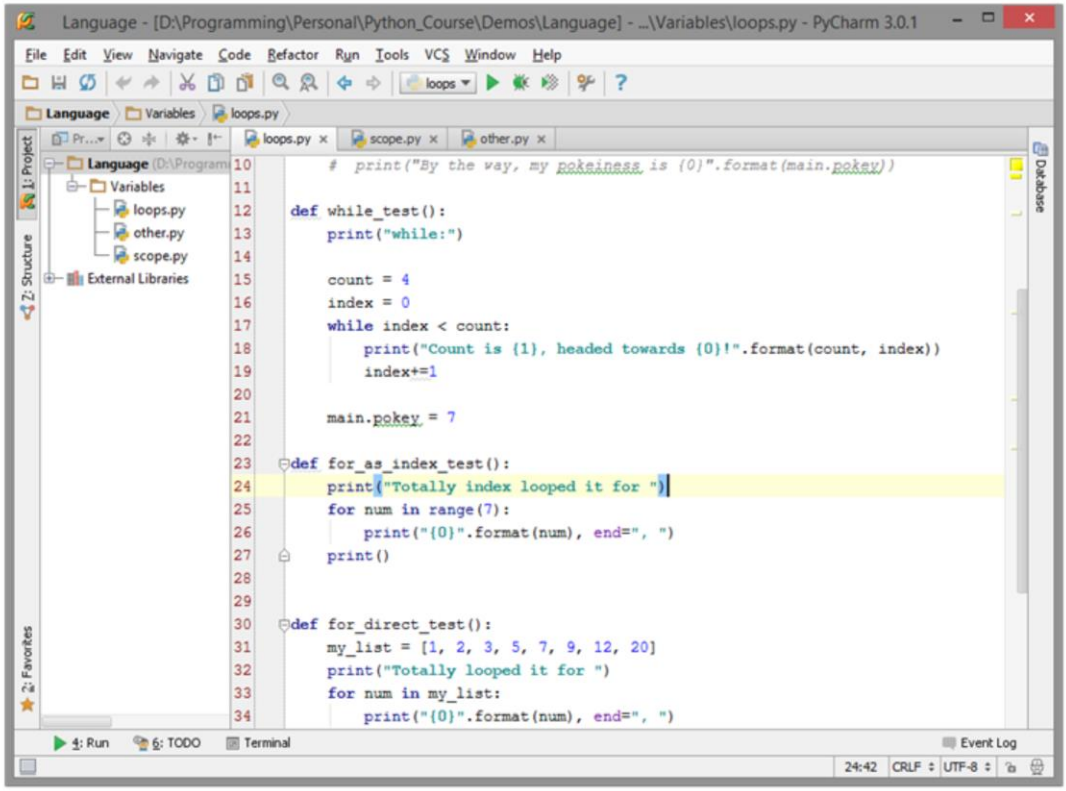
- Jython
  - Python implementation that compiles Python code to Java byte code
  - byte code that is then executed in a JVM
  - access to standard library
  - access to Java base class library
  - excellent for glue between libraries already written in Java

http://www.jython.org/

Choosing an IDE

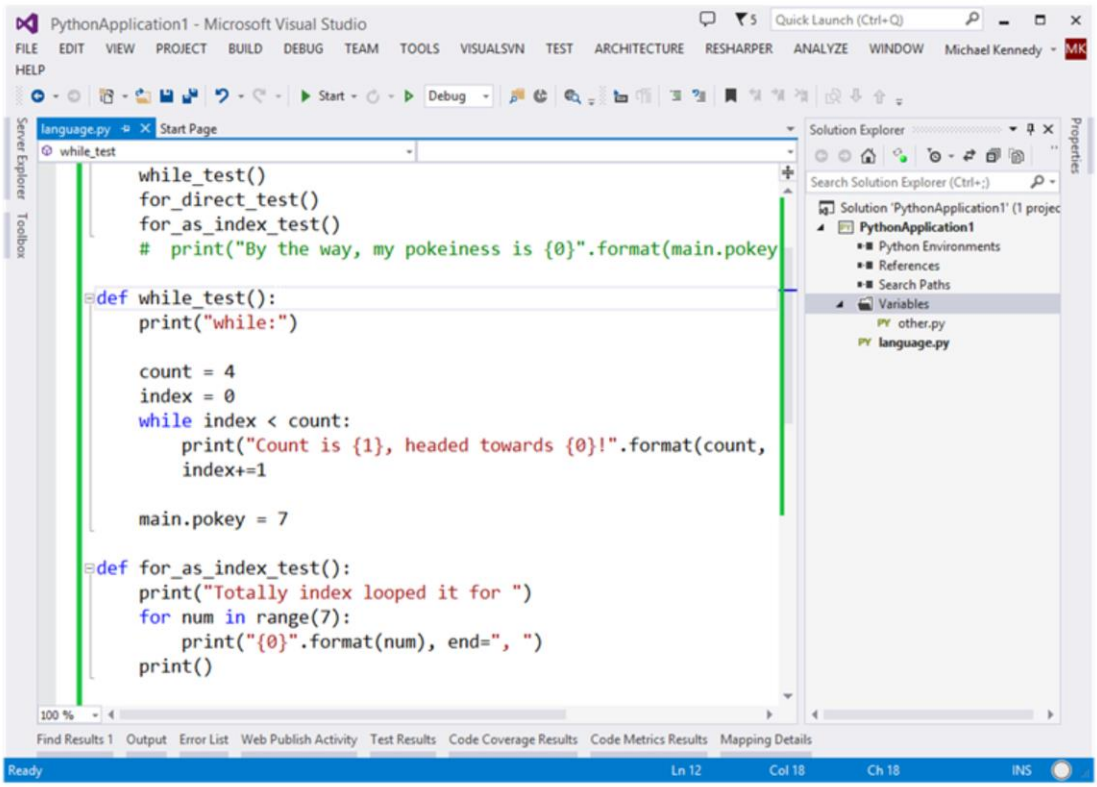
- IDEs have many advantages
  - Quick access to multiple files within a project
  - Debugging via breakpoints and code stepping
  - Creation and management of virtual environments
  - Unit testing
  - Refactoring
  - Code completions (intellisense)
  - Go to definition
  - Framework support (Django, Pyramid, etc.)
  - Code inspection
  - Code navigation
- IDEs are not required
  - Can use a basic text editor
  - Can use full featured editors (e.g. Emacs, Sublime Text, etc.)

Choosing an IDE [PyCharm]



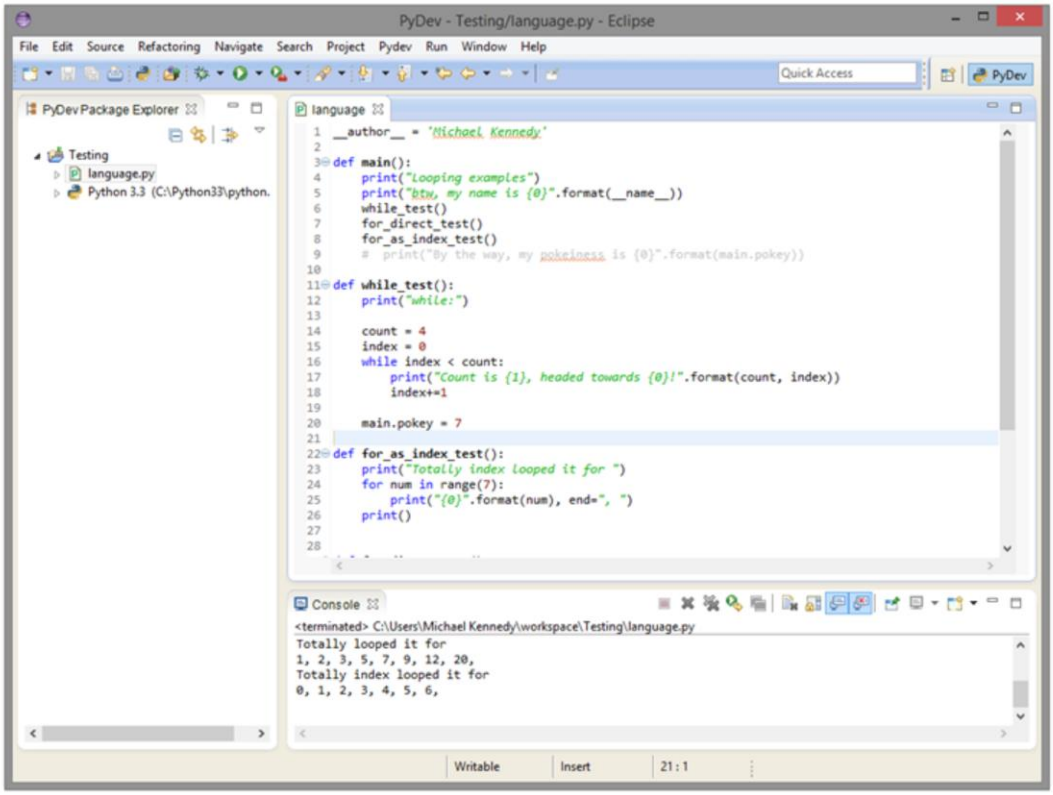
Java-based IDE from JetBrains: http://www.jetbrains.com/pycharm/

Choosing an IDE [Visual Studio + Python Tools]



Runs within Visual Studio: http://pytools.codeplex.com/

Choosing an IDE [PyDev]



Runs within Eclipse: http://pydev.org/

Choosing an IDE [PyDev]

- Installing PyDev into Eclipse
  1. Download and install Eclipse
  2. Help → Install New Software
  3. Click Add… (add repository)
  4. Name: PyDev, Location: http://pydev.org/updates
  5. Check PyDev → PyDev for Eclipse
  6. Next → Next → Agree → Finish
  7. Trust the certificate if prompted
  8. Restart Eclipse when prompted
  9. Window → Open perspective → Other → PyDev
  10. Window → Preferences → PyDev → Interpreters → Python
      - Python 2: Choose Auto-config (OS X and Linux)
      - Python 3: Choose New → Path to Python3(.exe)

Running scripts from command-line [Unix]

- Linux / Unix systems use 'shebangs' to associate files with Python:


```
#!/usr/bin/env python3

print("Hello python 3")
```


```
mkennedy — bash — 80×35
Michaels-MacBook-Pro-2:~ mkennedy$ ~/PycharmProjects/test/test.py
Hello python 3
```

Use /usr/bin/env because the python executable may be in different locations, or you may use virtual environments with their own links to executables.

Running scripts from command-line [Windows #2]

- Python launcher for Windows **allows the script to choose** Python version
  - Installed in **`c:\windows\py.exe`** with Python 3.3+
  - Can be called from the command line:
    - **`py`** launches Python 2.7 if installed.
    - **`py -3`** launches latest Python 3 version.

```
#!/usr/bin/env python3
import sys
print("Hello from Python {}".format(sys.version_info[0]))
```

```
C:\>py program.py
Hello from Python 3
```

```
#!/usr/bin/env python2
import sys
print("Hello from Python {}".format(sys.version_info[0]))
```

```
C:\>py program.py
Hello from Python 2
```

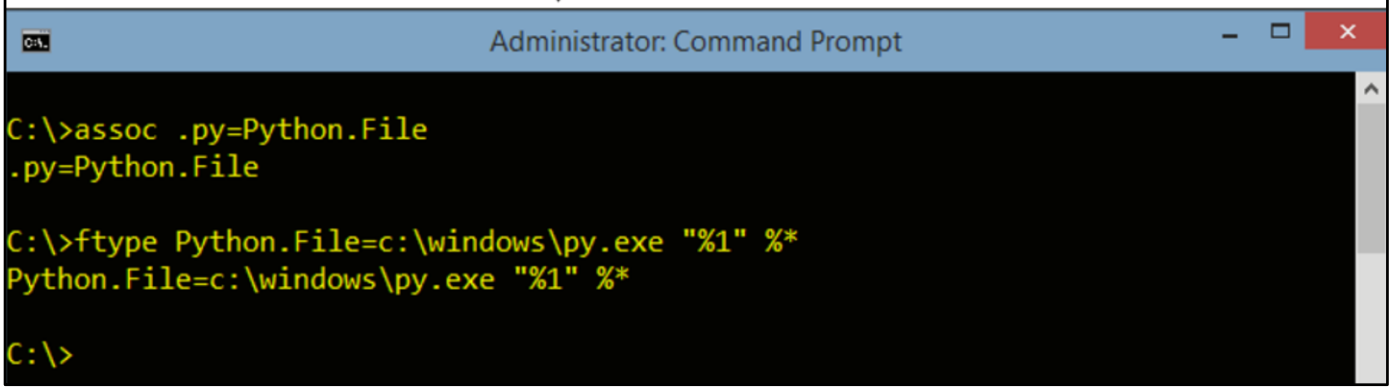
More details about Python Launcher for Windows (py.exe) is here:
https://docs.python.org/3/using/windows.html#python-launcher-for-windows

Running scripts from command-line [Windows]

- Associating Python scripts with the Python runner
  1. **assoc** .py=Python.File
  2. **ftype** Python.File=C:\windows\py.exe "%1" %*

py.exe requires at least Python 3.3 or higher.
Use C:\python3\python.exe to lock to a single version.

Must be run with **elevated privileges**

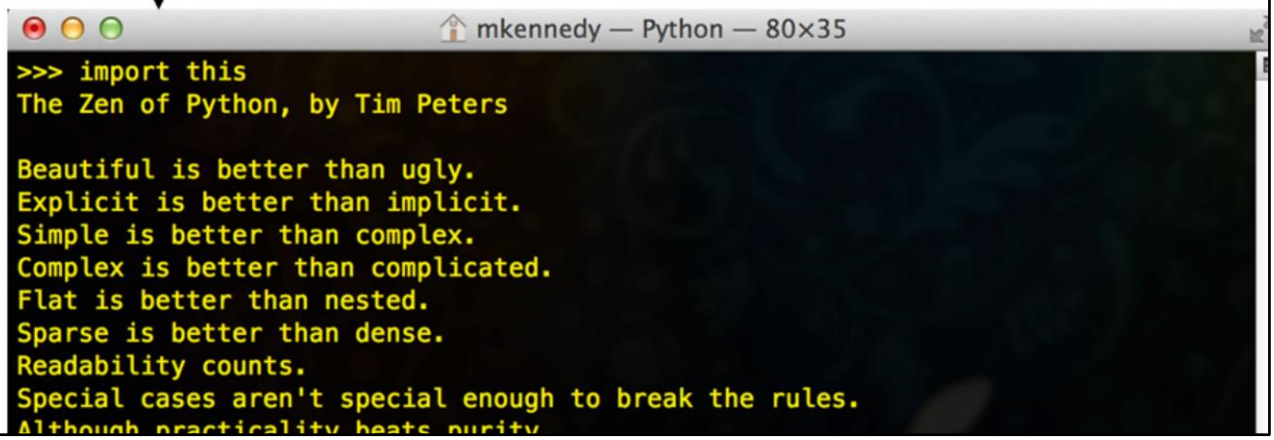


PEP 20 - The Zen of Python

- The Zen of Python
  - guiding principles for Python's design into 20 aphorisms, only 19 of which have been written down.

http://www.python.org/dev/peps/pep-0020/

Never far away with `import this`.



PEP 20 - The Zen of Python [details]

- Beautiful is better than ugly.
- Explicit is better than implicit.

```
import os
Print( os.getcwd() )
```

- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

```
Print("Hello, world!") # that's an entire program
```

PEP 20 - The Zen of Python [more details]

- Special cases aren't special enough to break the rules.

A string of length 1 is not special enough to deserve a dedicated char type.

- Although practicality beats purity.

That's why we have the `chr()` and `ord()` builtins.

- Errors should never pass silently.

```
try:
   import this
except ImportError:
   print( 'this is not available' )
```

- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.

PEP 20 - The Zen of Python [more details]

- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

- Namespaces are one honking great idea -- let's do more of those!

```
import os
print os.getcwd()
```

- There should be one -- and preferably only one -- obvious way to do it.

```
for element in sequence:
   # work with element...
```

- Although that way may not be obvious at first unless you're Dutch.

Summary

- Python was created in 1991 by Guido van Rossum
- Python 3 is cleaner than Python 2
  - Many of the features of Python 3 have been back-ported
- Python 3 has to be installed on OS X and Windows
- Python's interactive shell let's you try ideas quickly
- CPython, IronPython, Jython, Pyston, and PyPy are all implementations of Python
- PyCharm, Visual Studio, and Eclipse are all good IDEs