

The Python Language

- Objectives
 - Learn the basics of the Python language
 - Define code blocks
 - Work with variables
 - Test for true / false using various conditionals
 - Write loops to work with sets of data
 - Use deterministic clean up for recovering resources

The 'shape' of a Python program

- Python defines code blocks (known as suites in Python) using whitespace and colons.

```
def somemethod(name):  
    if name == "Michael":  
        print("Hi old friend")  
    else:  
        print("Nice to meet you")  
        print("My name is ... ")  
  
def main():  
    somemethod()
```

Things to note:

- No semicolons
- Code blocks start with ':'
- Whitespace *really* matters
- There are no braces
- There are no parentheses

Code suites

Variables

- Declaring variables

```
name = "Jeff"  
print( "Hi there " + name )  
name = 42  
print( name )
```

Note:

- You do not declare the type
- Discover current type via **type(var)**
- Compare references with **id(var)** and **var1 is var2**
- Compare values with **var1 == var2**

Variables [scope]

- Variable scope
 - Python does not have strict block scope like many C-based languages
 - Not restricted to the declaring scope
 - Scope is global or function level

```
num1 = 40

if num1 > 10:
    num2 = 2
    print("Num from if: " + str(num2))

print("Looks like the number is " + str(num1 + num2))

# prints 'Looks like the number is 42'
```

Variables [global scope]

- Variable scope
 - **global** keyword can promote scope

```
sharedVal = 3

def method1():
    global sharedVal
    if sharedVal == 3:
        sharedVal = 7

    sharedVal += 1

method1()
print( sharedVal ) # prints 8
```

Comments

- Comments are indicated with the **# character**
- They last for rest of a single line

```
name = "Jeff"  
print( "Hi there " + name ) # use string concat  
num = 42  
# I wouldn't try this one!  
print( "Hi there " + num) # this is an error!
```

Conditionals: Truthiness

- The following are considered **False**
 - **None**
 - **False**
 - zero of any numeric type, for example, `0`, `0.0`
 - any empty sequence, for example, `' '`, `()`, `[]`.
 - any empty mapping, for example, `{}`.
- Everything else is **True**

Conditionals: if statements

- **if** statements are simple suites
- Additional tests are done using **elif** (not **else if**)
- **and** and **or** are words (not symbols, e.g. **&&** and **||**)
- **else** statements can appear at the end

```
if len(name) > 5:
    print('Oh you have a long name!')
elif len(name) == 5:
    print('Let me guess, your name is Sarah?')
elif len(name) == 4 and name[0] == 'T':
    print('Let me guess, your name is Todd?')
else:
    print('Filling out forms must be quick for you!')
```


Conditionals: Ternary statements

- Ternary statements are compressed **if** / **else** suites
- They are meant to be readable rather than concise

```
name = "Jeff"  
val = "short name" if len(name) < 5 else "long name"  
print(val) # prints 'short name'
```

Empty code suites (blocks): Pass statement

- Sometimes you want an empty block
 - maybe you commented out some code
 - maybe you're sketching out the structure
- The **pass** keyword keeps things running

```
if len(name) > 5:  
    pass
```

While loops

- While loops run until a condition becomes false

```
num1 = 1
num2 = 2

while num1 < 100:
    num1 = num1 * num2
    print( num1 )

# prints 2,4,8,16,32,64,128
```

For in loops

- For loops in Python fundamentally work on iterable sets
 - There is no index-based looping construct!
 - Many types are iterable
 - lists, sets, dictionaries, strings, files, classes, ...

```
name = "Jeff"

for ch in name:
    print( ch, end=', ' )

# prints 'J', 'e', 'f', 'f',
```

For in loops [with indexes]

- For loops **can** use an index
 - Uses **range** function
 - But it's less Pythonic
 - range was considered harmful in Python 2.7 (it's not in 3)

```
name = "Jeff"

for i in range( len(name) ):
    print( name[i] )

# prints 'J', 'e', 'f', 'f',
```

Loops and else statements

- All looping constructs support a final clause using **else**
- Only runs if either
 - the loop completes **without** early breaks
 - the loop completes but never runs

```
print("Else loop test")
v = 7
while v < 10:
    v += 1
    print(v)
    if v == 9:
        break
else:
    print("else v is now " + str(v))
```

Consuming libraries

- Python can access functionality from other modules, packages, and libraries using the **import** statement.

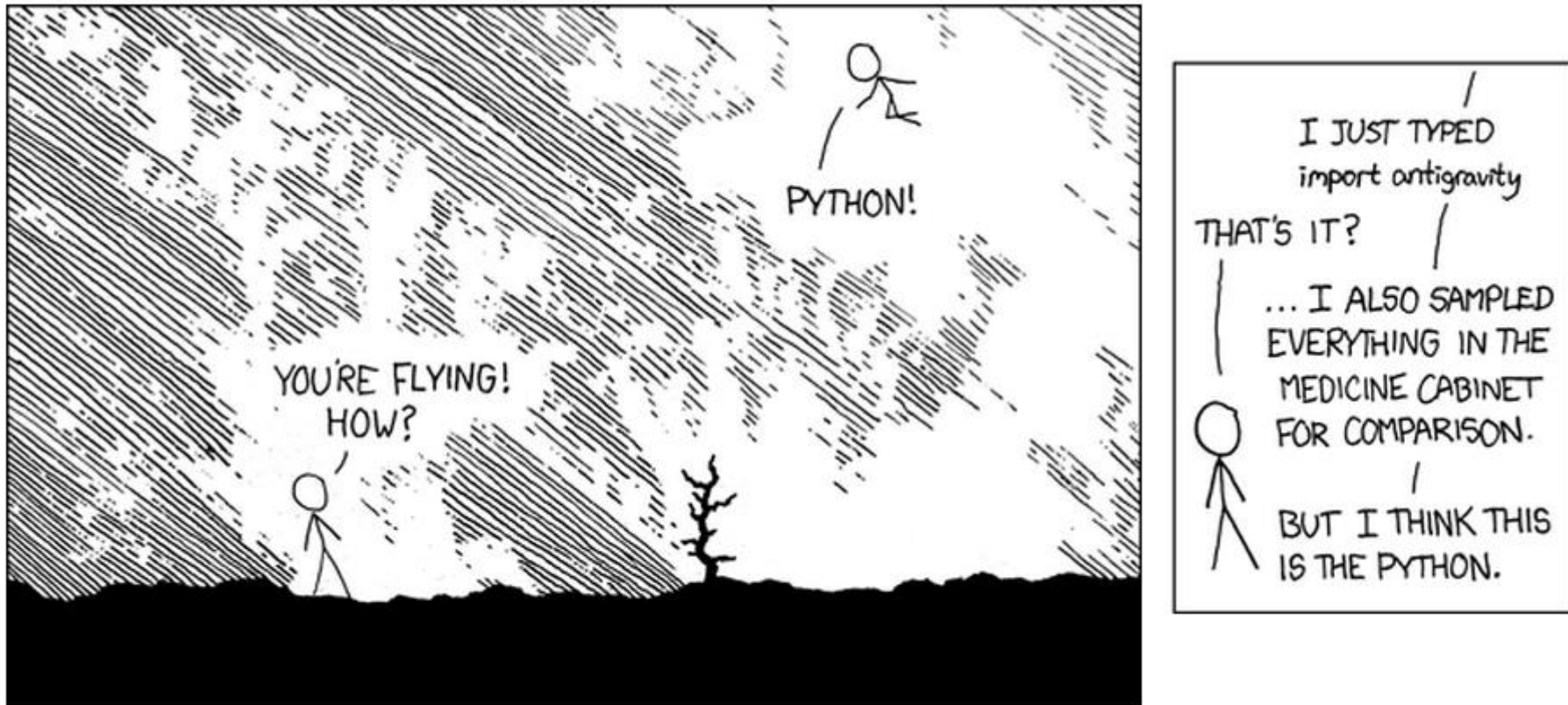


Image credit: XKCD: <http://xkcd.com/353/>

Consuming libraries [import keyword]

- Python can access functionality from other modules, packages, and libraries using the **import** statement.
- Import gives you access to
 - other scripts you have written
 - modules and packages from third parties
 - components of the standard library

Consuming libraries [standard library]

- Accessing the standard library

```
import sys

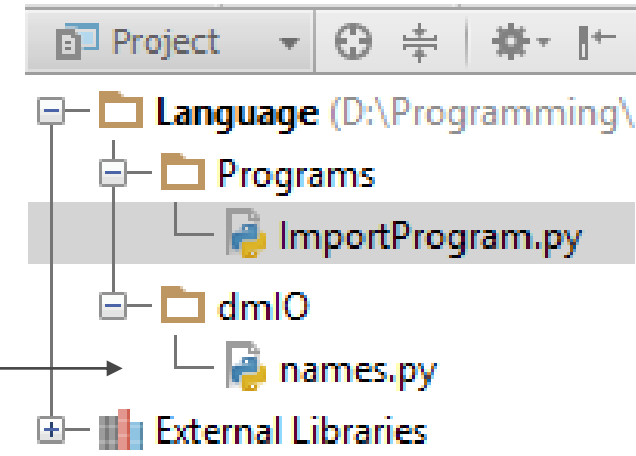
print("Please enter your name: ")
name = sys.stdin.readline()

print("Nice to meet you " + name)
```

Consuming libraries [other scripts]

- Accessing other scripts

Include relative paths for working folder



```
import dmIO.names
```

```
userName = dmIO.names.queryUserName()  
print("Nice to meet you " + userName)
```

```
# Sample execution:
```

```
# Please enter your name: Jeff
```

```
# Nice to meet you Jeff
```

Consuming libraries [importing your script]

- When your scripts are imported, they may run code you did not intend to run
 - Use the `__name__` convention to test if your script is the main script.

```
if __name__ == "__main__":  
    # your code here
```

Consuming libraries [import details]

- Import has several forms

```
import dmIO.names # default: keep namespace
```

```
userName = dmIO.names.queryUserName()  
print("Nice to meet you " + userName)
```

```
from dmIO.names import queryUserName # single method / class  
import
```

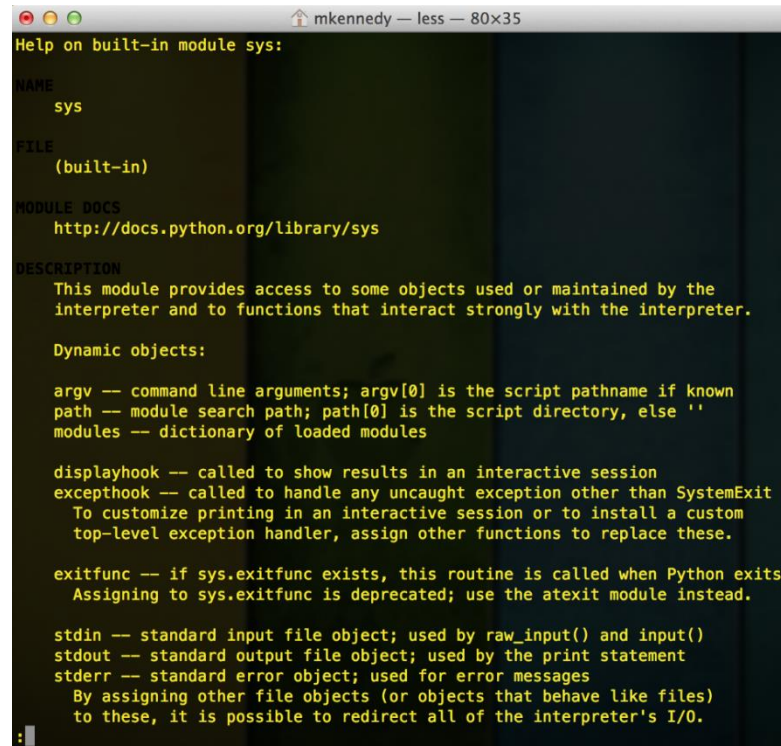
```
userName = queryUserName()  
print("Nice to meet you " + userName)
```

```
from dmIO.names import * # import everything, no namespaces
```

```
userName = queryUserName()  
print("Nice to meet you " + userName)
```

Getting help

- Python has good documentation with examples
 - Visit <http://docs.python.org/3.3/contents.html>
 - Just Google it (typically fastest access to docs.python.org)
 - Type **help(class)** or **help(namespace)** (hint: q to quit)



```
mkennedy — less — 80x35
Help on built-in module sys:

NAME
  sys

FILE
  (built-in)

MODULE DOCS
  http://docs.python.org/library/sys

DESCRIPTION
  This module provides access to some objects used or maintained by the
  interpreter and to functions that interact strongly with the interpreter.

  Dynamic objects:

  argv -- command line arguments; argv[0] is the script pathname if known
  path -- module search path; path[0] is the script directory, else ''
  modules -- dictionary of loaded modules

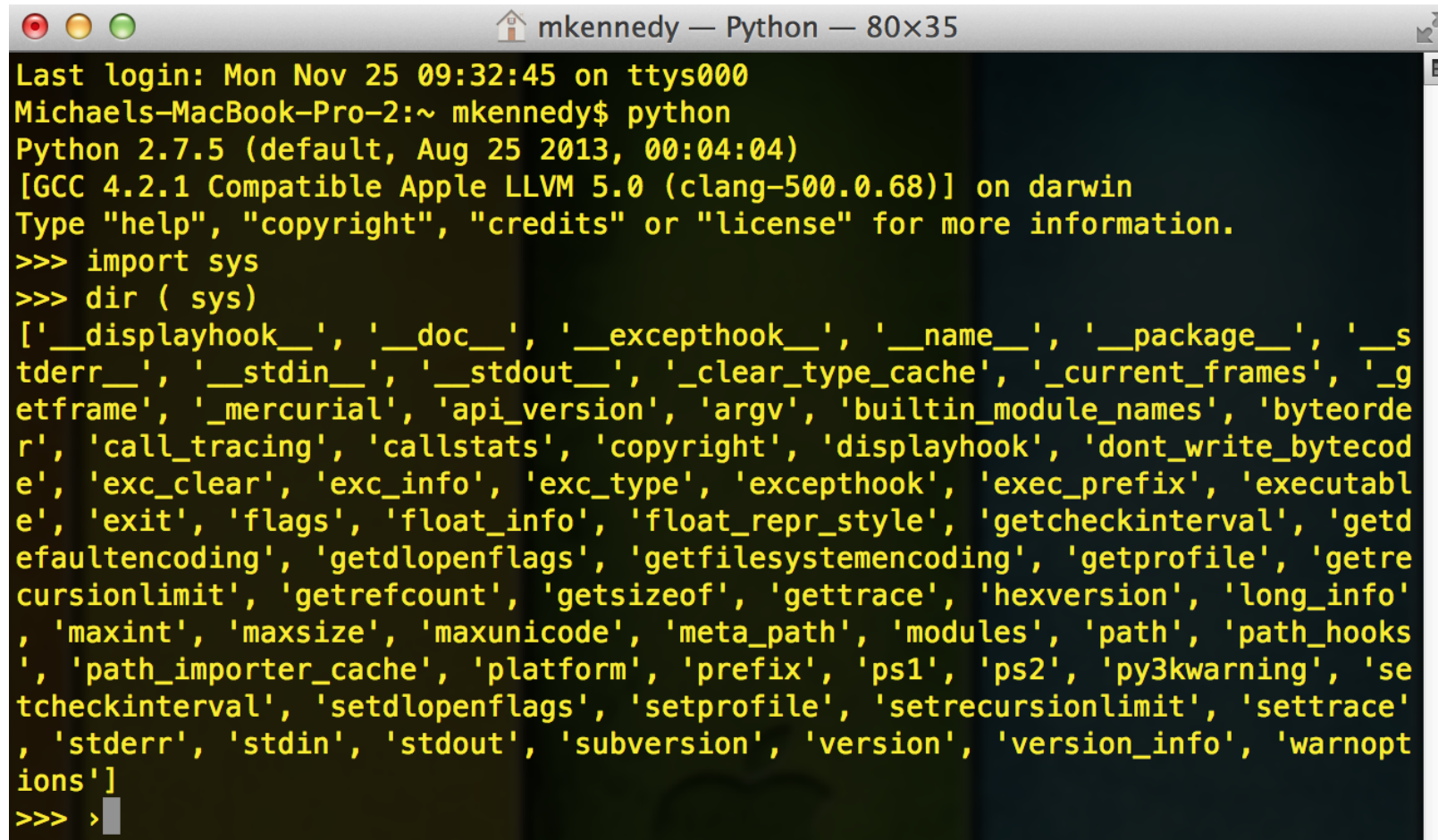
  displayhook -- called to show results in an interactive session
  excepthook -- called to handle any uncaught exception other than SystemExit
  To customize printing in an interactive session or to install a custom
  top-level exception handler, assign other functions to replace these.

  exitfunc -- if sys.exitfunc exists, this routine is called when Python exits
  Assigning to sys.exitfunc is deprecated; use the atexit module instead.

  stdin -- standard input file object; used by raw_input() and input()
  stdout -- standard output file object; used by the print statement
  stderr -- standard error object; used for error messages
  By assigning other file objects (or objects that behave like files)
  to these, it is possible to redirect all of the interpreter's I/O.
```

Getting help

- Not everything is documented, but you can still browse it with the **dir(entity)** command.

A screenshot of a terminal window titled 'mkennedy — Python — 80x35'. The terminal shows the output of running 'python', which includes the Python version (2.7.5), GCC version (4.2.1), and platform (darwin). It then shows the execution of 'import sys' and 'dir(sys)', which outputs a long list of attributes for the sys module, including __displayhook__, __doc__, __excepthook__, __name__, __package__, __stderr__, __stdin__, __stdout__, _clear_type_cache, _current_frames, _getframe, _mercurial, api_version, argv, builtin_module_names, byteorder, call_tracing, callstats, copyright, displayhook, dont_write_bytecode, exc_clear, exc_info, exc_type, excepthook, exec_prefix, executable, exit, flags, float_info, float_repr_style, getcheckinterval, getdefaultencoding, getdlopenflags, getfilesystemencoding, getprofile, getrecursionlimit, getrefcount, getsizeof, gettrace, hexversion, long_info, maxint, maxsize, maxunicode, meta_path, modules, path, path_hooks, path_importer_cache, platform, prefix, ps1, ps2, py3kwarning, setcheckinterval, setdlopenflags, setprofile, setrecursionlimit, settrace, stderr, stdin, stdout, subversion, version, version_info, and warnoptions. The prompt '>>>' is followed by a cursor.

```
mkennedy — Python — 80x35
Last login: Mon Nov 25 09:32:45 on ttys000
Michael's-MacBook-Pro-2:~ mkennedy$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> dir ( sys)
['_displayhook__', '__doc__', '__excepthook__', '__name__', '__package__', '__stderr__', '__stdin__', '__stdout__', '_clear_type_cache', '_current_frames', '_getframe', '_mercurial', 'api_version', 'argv', 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats', 'copyright', 'displayhook', 'dont_write_bytecode', 'exc_clear', 'exc_info', 'exc_type', 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info', 'float_repr_style', 'getcheckinterval', 'getdefaultencoding', 'getdlopenflags', 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof', 'gettrace', 'hexversion', 'long_info', 'maxint', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2', 'py3kwarning', 'setcheckinterval', 'setdlopenflags', 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion', 'version', 'version_info', 'warnoptions']
>>> >
```

Summary

- Python uses whitespace and colons to define blocks
- Variables do not require type definitions
- There are two types of conditionals
- Python does not have an index for loop
- modules are imported using the import keyword