POWERING POTENTIAL

Introduction to Python 3

Module 8

INTRODUCTION TO PYTHON 3

Contents

- What is Python 3?
- Python scripts
- Python help
- Anatomy of a Python script
- Modules
- Functions and built-ins

Summary

- Python built-in functions

What is Python?

Python is an object-oriented scripting language

- First published in 1991 by Guido van Rossum
- Designed as an OOP language from day one
- Does not need knowledge of OO to use

It is powerful

- General purpose, fully functional, rich
- Many extension modules

It is free

- Open source: Python licence is less restrictive than GPL

It is portable

- UNIX, Linux, Windows, OS X, Android, etc...
- Ported to the Java and .NET virtual machines

What is Python 3?

Python 3 was released in December 2008

- Also known as Python 3000 or Py3k

New version of the language

Not backward compatible with Python 2

- 2to3.py tool distributed from Python 2.6 and 3.*n*

Most language features are the same

- Some detail has changed
- Many deprecated features have been tided up and removed

In this course, Python 3 specifics will be indicated by:

Python scripts

python [*-options*]... [**-c** *cmd*|**-m** *mod*|*script file*|-] [*script arguments*]...

- **Python scripts are compiled into byte-code**

→ Like Java, Perl, .NET, etc...

- **Script files are suffixed .py by convention**

→ Compiled modules are suffixed .pyc

→ Make sure your script names do not conflict with standard modules

- **Often called direct on UNIX using `#!/usr/bin/python`**

hello.py

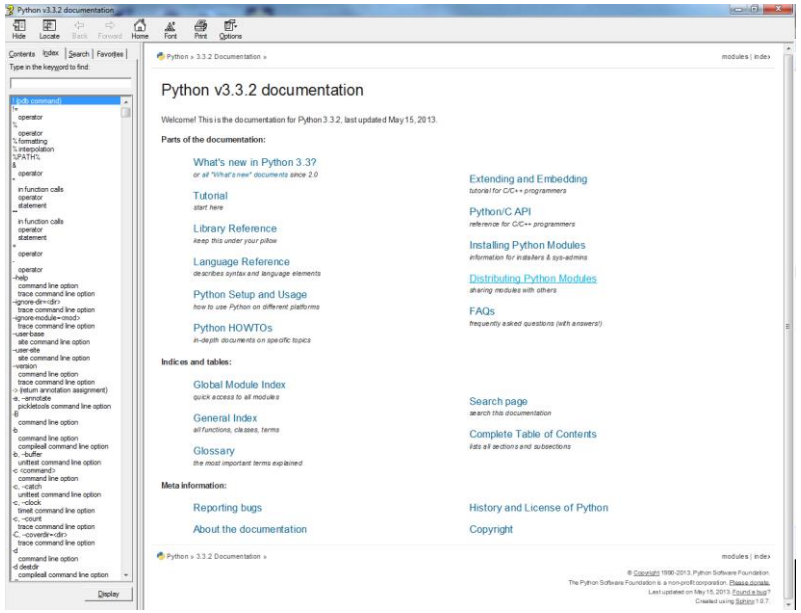
```
#!/usr/bin/python
print('Hello World!')
```

```
chmod u+x hello.py
./hello.py
```

Python help

- **http://docs.python.org**
  → http://docs.python.org/lib is the library reference
- **Or <F1> (Help) from IDLE**
- **UNIX man pages**
  → `man python`
- **Interactive `help()`**

```
>>> help()
```



- **Help module: pydoc**
  → Gives details of standard modules and keywords
  → Implemented as a /usr/bin script on UNIX/Linux
- **PEPs – Python Enhancement Proposals**
  → Explanation of Python changes and features

Anatomy of a Python program

```
#!/usr/bin/python

# Example Python script

import sys

argc = len(sys.argv)

if argc > 1:
    print('Too many args')
else:
    where = 'World'
    print("Hello", where)

print('Goodbye from ' +
      sys.argv[0])
```

Can enclose string literals in either " or '

#! line for UNIX/Linux
ignored on Windows

Comment line

Load an external module

Variable assignment and function call

Condition is terminate by a colon **:**
Limits within a conditional are by consistent indentation

print *inserts a space between parameters*

Statements are terminated by a new-line unless inside brackets
Note the + used to join strings

Modules

- **Most Python programs load other Python code**
  - → Standard Library modules bundled with Python
  - → Downloaded extensions, or modules written locally

```
>>> import sys
>>> print(sys.platform)
```

Find & compile 'sys'
Must specify the module name

```
>>> from sys import *
>>> print(platform)
```

Find & compile 'sys', and import all names to our namespace

- **Examples:**
  - → Operating system specific code - os
  - → Interface to the runtime environment - sys
  - → Scientific libraries like NumPy, DISLIN, SciPy, and many others
  - → Python's built-in functions are in the built-ins module
  - → Automatically imported

Functions and built-ins

- **A function is a named block of program code**
  - → It can be passed values, which might be altered
  - → It can return a value
  - → We can write our own functions

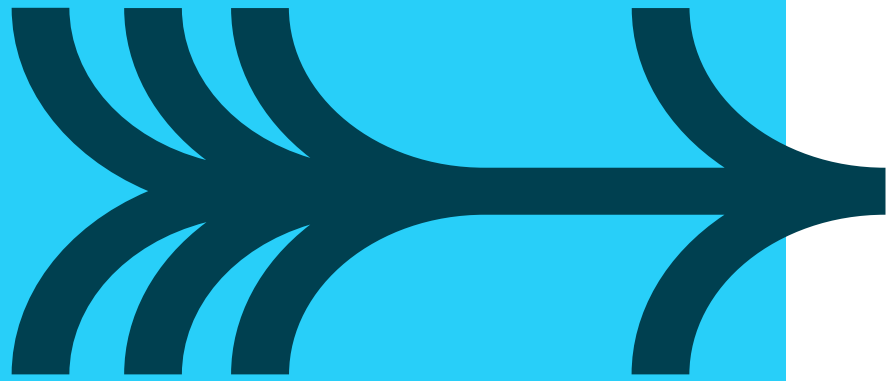
```
lhs = function_name(arg1, arg2,...)
```

- **Python includes many functions built into the product**
  - → Called (remarkably) built-ins
  - → Part of the built-ins module - always available
    - Does not need to be imported
  - → Examples: print, len, str, list, set
  - → See the on-line documentation
    - The Python Standard Library >> Built-in Functions
    - Summary list at the end of this chapter

Prior to Python 3, built-ins was called __built-ins__

INSTRUCTOR DEMONSTRATION

Your instructor will demonstrate a popular Python IDE (Integrated Development Environment) called PyCharm



SUMMARY

- **Python is a free (!) fully functional language**
- Extensive on-line documentation
- **Can be run from scripts, interactively, or from an IDE**
- **Python syntax is different!**
- **Python syntax requires good indentation**
- Intentional!
- **Using external modules is commonplace**
- Load a module by using `import` or `from`
- **Built-in functions are fast, always available, and always used**

Python built-in functions (1)

abs(*x*)
all(*iterable*)
any(*iterable*)
ascii(*object*)
bin(*x*)
bool([*x*])
bytearray([*arg*[, *encoding*[, *errors*]]])
bytes([*arg*[, *encoding*[, *errors*]]])
callable(*object*)
chr(*i*)
classmethod(*function*)
compile(*source*, *filename*,
mode[, *flags*[, *dont_inherit*]])
complex([*real*[, *imag*]])
delattr(*object*, *name*)
dict([*arg*])
dir([*object*])
divmod(*a*, *b*)

enumerate(*iterable*[, *start=0*])
eval(*expression*[, *globals*[, *locals*]])
exec(*object*[, *globals*[, *locals*]])
filter(*function*, *iterable*)
float([*x*])
format(*value*[, *format_spec*])
frozenset([*iterable*])
getattr(*object*, *name*[, *default*])
globals()
hasattr(*object*, *name*)
hash(*object*)
help([*object*])
hex(*x*)
id(*object*)
input([*prompt*])
int([*number* | *string*[, *radix*]])
isinstance(*object*, *classinfo*)
issubclass(*class*, *classinfo*)

Python built-in functions (2)

iter(*o*[, *sentinel*])
len(*s*)
list([*iterable*])
locals()
map(*function*, *iterable*, *...*)
max(*iterable*[, *args...*], *[, *key*])
memoryview(*obj*)
min(*iterable*[, *args...*], *[, *key*])
next(*iterator*[, *default*])
object()
oct(*x*)
open(*file*[, *mode='r'*[, *buffering=None*
[, *encoding=None*[, *errors=None*
[, *newline=None*[, *closefd=True*]]]]]])
ord(*c*)
pow(*x*, *y*[, *z*])
property([*fget*[, *fset*[, *fdel*[, *doc*]]]])

range([*start*], *stop*[, *step*])
repr(*object*)
reversed(*seq*)
round(*x*[, *n*])
set([*iterable*])
setattr(*object*, *name*, *value*)
slice([*start*], *stop*[, *step*])
sorted(*iterable*[, *key*[, *reverse*]])
staticmethod(*function*)
str([*object*[, *encoding*[, *errors*]]])
sum(*iterable*[, *start*])
tuple([*iterable*])
type(*name*, *bases*, *dict*)
vars([*object*])
zip(**iterables*)