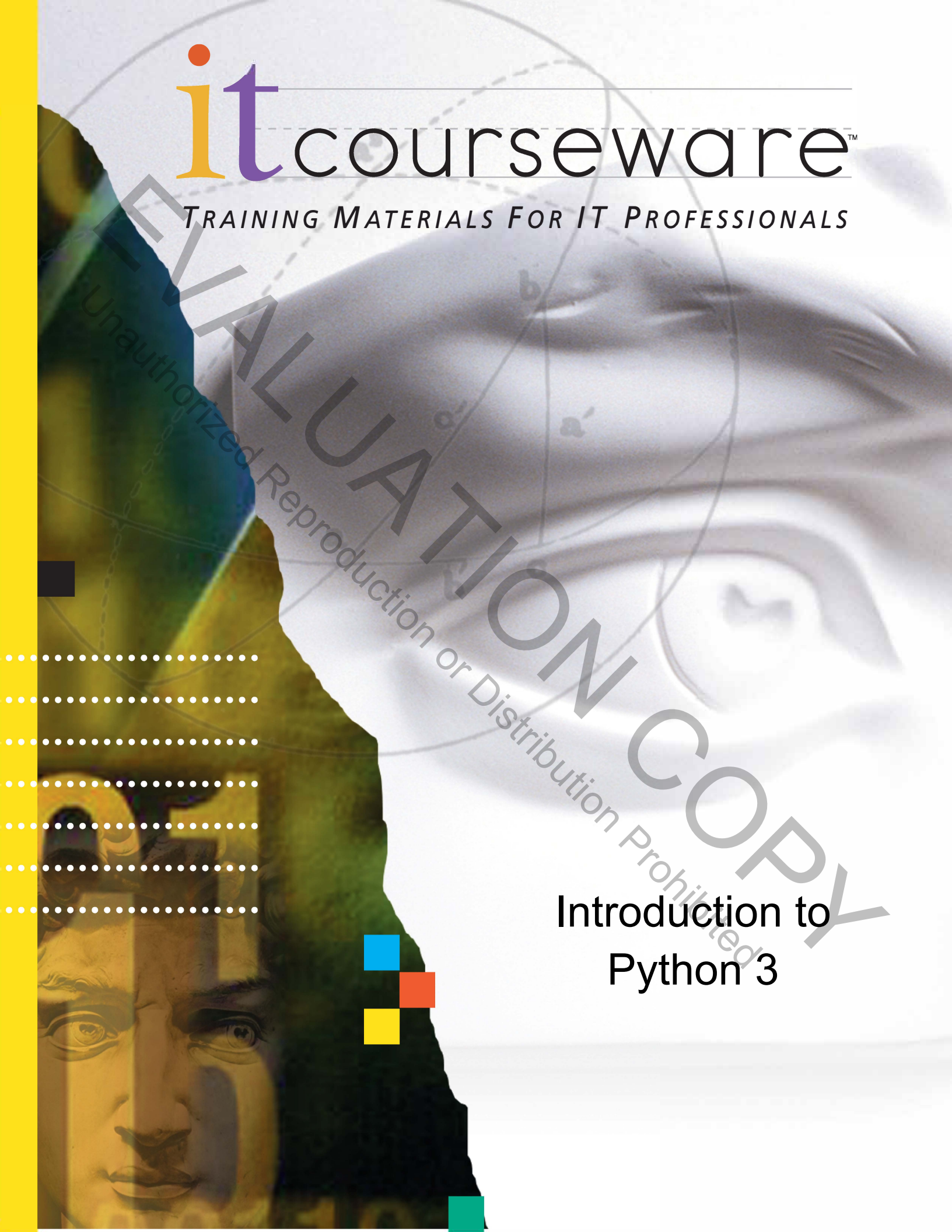
itcourseware™
TRAINING MATERIALS FOR IT PROFESSIONALS
EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited
Introduction to
Python 3

Table of Contents

About this course 1
- Welcome! 2
- Classroom etiquette 3
- Course Outline 4
- Student files 5
- Extracting the student files 6
- Examples 7
- Lab Exercises 8
- Appendices 8

Chapter 1: An Overview of Python 9
- What is Python? 10
- The Birth of Python 12
- About Interpreted Languages 14
- Advantages of Python 15
- Disadvantages of Python 15
- How to get Python 16
- Which version of Python? 16
- The end of Python 2 17
- Getting Help 18
- One day on Dagobah 19

Chapter 2: The Python Environment 21
- Starting Python 22
- If the interpreter is not in your PATH 23
- Using the interpreter 25
- Trying out a few commands 26
- Running Python scripts 27
- Using pydoc 28
- Python Editors and IDEs 30

Chapter 3: Getting Started 33
- Using variables 34
- Keywords and Builtins 35
- Variable typing 37

Strings . . . 38
Single-delimited string literals . . . 39
Triple-delimited string literals . . . 40
Raw string literals . . . 41
Unicode characters . . . 42
String operators and methods . . . 45
String Methods . . . 47
Numeric literals . . . 51
Math operators and expressions . . . 53
Converting among types . . . 56
Writing to the screen . . . 56
String Formatting . . . 59
Legacy String Formatting . . . 61
Command line parameters . . . 64
Reading from the keyboard . . . 65
Chapter 4: Flow Control . . . 69
About flow control . . . 70
What's with the white space? . . . 71
if and elif . . . 72
Conditional Expressions . . . 74
Relational Operators . . . 75
Boolean operators . . . 77
while loops . . . 79
Alternate ways to exit a loop . . . 80
Chapter 5: Array Types . . . 85
About Array Types . . . 86
Lists . . . 88
Indexing and slicing . . . 91
Iterating through a sequence . . . 95
Tuples . . . 97
Iterable Unpacking . . . 99
Nested sequences . . . 100
Functions for all sequences . . . 103
Using enumerate() . . . 106
Operators and keywords for sequences . . . 109

The range() function . . . 112
List comprehensions . . . 115
Generator Expressions . . . 118
Chapter 6: Working with Files . . . 125
Text file I/O . . . 126
Opening a text file . . . 127
The *with* block . . . 128
Reading a text file . . . 129
Writing to a text file . . . 134
Chapter 7: Dictionaries and Sets . . . 139
About dictionaries . . . 140
When to use dictionaries? . . . 141
Creating dictionaries . . . 142
Getting dictionary values . . . 146
Iterating through a dictionary . . . 149
Reading file data into a dictionary . . . 151
Counting with dictionaries . . . 153
About sets . . . 155
Creating Sets . . . 156
Working with sets . . . 157
Chapter 8: Functions . . . 163
Defining a function . . . 164
Returning values . . . 167
Function parameters . . . 168
Variable scope . . . 176
Chapter 9: Sorting . . . 185
Sorting Overview . . . 186
The sorted() function . . . 187
Custom sort keys . . . 188
Lambda functions . . . 193
Sorting nested data . . . 196
Sorting dictionaries . . . 199
Sorting in reverse . . . 201
Sorting lists in place . . . 203

Chapter 10: Errors and Exception Handling . . . **205**
Syntax errors . . . 206
Exceptions . . . 207
Handling exceptions with try . . . 208
Handling multiple exceptions . . . 209
Handling generic exceptions . . . 210
Ignoring exceptions . . . 211
Using else . . . 212
Cleaning up with finally . . . 214
Chapter 11: Using Modules . . . **221**
What is a module? . . . 222
Creating Modules . . . 223
The import statement . . . 226
Where did __pycache__ come from? . . . 227
Module search path . . . 228
Packages . . . 229
Example . . . 230
Module Aliases . . . 231
When the batteries aren't included . . . 232
Chapter 12: Regular Expressions . . . **235**
Regular Expressions . . . 236
RE Syntax Overview . . . 237
Finding matches . . . 239
RE Objects . . . 242
Compilation Flags . . . 245
Groups . . . 249
Special Groups . . . 252
Replacing text . . . 254
Replacing with a callback . . . 256
Splitting a string . . . 259
Chapter 13: Using the Standard Library . . . **261**
The sys module . . . 262
Interpreter Information . . . 262
STDIO . . . 263

Launching external programs . . . 264
Paths, directories and filenames . . . 266
Walking directory trees . . . 270
Grabbing data from the web . . . 273
Sending email . . . 276
math functions . . . 282
Random values . . . 283
Dates and times . . . 286
Zipped archives . . . 289
Chapter 14: Introduction to Python Classes . . . 293
About O-O programming . . . 294
Defining classes . . . 295
Constructors . . . 297
Instance methods . . . 298
Properties . . . 301
Class methods and data . . . 305
Static Methods . . . 307
Private methods . . . 308
Inheritance . . . 309
Untangling the nomenclature . . . 312
Appendix A: Where do I go from here? . . . 315
Resources for learning Python . . . 315
Appendix B: Python Bibliography . . . 317
Appendix C: String Formatting . . . 323
Overview . . . 323
Parameter Selectors . . . 324
Data types . . . 326
Field Widths . . . 329
Alignment . . . 332
Fill characters . . . 335
Signed numbers . . . 337
Parameter Attributes . . . 340
Formatting Dates . . . 342
Run-time formatting . . . 346

Miscellaneous tips and tricks 349
Index **351**

About this course

Welcome!

- We're glad you're here
- Class has hands-on labs for nearly every chapter
- Please make a name tent

Instructor name:

Instructor e-mail:

Have Fun!

Classroom etiquette

- Noisemakers off
- No phone conversations
- Come and go quietly during class.

Please turn off cell phone ringers and other noisemakers.

If you need to have a phone conversation, please leave the classroom.

We're all adults here; feel free to leave the classroom if you need to use the restroom, make a phone call, etc. You don't have to wait for a lab or break, but please try not to disturb others.

IMPORTANT Please do not bring killer rabbits to class. They might maim, dismember, or otherwise disturb your fellow students.

Course Outline

Day 1

Chapter 1 An overview of Python
Chapter 2 The Python environment
Chapter 3 Getting started
Chapter 4 Flow control

Day 2

Chapter 5 Array types
Chapter 6 Working with files
Chapter 7 Dictionaries and sets
Chapter 8 Functions

Day 3

Chapter 9 Sorting
Chapter 10 Errors and exception handling
Chapter 11 Using modules
Chapter 12 Regular expressions

Day 4

Chapter 13 Using the standard library
Chapter 14 An introduction to Python classes

NOTE The actual schedule varies with circumstances. The last day may include *ad hoc* topics requested by students

Student files

You will need to load some files onto your computer. The files are in a compressed archive. When you extract them onto your computer, they will all be extracted into a directory named **py3intro**.

What's in the files?

py3intro contains data and other files needed for the exercises
py3intro/EXAMPLES contains the examples from the course manuals.
py3intro/ANSWERS contains sample answers to the labs.

WARNING The student files do not contain Python itself. It will need to be installed separately. This has probably already been done for you.

Extracting the student files

Windows

Open the file **py3intro.zip**. Extract all files to your desktop. This will create the folder **py3intro**.

Non-Windows (includes Linux, OS X, etc)

Copy or download **py3intro.tgz** to your home directory. In your home directory, type

```
tar xzvf py3intro.tgz
```

This will create the **py3intro** directory under your home directory.

Examples

Nearly all examples from the course manual are provided in the EXAMPLES subdirectory.

It will look like this:

Example

cmd_line_args.py

```
#!/usr/bin/env python

import sys  ①

print(sys.argv) ②

name = sys.argv[1]  ③
print("name is", name)
```

① Import the **sys** module

② Print all parameters, including script itself

③ Get the first actual parameter

cmd_line_args.py Fred

```
['/Users/jstrick/curr/courses/python//examples3/cmd_line_args.py', 'Fred']
name is Fred
```

Lab Exercises

- Relax – the labs are not quizzes
- Feel free to modify labs
- Ask the instructor for help
- Work on your own scripts or data
- Answers are in py3intro/ANSWERS

Appendices

- Appendix A: Where do I go from here?
- Appendix B: Python Bibliography
- Appendix C: String formatting

Chapter 1: An Overview of Python

Objectives

- Learning brief history of Python
- Understanding Python's good points (or bad points)
- Downloading and installing Python
- Comparing Python 2 to Python 3
- Getting help

What is Python?

- All-purpose interpreted language
- Created by Guido van Rossum
- First released (ver. 0.9) February 20, 1991

Python is an open-source, all-purpose programming language.

Python was created by Guido van Rossum beginning in 1989. He was involved with the development of Amoeba, a distributed operating system, and had previously worked on ABC, a scripting language designed to be easier to learn for non-programmers.

Van Rossum took ABC and improved it, adding new features, some of which came from other languages such as Perl and Lisp. His design goal was

> to serve as a second language for people who were C or C++ programmers, but who had work where writing a C program was just not effective.

The first public release was version 0.9 (beta) in 1991.

Guido van Rossum

The Birth of Python

About the origin of Python, Van Rossum wrote in 1996:

> Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office … would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus). (Introduction to Programming Python, by Mark Lutz, published by O'Reilly)
>
> He says, "The first sound bite I had for Python was, 'Bridge the gap between the shell and C.'".
>
> — Guido van Rossum

Monty Python in 1970

Table 1. Python Timeline

Year	Event	Notes
1969	"Monty Python's Flying Circus" premieres on the BBC	
1991	version 0.9.0 (first release)	classes, try/except, lists, dictionaries, modules
1992	version 0.9.6 ported to MS-DOS	
1993	comp.lang.python created	
1994 (Jan)	version 1.0	lambda, reduce(), filter() and map()
1995	version 1.4	keyword arguments, complex numbers
1997 (Dec 31)	version 1.5	
2000 (Oct 16)	version 2.0 final release	list comprehensions
2002 (Dec 21)	version 2.2	unification of types and classes
2004 (Nov 30)	Version 2.4	
2006 (Sept 19)	version 2.5	
2008 (Oct 1)	version 2.6	backward-compatible with 2.5
2008 (Dec 3)	version 3.0	removing old features and adding new features (not backward-compatible with 2.5 or 2.6)
2009 (June 27)	Version 3.1	Backward-compatible with 3.0; new features/modules; deprecated modules
2018 (July 3)	Version 2.7	Final 2.x version
2012 (Sept 29)	Version 3.3	
2014 (Mar 16)	Version 3.4	pip is included, pathlib
2015 (Sept 13)	Version 3.5	subprocess.run
2016 (Dec 23)	Version 3.6	
2018 (June 27)	Version 3.7	
2019 (Oct 14)	Version 3.8	

About Interpreted Languages

- Python is an interpreted language
- The Python interpreter reads a script and interprets it on-the-fly.
- Since there is no compile phase, the development cycle can be very rapid

Like Perl, Ruby, and Bash, Python is an interpreted language. The program consists of a text file containing Python commands. To run the program, you run the interpreter (normally called "python.exe", "python", *etc.*) and tell it which file contains the commands.

Advantages of Python

- Clear, readable syntax
- Multi-paradigm
  - object-oriented programming
  - procedural
  - functional
- Code can be organized into modules and packages
- Exception-based error handling
- Dynamic data structures (e.g., lists and dictionaries)
- Extensive standard library and third party modules
- Strong introspection capabilities
- Can be extended with C/C++

Disadvantages of Python

How to get Python

- Download from **www.python.org**
- Versions available for most operating systems
- **Anaconda** is superset of standard Python

The latest version of Python is always available via the Python home page. http://www.python.org/download will direct you to the latest binaries.

The above URL has Windows MSI installation files.

Linux and OS X come with Python.

For scientific and engineering tasks, the **Anaconda** bundle is a great choice. It contains the Python interpreter plus hundreds of libraries in addition to the standard modules. Among others, it contains **NumPy**, **SciPy**, **Pandas**, **iPython**, and **Matplotlib**. Even if you're not doing scientific programming, it includes **Requests**, **PyQt**, **OpenPyxl**, and many other useful modules.

NOTE Get Anaconda for Windows, Linux, or Mac at https://www.anaconda.com

Which version of Python?

- python -V displays current version

The -V option displays the version of the Python interpreter. Note the *capital* V.

The current release is 3.7

```
$ python -V
Python 3.6.0
```

The end of Python 2

Python 2.7 is intended to be the last minor release in the 2.x series. The Python maintainers are planning to focus their future efforts on Python 3.

This means that 2.7 will continue to run production systems that have not been ported to Python 3. Two consequences of the long-term significance of 2.7 are:

1. It's very likely the 2.7 release will have a longer period of maintenance compared to earlier 2.x versions. Python 2.7 will continue to be maintained while the transition to 3.x continues, and the developers are planning to support Python 2.7 with bug-fix releases beyond the typical two years.
2. A policy decision was made to silence warnings only of interest to developers. DeprecationWarning and its descendants are now ignored unless otherwise requested, preventing users from seeing warnings triggered by an application. This change was also made in the branch that will become Python 3.2. (Discussed on stdlib-sig and carried out in issue 7319.)

```
-- from the Python 2.7 documentation_
```

At PyCon 2014 in Montreal, Guido van Rossum extended the end-of-life date for 2.7 to 2020. More recently, the end-of-life date for Python 2.7 has been established on January 1, 2020. This means that there will be no more support from the core Python developers, including bug fixes.

Getting Help

- Books
- Web sites
- pydoc

There are many ways of getting help with Python. Appendix A lists some of the best Python books.

A good starting place is http://docs.python.org/3/index.html.

A good source of Python books is Packt Publishing: https://www.packtpub.com/

One day on Dagobah

```
EXTERIOR: DAGOBAH -- DAY

With Yoda strapped to his back, Luke climbs up one of the many thick vines
that grow in the swamp until he reaches the Dagobah statistics lab. Panting
heavily, he continues his exercises -- grepping, installing new packages,
logging in as root, and writing replacements for two-year-old shell scripts
in Python.
YODA: Code!  Yes.  A programmer's strength flows from code maintainability. But
beware of Perl. Terse syntax... more than one way to do it... default variables.
The dark side of code maintainability are they. Easily they flow, quick to join
you when code you write.  If once you start down the dark path, forever will it
dominate your destiny, consume you it will.
LUKE: Is Perl better than Python?
YODA: No... no... no.  Quicker, easier, more seductive.
LUKE: But how will I know why Python is better than Perl?
YODA: You will know.  When your code you try to read six months from now.
```

Chapter 2: The Python Environment

Objectives

- Using the interpreter
- Getting help
- Running scripts on Windows, Linux, and Mac
- Learning best editors and IDEs

> I think the real key to Python's platform independence is that it was conceived right from the start as only very loosely tied to Unix.
>
> — Guido van Rossum

Starting Python

- Type **python** (or **python3**) at a command prompt
- **python** should be in your PATH
- If python was not found, install it or add directory to PATH

To start the Python interpreter, just type **python** (or **python3**) at the command prompt. If you get an error message, one of two things has happened:

- Python is not installed on your computer
- The interpreter (`python`) is not in your PATH variable

If the interpreter is not in your PATH

If the directory where the interpreter lives is not in your PATH variable, you have several choices.

Type the full path

Start **python** by typing the full path to the interpreter (e.g., C:\python35\python or /usr/bin/python)

Add the directory to PATH temporarily

Windows (at a command prompt)

```
set PATH="%PATH%";c:\python35
```

Linux/Mac

```
PATH="$PATH:/usr/dev/bin"   sh,ksh,bash
setenv PATH "$PATH:/usr/dev/bin"    csh,tcsh
```

Add the directory to PATH permanently

Windows

Right-click on the **My Computer** icon. Select **Properties**, and then select the **Advanced** tab. Click on the **Environment Variables** button, then double-click on **PATH** in the **System Variables** area at the bottom of the dialog. Add the directory for the Python interpreter to the existing value and click OK. Be sure to separate the path from existing text with a semicolon.

Linux/Mac

Add a line to your shell startup file (e.g. .bash_profile, .profile, etc.) to add the directory containing the Python interpreter to your PATH variable .

The command should look something like

```
PATH="$PATH:/path/to/python"
```

Using the interpreter

- Type any Python statement or expression
- Prompt is >>>
- Command line editing supported
- Ctrl-D (Unix) or Ctrl-Z <Enter> (Windows) to exit

Once you have started the Python interpreter, it provides an interactive interpreter. The prompt is ">>>". You can type in any Python commands at this prompt.

For Windows, it supports the editing keys on a standard keyboard, which include Home, End, etc., as well as the arrow keys. Normal PC shortcuts such as Ctrl-RightArrow to jump to the next word also work.

For other systems, Python supports **GNU readline** editing, which uses emacs-style commands. These commands are detailed in the table below.

On all versions, you can use arrow keys and backspace to edit the line.

As of version 3.4, the interpreter does autocomplete when you press the TAB key

Table 2. emacs-style command line editing

Emacs-mode Command	**Function**
^P	Previous command
^N	Next command
^F	Forward 1 character
^B	Back 1 character
^A	Beginning of line
^E	End of line
^D	Delete character under cursor
^K	Delete to end of line

Trying out a few commands

Try out the following commands in the interpreter:

```
>>> print("Hello, world")
Hello, world
>>> print(4 + 3)
7
>>> print(10/3)
3.3333333333333335
>>>
```

You don't really need **print()**

```
>>> "Hello, world"
'Hello, world'
>>> 4 + 3
7
>>>
```

When you press <Enter>, the interpreter evaluates and prints out whatever you typed in.

NOTE If you have **ipython** installed, use **ipython** for a better interactive interpreter (**ipython** is included with Anaconda).

Running Python scripts

- Use Python interpreter
- Same for any OS

To run a Python script (a file with the extension **.py**, call the Python interpreter with the script as its argument:

```
python myscript.py
```

This will work on any operating system.

NOTE If you are sure that Python is installed, and the above technique does not work, it might be because the Python interpreter is not in your path. See the earlier discussion about adding the python executable to your path.

Using pydoc

From the Python interpreter

Type

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

Where *thing* can be either the name (in quotes) of a function, module or package, or the actual (imported) function, module, or package object.

```
>>> help(len)
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

From a command line

Use `pydoc name` to display the documentation for *name*, which can be the name of a function, module, package, or a method or attribute of an object.

```
$ pydoc len
Help on built-in function len in module __builtin__:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or mapping.
```

NOTE On Windows, open an Anaconda prompt (if available) to make sure that **pydoc** is in the search path.

TIP Run pydoc -k <keyword> to search packages by keyword

From iPython

iPython makes it easy to get help. Just put a question mark before or after an object, and it will display help.

```
In [1]: len?
Signature: len(obj, /)
Docstring: Return the number of items in a container.
Type:      builtin_function_or_method
```

Python Editors and IDEs

- Editor is programmer's most-used tool
- Select Python-aware editor or IDE
- Many open source and commercial choices

There are two pages on the Python Wiki that discuss editors and IDEs:

```
http://wiki.python.org/moin/PythonEditors
http://wiki.python.org/moin/IntegratedDevelopmentEnvironments
```

PyCharm Community Edition is the most full-featured free IDE available. Other good multi-platform IDEs include Spyder, Eclipse, Visual Studio Code, and Sublime Edit. These work on Windows, Unix/Linux, and Mac platforms and probably some others.

Chapter 2 Exercises

Exercise 2-1 (hello.py)

Using any editor, write a "Hello, world" python script.

Run the script from the command line.

Open the script in your IDE and run it from there.

TIP In PyCharm, you can right-click (Ctrl-click on Mac) the script's tab and select **Run**

7400 E. Orchard Road, Suite 1450 N
Greenwood Village, Colorado 80111
Ph: 303-302-5280
www.ITCourseware.com