



nclab

Learning for Industry 4.0

Introduction to

Python

Programming



Introduction to Python Programming

Introduction to Python Programming

Dr. Pavel Solin

Revision October 11, 2018

About the Author

Dr. Pavel Solin is Professor of Applied and Computational Mathematics at the University of Nevada, Reno. He is an expert in scientific computing, leader of open source projects, organizer of international scientific congresses, and the author of several textbooks, research monographs, and many research articles in international journals.

Acknowledgment

Our sincere thanks go to many educators and students for providing valuable feedback on this textbook as well as on the self-paced Python I and Python II courses in NCLab.

Copyright:

Copyright (c) 2018 NCLab Inc. All Rights Reserved.

Preface

Python is a modern high-level dynamic programming language which is widely used in business, science, and engineering applications. According to Github, as of 2018 Python is the second most popular programming language after Javascript. Python's growing popularity has the following reasons:

- It is known as a beginner's language because of its simplicity.
- It helps developers to be more productive from development to deployment and maintenance.

Python's syntax is very simple and high level when compared to Java, C and C++. Therefore, applications can be built with less code. Last, Python has a large collection of powerful libraries.

Python libraries

Thanks to its free libraries, Python is replacing traditional software products in many areas. In data analysis and statistics, Python has become more popular than SAS and even than R. In scientific computing and visualization, the collection of the libraries Scipy, Numpy, Sympy, Matplotlib and Mayavi has become more popular than MATLAB. Python also has powerful libraries for machine learning (Tensorflow, Keras), for graph theory (NetworkX), and many others.

How to read this textbook

In contrast to other languages, Python can be used by non-programmers. Therefore, as the first thing we will show you how to use Python as a powerful command-line scientific calculator and data visualization tool.

Starting with Section 4 we will explore Python as a programming language. We will show you how to work with text strings, variables, tuples, lists, dictionaries, functions, loops, conditions, files, exceptions, recursion, object-oriented programming, recursion, decorators, and more.

In every section, we will begin with simple examples and gradually progress to advanced applications. Therefore, you don't need to read every section until the end. It's perfectly fine to read the first part of a section, move on, and return to the advanced topics later.

Enjoy the textbook, and contact me at pavel@nclab.com if you have any questions, find typos, or have suggestions for other topics you would like to see here. Many thanks! – Pavel

Table of Contents

1	Introduction	1
1.1	Objectives	1
1.2	Why learn Python?	1
1.3	Compiled and interpreted programming languages	2
1.4	A few more details about Python	3
1.5	Create a user account in NCLab	3
1.6	Working in the cloud setting	4
1.7	Internet connection requirements	4
1.8	Recommended devices	4
2	Using Python as a Scientific Calculator	5
2.1	Objectives	5
2.2	Why use Python as a calculator?	5
2.3	Addition and subtraction	5
2.4	Multiplication	6
2.5	Division	6
2.6	Floor division	6
2.7	Modulo	7
2.8	Powers	7
2.9	Priority of operations	8
2.10	Using empty spaces makes your code more readable	9
2.11	Importing the Numpy library	9
2.12	Finite computer arithmetic	10
2.13	Rounding numbers	10
2.14	List of math functions and constants	11
2.15	Using the Fractions library	12
2.16	Working with random numbers	12
2.17	Complex numbers	14
3	Drawing, Plotting, and Data Visualization with Matplotlib	15
3.1	Objectives	15
3.2	Why learn about data visualization with Python?	15
3.3	RGB colors	15
3.4	Secondary colors	16
3.5	Figuring out RGB codes	16
3.6	RGB color palettes	17
3.7	Starting with Matplotlib	17

3.8	Two ways to specify color	18
3.9	Making axes equally scaled	20
3.10	Filling shapes with color	21
3.11	Borders	22
3.12	Interrupting lines	23
3.13	Making holes	23
3.14	Plotting functions of one variable	24
3.15	Line style, label, and legend	26
3.16	Showing the grid	28
3.17	Adjusting plot limits	29
3.18	Plotting multiple functions at once	30
3.19	Plotting circles	32
3.20	Making a circular hole	33
3.21	Plotting ellipses	34
3.22	Plotting spirals	35
3.23	Parametric curves in the 3D space	36
3.24	Wireframe plots of functions of two variables	37
3.25	Surface plots	38
3.26	Color maps	39
3.27	Contour plots	42
3.28	Changing view and adding labels	43
3.29	Patches and artists	44
3.30	Pie charts	46
3.31	Donut charts	48
3.32	Bar charts	49
3.33	Statistical data visualization with the Seaborn library	52
3.34	Interactive scientific data visualization with Mayavi2	53
4	Working with Text Strings	54
4.1	Objectives	54
4.2	Why learn about text strings?	54
4.3	Defining text strings	54
4.4	Storing text strings in variables	55
4.5	Measuring the length of text strings	55
4.6	Python is case-sensitive	56
4.7	The <code>print</code> function	56
4.8	Function <code>help</code>	57
4.9	Changing the default behavior of the <code>print</code> function	57
4.10	Undesired trailing spaces and function <code>repr</code>	59
4.11	Cleaning text strings with <code>strip</code> , <code>lstrip</code> , and <code>rstrip</code>	59

4.12	Wait - what is a "method" exactly?	60
4.13	Calling text string methods on raw text strings	61
4.14	Using single and double quotes in text strings	61
4.15	A more robust approach - using characters <code>'</code> and <code>"</code>	62
4.16	Length of text strings containing special characters	62
4.17	Newline character <code>\n</code>	63
4.18	Writing long code lines over multiple lines with the backslash <code>\</code>	63
4.19	Multiline strings enclosed in triple quotes	64
4.20	Adding text strings	65
4.21	Multiplying text strings with integers	66
4.22	Parsing text strings with the <code>for</code> loop	66
4.23	Reversing text strings with the <code>for</code> loop	67
4.24	Accessing individual characters via their indices	68
4.25	Using negative indices	68
4.26	ASCII table	69
4.27	Finding the ASCII code of a given character	69
4.28	Finding the character for a given ASCII code	70
4.29	Slicing text strings	70
4.30	Third index in the slice	71
4.31	Creating copies of text strings	71
4.32	Reversing text strings using slicing	72
4.33	Retrieving current date and time	72
4.34	Making text strings lowercase	73
4.35	Checking for substrings in a text string	73
4.36	Making the search case-insensitive	74
4.37	Making text strings uppercase	75
4.38	Finding and replacing substrings	75
4.39	Counting occurrences of substrings	76
4.40	Locating substrings in text strings	76
4.41	Splitting a text string into a list of words	77
4.42	Splitting a text string while removing punctuation	77
4.43	Joining a list of words into a text string	78
4.44	Method <code>isalnum</code>	78
4.45	Method <code>isalpha</code>	79
4.46	Method <code>isdigit</code>	79
4.47	Method <code>capitalize</code>	80
4.48	Method <code>title</code>	80
4.49	C-style string formatting	80
4.50	Additional string methods	81

4.51	The <code>string</code> library	81
4.52	Natural language toolkit (NLTK)	81
5	Variables and Types	83
5.1	Objectives	83
5.2	Why learn about variables?	83
5.3	Statically vs. dynamically typed languages	84
5.4	Types of variables (data types) in Python	84
5.5	Using a non-initialized variable	84
5.6	Various ways to create variables and initialize them with values	85
5.7	Casting variables to text strings	86
5.8	Casting can be tricky sometimes	87
5.9	Inadmissible casting	88
5.10	Dynamic type interpretation and its abuse	88
5.11	Determining the type of a variable at runtime	89
5.12	Operators <code>+=</code> , <code>-=</code> , <code>*=</code> and <code>\=</code>	91
5.13	Local and global variables	92
5.14	Using global variables in functions can get you in trouble	92
5.15	Changing global variables in functions can get you in big trouble	93
5.16	Shadowing of variables	94
5.17	Callable variables	94
6	Boolean Values, Functions, Expressions, and Variables	95
6.1	Objectives	95
6.2	Why learn about Booleans?	95
6.3	Boolean values	95
6.4	Boolean functions	96
6.5	Comparison operators for numbers	96
6.6	Comparison operator <code>==</code> vs. assignment operator <code>=</code>	97
6.7	Comparison operators for text strings	98
6.8	Storing results of Boolean expressions in variables	98
6.9	Example: Checking if there exists a triangle with edge lengths a, b, c	99
6.10	Logical <code>and</code> and its truth table	100
6.11	Logical <code>or</code> and its truth table	101
6.12	Negation <code>not</code>	102
6.13	Booleans in conditions and the <code>while</code> loop	102
6.14	Example: Monte Carlo calculation of π	104
7	Lists, Tuples, Dictionaries, and Sets	106
7.1	Objectives	106
7.2	Why learn about lists?	106
7.3	Creating a list	106

7.4	Important properties of lists	107
7.5	Measuring the length of a list	107
7.6	Appending new items to lists	107
7.7	Adding lists	108
7.8	Adding is not appending	109
7.9	Multiplying lists with integers	109
7.10	Parsing lists with the <code>for</code> loop	110
7.11	Accessing list items via their indices	110
7.12	Slicing lists	111
7.13	Creating a copy of a list – the wrong way and the right way	111
7.14	Popping list items by index	112
7.15	Deleting list items by index	113
7.16	Checking if an item is in a list	113
7.17	Locating an item in a list	114
7.18	Finding and deleting an item from a list	114
7.19	Finding and deleting all occurrences of an item	115
7.20	Counting occurrences of items	115
7.21	Inserting list items at arbitrary positions	115
7.22	Sorting a list in place	116
7.23	Creating sorted copy of a list	116
7.24	Using key functions in sorting	117
7.25	Using lambda functions in sorting	117
7.26	Reversing a list in place	118
7.27	Creating reversed copy of a list	118
7.28	Zippping lists	118
7.29	List comprehension	119
7.30	List comprehension with <code>if</code> statement	119
7.31	List comprehension with <code>if</code> and <code>else</code>	120
7.32	List comprehension with nested loops	121
7.33	Mutable and immutable objects	122
7.34	Mutability of lists	124
7.35	Tuples	126
7.36	Read-only property of tuples	127
7.37	Immutability of tuples	127
7.38	Dictionaries	128
7.39	Creating an empty and nonempty dictionary	129
7.40	Keys, values, and items	129
7.41	Accessing values using keys	130
7.42	Adding, updating, and deleting items	130

7.43	Checking for keys, values, and items	131
7.44	Finding all keys for a given value	131
7.45	Finding all keys for a given value using comprehension	132
7.46	Reversing a dictionary using comprehension	133
7.47	Beware of repeated values	133
7.48	Creating a dictionary from two lists	134
7.49	Reversing a dictionary using <code>zip</code>	134
7.50	Creating a copy of a dictionary	134
7.51	Merging dictionaries	135
7.52	Adding dictionaries	136
7.53	Composing dictionaries	136
7.54	Sets	137
7.55	Creating sets	138
7.56	Adding and removing elements	138
7.57	Popping random elements	139
7.58	Checking if an element is in a set	140
7.59	Checking for subsets and supersets	140
7.60	Intersection and union	141
7.61	Difference and symmetric difference	142
7.62	Using a set to remove duplicated items from a list	143
8	Functions	144
8.1	Objectives	144
8.2	Why learn about functions?	144
8.3	Defining new functions	144
8.4	Docstrings and function <code>help</code>	145
8.5	Function <i>parameters</i> vs. <i>arguments</i>	146
8.6	Names of functions should reveal their purpose	146
8.7	Functions returning multiple values	147
8.8	What is a <i>wrapper</i> ?	149
8.9	Using parameters with default values	149
8.10	Functions accepting a variable number of arguments	151
8.11	<code>args</code> is not a keyword	152
8.12	Passing a list as <code>*args</code>	153
8.13	Obtaining the number of <code>*args</code>	154
8.14	Combining standard arguments and <code>*args</code>	154
8.15	Using keyword arguments <code>*kwargs</code>	155
8.16	Passing a dictionary as <code>**kwargs</code>	156
8.17	Defining local functions within functions	157
8.18	Anonymous lambda functions	158

8.19	Creating multiline lambdas	159
8.20	Using lambdas to create a live list of functions	159
8.21	Using lambdas to create a live table of functions	160
8.22	Using lambdas to create a live table of logic gates	160
8.23	Using lambdas to create a factory of functions	161
8.24	Variables of type 'function'	162
8.25	Inserting conditions into lambda functions	163
8.26	Using lambdas to customize sorting	163
8.27	Obtaining useful information about functions from their attributes	164
9	The 'For' Loop	165
9.1	Objectives	165
9.2	Why learn about the <code>for</code> loop?	165
9.3	Parsing text strings	166
9.4	Splitting a text string into a list of words	166
9.5	Parsing lists and tuples (including comprehension)	166
9.6	Parsing two lists simultaneously	166
9.7	Iterating over sequences of numbers and the built-in function <code>range</code>	167
9.8	Parsing dictionaries (including comprehension)	168
9.9	Nested <code>for</code> loops	169
9.10	Terminating loops with the <code>break</code> statement	170
9.11	Terminating current loop cycle with <code>continue</code>	171
9.12	The <code>for-else</code> statement	173
9.13	The <code>for</code> loop behind the scenes – iterables and iterators	175
9.14	Making your own classes iterable	178
9.15	Generators	178
9.16	Functional programming	179
10	Conditions	180
10.1	Objectives	180
10.2	Why learn about conditions?	180
10.3	Boolean values, operators, and expressions	180
10.4	Using conditions without the keyword <code>if</code>	180
10.5	The <code>if</code> statement	181
10.6	Types of values accepted by the <code>if</code> statement	182
10.7	The optional <code>else</code> branch	183
10.8	The full <code>if-elif-else</code> statement	183
10.9	Example – five boxes of apples	184
10.10	Conditional (ternary) expressions	185
10.11	Nested conditional expressions	187
10.12	Famous workarounds	187

11	The 'While' Loop	189
11.1	Objectives	189
11.2	Why learn about the <code>while</code> loop?	189
11.3	Syntax and examples	189
11.4	How to time your programs	190
11.5	The <code>break</code> statement	192
11.6	The <code>continue</code> statement	192
11.7	Emulating the <code>do-while</code> loop	193
11.8	The <code>while-else</code> statement	194
11.9	Abusing the <code>while</code> loop	195
11.10	Solving an unsolvable equation	196
11.11	Numerical methods and scientific computing	198
12	Exceptions	199
12.1	Objectives	199
12.2	Why learn about exceptions?	199
12.3	Catching exceptions – statement <code>try-except</code>	201
12.4	List of common exceptions	202
12.5	Catching a general exception	204
12.6	Function <code>assert</code>	204
12.7	Throwing exceptions – statement <code>raise</code>	206
12.8	The full statement <code>try-except-else-finally</code>	206
13	File Operations	207
13.1	Objectives	207
13.2	Why learn about files?	207
13.3	NCLab file system and security of your data	207
13.4	Creating sample text file	208
13.5	Creating sample Python file	209
13.6	Opening a text file for reading – method <code>open</code>	210
13.7	Closing a file – method <code>close</code>	210
13.8	Checking whether a file is closed	210
13.9	Using the <code>with</code> statement to open files	211
13.10	Getting the file name	211
13.11	A word about encoding, Latin-1, and UTF-8	211
13.12	Checking text encoding	212
13.13	Other modes to open a file	212
13.14	Reading the file line by line using the <code>for</code> loop	213
13.15	The mystery of empty lines	214
13.16	Reading individual lines – method <code>readline</code>	215
13.17	Reading the file line by line using the <code>while</code> loop	217

13.18	Reading the file using <code>next</code>	218
13.19	Reading the file as a list of lines – method <code>readlines</code>	218
13.20	File size vs available memory considerations	219
13.21	Reading the file as a single text string – method <code>read</code>	220
13.22	Rewinding a file	220
13.23	File pointer, and methods <code>read</code> , <code>tell</code> and <code>seek</code>	222
13.24	Using <code>tell</code> while iterating through a file	225
13.25	Another sample task for <code>tell</code>	227
13.26	Writing text to a file – method <code>write</code>	227
13.27	Writing a list of lines to a file – method <code>writelines</code>	230
13.28	Writing to the middle of a file	231
13.29	Things can go wrong: using exceptions	233
13.30	Binary files I – checking byte order	234
13.31	Binary files II – writing unsigned integers	234
13.32	Binary files III – writing bit sequences	235
13.33	Binary files IV – reading byte data	237
14	Object-Oriented Programming I - Introduction	238
14.1	Objectives	238
14.2	Why learn about OOP?	238
14.3	Procedural (imperative) programming and OOP	238
14.4	Main benefits of OOP in a nutshell	239
14.5	There are disadvantages, too	239
14.6	Procedural vs. object-oriented thinking	239
14.7	Classes, objects, attributes and methods	240
14.8	Defining class <code>Circle</code>	241
14.9	Using class <code>Circle</code>	244
15	Object-Oriented Programming II - Class Inheritance	247
15.1	Objectives	247
15.2	Why learn about class inheritance?	247
15.3	Hierarchy of vehicles	247
15.4	Class inheritance	249
15.5	Base class <code>Geometry</code>	249
15.6	Hierarchy of geometrical shapes	250
15.7	Deriving class <code>Polygon</code> from class <code>Geometry</code>	251
15.8	Deriving classes <code>Triangle</code> and <code>Quad</code> from class <code>Polygon</code>	253
15.9	Deriving class <code>Rectangle</code> from class <code>Quad</code>	254
15.10	Deriving class <code>Square</code> from class <code>Rectangle</code>	254
15.11	Deriving class <code>Circle</code> from class <code>Geometry</code>	255
15.12	Sample application	256

16	Object-Oriented Programming III - Advanced Aspects	259
16.1	Objectives	259
16.2	Why learn about polymorphism and multiple inheritance?	259
16.3	Inspecting objects with <code>isinstance</code>	259
16.4	Inspecting instances of custom classes with <code>isinstance</code>	260
16.5	Inspecting objects with <code>type</code>	261
16.6	Inspecting classes with <code>help</code>	261
16.7	Obtaining class and class name from an instance	265
16.8	Inspecting classes with <code>issubclass</code>	265
16.9	Inspecting objects with <code>hasattr</code>	266
16.10	Polymorphism I - Meet the Lemmings	267
16.11	Polymorphism II - Geometry classes	269
16.12	Multiple inheritance I - Introduction	270
16.13	Multiple inheritance II - The Diamond of Dread	271
16.14	Multiple inheritance III - Leonardo da Vinci	271
17	Recursion	274
17.1	Objectives	274
17.2	Why learn about recursion?	274
17.3	Introduction	274
17.4	Is your task suitable for recursion?	275
17.5	Calculating the factorial	276
17.6	Stopping condition and infinite loops	279
17.7	Parsing binary trees	280
18	Decorators.....	282
18.1	Introduction	282
18.2	Building a timing decorator	284
18.3	Building a debugging decorator	287
18.4	Combining decorators	288
18.5	Decorating class methods	289
18.6	Passing arguments to decorators	291
18.7	Debugging decorated functions	293
18.8	Python Decorator Library.....	294
19	Selected Advanced Topics	295
19.1	Objectives	295
19.2	Maps.....	295
19.3	Filters	296
19.4	Function <code>reduce()</code>	297
19.5	Shallow and deep copying	299

Full text is available upon enrollment.