

Andrew Shitov

A Tour of Python 3

*300+ ready-to-use examples
guiding you from the very beginning*

Including Python 3.8

DeepText — 2020

A Tour of Python 3

300+ ready-to-use examples guiding you from the very beginning.

Including Python 3.8

© Andrew Shitov, author, 2020

This book is a collection of small programs that demonstrate different features of the Python programming language. The reader can use this book as a textbook and a collection of exercises for learning Python 3.

The 1st edition published on 13th July 2020

Published by DeepText, Amsterdam

<https://andrewshitov.com/a-tour-of-python3>

Preface

Dear reader,

This book is an overview of the features of the Python programming language. In the 36 chapters, there are more than 300 small programs, where you will learn many details about all main aspects of programming in this language.

The book is primarily intended to be used with the modern compiler of the third version of the language and highlights the differences between Python 2 and Python 3 where that makes sense. A number of sections explain the new features appeared in Python 3.8.

In general, the difficulty of the examples increases to the end of the book, but you can read the chapters and the sections in any order.

To run the examples, you need a working compiler, which you can download and install from the official site python.org.

The author wishes you a pleasant journey to the world of Python.

Andrew Shitov

July 12, 2020

Amsterdam

Contents

Part 1

Chapter 1. Printing and Formatting

Hello, World!	24
Formatted output	24
Formatting strings	25
F-strings	26
More on formatting.....	26
Formatted strings with named arguments	27
Echo.....	28
Another echo.....	28
Using <code>pprint</code>	29
Printing in Python 2 vs Python 3	30
Format precision.....	31
Emoji	32

Chapter 2. Python Code

Indentation.....	34
Namespaces.....	34

Chapter 3. Variables

Variables	38
Variable names	38

Deleting a variable	39
Dummy variable	40
The <code>global</code> keyword.....	40
How to swap the values	41
Star assignment	42

Chapter 4. Numbers

Integer vs floating-point division.....	44
Using complex numbers.....	44
Decimal calculations	45
Complex roots.....	46
Big integers.....	47
Size of an integer	48
Fractions	49
Equal floats.....	50
Round a number	51

Chapter 5. Strings

The length of a string	54
Multi-line strings	54
Indexing a string character	55
Taking the last character	56
The second-to-last character	57
Getting a substring	57
String repetition	58
Reversing a string.....	58
Splitting a string	59

Raw strings	60
More on raw strings	60
Removing accents from a string	61
Removing <code>\n</code>	62
A more complex stripping.....	62
Using lambdas to strip a line.....	63
ASCII values of the characters	64
Character Unicode names	65
Using casefolding.....	66
Unicode normalization.....	66
Split words.....	67

Chapter 6. Operators

Non-trivial assignments	70
Augmenting assignment	70
The <code>+</code> operator.....	71
The <code>in</code> keyword	72
Redefine <code>+</code>	73
More on redefining <code>+</code>	74
Walrus <code>:=</code>	75
Using <code>*</code>	75

Chapter 7. Functions

Creating a function	78
Default parameter values	78
Named arguments.....	79
More on default values.....	80

Passing a list to a function	80
Multiple return from a function	81
Return a tuple.....	82
Arbitrary number of function arguments.....	83
Using kwargs	84
Position of the variadic arguments	85
Positional parameters only.....	86
Default parameters are frozen.....	86
Simple documentation.....	87
Annotations.....	88
LRU cache.....	89
Decorators.....	90
Double decorator.....	91

Chapter 8. Built-in Functions

The built-in function <code>len</code>	94
<code>reverse</code> and <code>reversed</code>	94
<code>any</code> and <code>all</code>	95
Using <code>enumerate</code>	96
The <code>map</code> function.....	96
The <code>id</code> function.....	97
Using <code>filter</code>	98
Custom sort function.....	100
More on sort key	101

Part 2

Chapter 9. Types

Data types	106
Type casting	106
Tuples vs lists	107
None	108
Ordering None.....	108
is vs ==	109
A tuple with a single element	110
Create a tuple.....	111

Chapter 10. Lists

Pushing to an array and popping back	114
Negative index.....	115
Last elements	116
Appending data to a list	117
Remove an item from a list	118
Another way of removing an element	118
Copying a list.....	119
Find a position in a list	120
Reverse a list	121
Rotating a list.....	122
Shuffle a list	123
Join list items	123
Clone a list	124
More on cloning lists.....	125
Append vs. extend.....	126

Beware of deleting items.....	127
Chapter 11. Dictionaries	
Iterating a dictionary.....	130
The <code>items</code> method of dictionaries.....	130
Keys and values of a dictionary	131
Using <code>get</code> with dictionaries.....	132
Iterating over a dictionary	133
Sort by value	134
Merge dictionaries	135
More on merging dictionaries	136
Lists to dictionaries.....	137
Invert a dictionary.....	137
Chapter 12. Sets	
Sets.....	140
Subtracting sets	140
Working with sets.....	141
Frozen sets	142
Set operations.....	142
Chapter 13. Ranges	
Ranges.....	146
An open range	146
More on ranges	147
Counting backwards	148
Range properties.....	148

Sum of a range.....	149
Ranges and strings.....	149

Chapter 14. Regular Expressions

Getting a number using regular expressions	152
Date and regular expressions	153
Case-insensitive matching	153
Unicode digits	154
Compiled regular expressions.....	155
Extract the numbers and the words.....	156

Chapter 15. Date and Time

Daylight saving time	160
Leap year.....	160
Date difference	161
Weekday today.....	162
Measuring time.....	162
Another way of measuring time.....	163

Part 3

Chapter 16. Control Flow

Boolean if	168
The <code>elif</code> keyword.....	168
Chained condition	169
Empty values	170
A string with a zero.....	171
<code>for ... in</code>	171

Using <code>for ... in</code> with strings.....	172
The <code>while</code> loop	173
Using <code>else</code> in a <code>while</code> loop	173
More on <code>else</code> in a loop.....	174
Breaking a <code>while</code> loop.....	175
Continuing a <code>while</code> loop.....	175
Using <code>pass</code> in loops	176
Variable state after the loop	177
Using <code>finally</code>	178
<code>for</code> and <code>else</code>	179
Ternary operator.....	180
More on conditional statements	181
Breaking nested loops.....	181
Boolean shortcuts	182
<code>if</code> vs. <code>in</code>	183
Functions in <code>for</code>	184

Chapter 17. Command Line

Getting user input	188
Command-line arguments	188
<code>argv[0]</code>	189
Integer as input.....	189
Read a password	190
Clear screen	191

Chapter 18. Modules

Installing a module.....	194
--------------------------	-----

<code>import</code> vs <code>local</code>	194
Testing the main code	195
Module alias	196
Path to the module	197

Chapter 19. Files

Reading a file	200
Write to a file	200
File size	201
Does the file exist?	202
Using <code>with</code> to open files	202
Open two files in a single <code>with</code>	203
Using <code>writelines</code>	204
Temporary file	204
Compare files	205
Reading CSV	206
Remove empty lines	207
Get the line N from a file	208
Another way to read the N^{th} line	209
Last line of a file	210

Chapter 20. Working with the System

Using <code>sys</code>	214
Printing to <code>STDERR</code>	214
React to <code>Ctrl+C</code>	215
System calls	216
Capture system output	216

Capture lines	217
C interface	218
Fortran interface.....	219

Chapter 21. Exceptions

Using <code>assert</code>	222
Custom Exception.....	223
Exception with a message.....	224

Part 4

Chapter 22. List and Map Comprehensions

List comprehensions	228
Map comprehensions	229
Using list comprehension as a filter	230
Nested <code>for</code> s in a list comprehension.....	231

Chapter 23. Operations with Lists

Adding up lists.....	234
Reversing a list.....	234
Add up two lists	235
How to zip the lists.....	236
Count list items.....	237
Increment items in a list.....	238
Another way to increment items in a list.....	239
Difference of the lists	239
Removing duplicates from a list.....	240
Another method of de-duplicating a list.....	241

Merge the lists and preserve the order	242
Another way of merging lists	243
Print indices and values of a list.....	244
Concatenate a list of strings.....	245

Chapter 24. Multi-Dimensional Lists

Multi-dimensional lists.....	248
List of tuples	248
Select sublists based on criteria	249
Version 3 of converting list of tuples	250
Sort the lists by their second item	251
Nested subscripting	252
More on subscripting nested lists.....	252
Flatten a list	253
Flattening a list with iterators	254
Transpose a matrix.....	255
Another way to transpose a matrix	256
Hello, World! with NumPy	257
Transpose a matrix in NumPy	258
The T attribute in NumPy.....	259

Chapter 25. Iterators and Generators

Iterators	262
Using <code>yield</code>	263
Two generators.....	264
Custom iterator.....	265
Using <code>StopIteration</code>	266

Customr iterators in a loop.....	267
Chapter 26. Functional Programming	
Lambda function.....	270
Currying.....	271
Closure	271
More on closures	272
Chapter 27. Concurrent Programming	
Coroutines	276
Asynchronous coroutines	277
Run a thread.....	278
Thread speed.....	279

Part 5

Chapter 28. Object-Oriented Programming	
Classes	284
Class name	284
Count the objects	286
Objects to string.....	287
Empty objects.....	287
Using <code>with</code> with classes	288
<code>__init__</code> and <code>__del__</code>	290
The <code>@classmethod</code> and <code>@staticmethod</code> decorators.....	291
Using <code>dir</code> with objects	292
Polymorphic methods	293
Using <code>super</code>	294

Using <code>dir</code> with classes	295
A custom <code>dir</code>	297
Custom <code>len</code>	297
Using <code>__repr__</code>	298
<code>__str__</code> vs. <code>__repr__</code>	299
Sorting objects.....	300
More on sorting objects.....	301
<code>+</code> as a <code>-</code>	302
Comparing objects.....	303
Matrix multiplication <code>@</code>	304
Class operators	305
Deep copy of a list.....	306

Part 6

Chapter 29. Numerical Sequences

Factorial	312
Recursive factorial	312
Another recursive factorial	313
Computing a factorial	313
Prime numbers	314
Fibonacci numbers.....	315
Armstrong number	316

Chapter 30. Algorithms

Bubble sort.....	320
Quick sort	321

Multiply by 2.....	322
Eratosthenes sieve.....	323

Chapter 31. Maths Problems

Euler problem #1	326
Minimum and maximum.....	326
Computing a product.....	327
Number of digits	328
Area of a triangle	329
More on area of a triangle.....	330
A recursive sum	331
Average value	332
Mean value.....	333
Median value	333
Compute the value of π	334
Approaching the value of π with Fractions	335

Chapter 32. String Problems

Strings to integers	338
Count capitals	339
Check if is a binary	340
Keep vowels	340
Reverse a number	341
Count letters	342
Reverse a sentence.....	344
Remove double spaces	344
Spell a number	345

Word to number.....	346
Balanced parentheses	347
Anagram test	348

Chapter 33. Interview Problems

Unique random values	352
Another version of unique randoms	353
Odd and even.....	354
Find a missing number	355
Another way to find a missing number.....	356
Find a duplicate	357
Calculator.....	358
Monte Carlo method	359
Count bits	360
Guess the number	360
Random histogram.....	361
Random Gauss.....	363
All unique.....	364
Find a repeating element.....	365
Are all items the same?	366
Compare the elements in two lists	368
Print a tree	369
Pascal's triangle.....	371

Part 7

Chapter 34. Internet and Web

Load a page by URL.....	376
Web server	376

Chapter 35. Graphics

Simple drawing.....	380
Opening an image	380
Image size.....	381
Rotate an image	381
Thumbnails	382
Thumbnail vs. resize	383
Pixel's colour.....	384

Chapter 36. Miscellaneous

Using the <code>pickle</code> module.....	386
Working with JSON	387
UUID	387
UUID v4.....	388
The <code>array</code> module	389
MySQL connection.....	390
Using memcached	391

Chapter 5

Strings

The length of a string

What is the length of the string?

Save the following file as a UTF-8 file:

```
string = '你好，世界！'

print(len(string)) # Length in characters

print(len(string.encode('utf-8')))
# Length in bytes (if it is UTF-8)
```

In Python 3, you mostly don't have to care about the encoding of text strings if you work with UTF-8.

To get a length of the string in characters, use the built-in `len` function.

To get a length in bytes, you need to explicitly convert the string to UTF-8 and get its length.

```
$ python3 len.py
6
18
```

Multi-line strings

Assign a long string to a variable.

Sometimes you have a long string, but you'd like to keep the lines of your program short. Or you have a string that contains a few logical lines (that is, the string has newline characters in it).

```
str = '''This is a multi-line
string. That you can continue
on another line pressing the
Enter key'''
print(str)
```

Use triple quotes ''' to create a multi-line string in Python. Just simple as that.

Take care of indentation within the quotes.

```
str2 = '''Notice the spaces
        on the left. Will they
        appear in the output?
...
print(str2)
```

Run the program:

```
$ python3 multiline.py
This is a multi-line
string. That you can continue
on another line pressing the
Enter key
Notice the spaces
        on the left. Will they
        appear in the output?
```

Indexing a string character

Print the first, the second, and the third character of a string.

Accessing single characters in a string is as easy as providing character's positions.

```
s = 'Hello, World!'
#   01234567...

print(s[0]) # H
print(s[1]) # e
print(s[7]) # W
```

Here is what the program prints:

```
$ python3 sting-index.py
H
e
W
```

Don't forget that Python counts characters from 0, so the index of the first character is 0, the index of the second character is 1 and so on.

```
s = 'Hello, World!'
#   ^^^^^^^^
#   01234567...
```

Taking the last character

Print the last character of the given string.

```
str = 'Hello, World!'
#           ^

print(str[-1])
```

Of course, you can first determine the length of the string, and then index the corresponding characters, but there is a simpler way.

To count characters from right to left, use negative indices. For instance, to get the last character, index it as [-1].

```
$ python3 last-character.py
!
```

The second-to-last character

Print the second-to-last character of the given string.

To take the second-to-last character you will index it as [-2]:

```
str = 'Hello, World!'
#           ^

print(str[-2]) # d
```

Here is the result:

```
$ python3 last-character-2.py
d
```

Getting a substring

From a given string, take and print its part.

To make it easier for understanding, the index counting of the characters are shown in the comment. In the second comment, the desired part is marked with the ^ characters in the comment.

```
str = 'Hello, World!'
#   0123456789012
#           ^^^^^

# I want to print "World"
```



```
str_world = str[7:12]
print(str_world)
```

With ranges, you specify the index of the first character to copy and the index of the character followed by the last one you need.

```
$ python3 substring.py
World
```

Notice that the second number (12 in the example) is the index of the character after the last character in the substring. The letter **W** has index 7, and the letter **d** has index 11, thus the range is [7:12].

String repetition

*Explaining the * operator when it is applied to strings.*

```
str = 'A'
long_str = str * 15
print(long_str)
```

In Python, you can use `*` to repeat the string the given number of times.

```
$ python3 string-repetition.py
AAAAAAAAAAAAAAAAAAAA
```

Reversing a string

Reverse a string and print it.

```
str = 'Hello, World!'
rev_str = str[::-1]

print(rev_str)
```

To reverse a string, walk along it using a range with the step `-1` from right to left. As we want the whole string to be processed, the beginning and the end of the range can be omitted: `[::-1]`. With such a range, index counting goes backwards.

```
$ python3 reverse-string.py
!dlroW ,olleH
```

Splitting a string

How to split a string into characters.

```
str = 'Hello, World!'

# First, print a character
print(str[3])

# Now, make a list out of a string
# Just convert a type:
chars = list(str)
print(type(chars))
print(chars[3])
```

To split a string into characters, convert the string to a list, and you'll get a list of separate characters. Having a list, you can access individual characters directly by indexing a string.

```
$ python3 split-string.py
l
<class 'list'>
l
```

Raw strings

Understanding raw strings in Python.

Here is a regular string:

```
print('Hello, \nWorld!')
```

And here is the so-called raw string. It is prefixed with `r`:

```
print(r'Hello, \nWorld!')
```

In the second example, no escaping happens, and the `\n` sequence appears in the output as is, not as a newline.

```
$ python3 raw-strings.py
Hello,
World!
Hello, \nWorld!
```

More on raw strings

Strings and raw strings.

Here is a regular string with an escaped character:

```
s1 = 'a\tb'
```

And here is a raw string, where `\t` is not escaped and will appear as is in printing:

```
s2 = r'a\tb'
```

```
print(s1)
print(s2)
```

Normally, a character prefixed with a backslash is a special character such as a newline `\n` or a tab `\t`.

In the raw strings, backslash has no special meaning. You prefix raw strings with `'r'`

```
$ python3 raw-string.py
a b
a\tb
```

Removing accents from a string

Removing accents from characters.

This program removes accents from the accented letters such as `é` or `ü`.

```
import unidecode

str = 'Café München'

str = unidecode.unidecode(str)
print(str)
```

One of the simplest ways to do that is to use a module called `unidecode`. It does not come with the default Python distribution but can be easily installed using `pip`.

```
$ python3 remove-accents.py
Cafe Munchen
```

Removing \n

Remove newline characters from the strings in a list.

You have a list of strings which have a newline character at the end and you want to remove the newlines from them.

```
data = [  
    'alpha\n',  
    'beta\n',  
    'gamma\n'  
]  
print(data)
```

```
data = [  
    s.strip() for s in data  
    # List comprehension  
]  
print(data)
```

Here, we are using the list comprehension that loops over the strings in `data` and calls the `strip` method on each of the strings. As a result, the data gets cleaned and you have pure strings without newlines in them.

```
$ python3 strip.py  
['alpha\n', 'beta\n', 'gamma\n']  
['alpha', 'beta', 'gamma']
```

A more complex stripping

Another way of removing the newline characters from the strings in a list.

Let's solve the previous task of removing `\n` from a list of strings using the built-in `map` function.

```
data = [  
    'alpha\n',  
    'beta\n',  
    'gamma\n'  
]  
print(data)  
  
def strip(s):  
    return s.strip()  
  
# Let's use "map" this time  
data = list(map(strip, data))  
# map returns an object of the "map" type,  
# so convert it to a list  
print(data)
```

`map` requires a function that takes one argument and returns one argument. Here, we define our own function called `strip` that simply calls the `strip` method on the string. When `map` loops over the data items, the function is called for each of them, and the result is collected in the resulting object.

```
$ python3 strip2.py  
['alpha\n', 'beta\n', 'gamma\n']  
['alpha', 'beta', 'gamma']
```

Using lambdas to strip a line

Even more complex way to remove the newline characters from the strings in a list.

Here is another solution of the same task of removing newline characters from the strings in a list.

```
data = [  
    'alpha\n',  
    'beta\n',  
    'gamma\n'  
]  
print(data)
```

Let's replace a separate function with a lambda.

```
data = list(map(  
    lambda s: s.strip(), data))  
  
print(data)
```

This code is based on the previous program, but instead of stripping a string in a separate function, it uses an inline lambda.

```
$ python3 strip3.py  
['alpha\n', 'beta\n', 'gamma\n']  
['alpha', 'beta', 'gamma']
```

ASCII values of the characters

Print ASCII values of all characters in a given string.

```
str = 'Hello'  
  
ascii = [ord(s) for s in str]  
  
print(ascii)
```

This program gets a string and prints a list containing the ASCII numbers of each character in that string.

Here, the built-in `ord` function is used to convert a character to its ASCII value (or, in other words, ASCII position number).

To make the code simple, list comprehension is used.

```
$ python3 str-char-nums.py  
[72, 101, 108, 108, 111]
```

Character Unicode names

Printing the Unicode names of the characters.

```
import unicodedata  
  
str = 'Héllö'  
  
for s in str:  
    print(unicodedata.name(s))
```

Each character has its name in the Unicode space. To get the names, use the `name` function from the `unicodedata` module (it comes with the Python distribution).

```
$ python3 unicode-name.py  
LATIN CAPITAL LETTER H  
LATIN SMALL LETTER E WITH ACUTE  
LATIN SMALL LETTER L WITH CEDILLA  
LATIN SMALL LETTER L WITH CEDILLA  
LATIN SMALL LETTER O WITH DIAERESIS
```


Using casefolding

Handling the German letter ß (which is equivalent to 'ss').

```
str1 = 'strasse'
str2 = 'straße' # street in English

print(str1 == str2) # False

print(
    str1.casefold() ==
    str2.casefold()
) # True
```

Before talking about actual normalisation in Unicode, let's see at how the German letter ß can be compared with its equivalent sequence ss (let's omit the tiny grammatical differences that may occur). Use the `casefold` string method, which not only converts the string to lowercase but also take care of converting ß to ss. For example, `'Straße'.casefold()` gives the string `'strasse'`.

```
$ python3 unicode-normalize.py
False
True
```

Unicode normalization

Canonical normalization in Unicode.

Unicode normalization is required to get different texts to the common representation. For example, the letter á can be represented in two ways: by a single character and by a pair of codepoints, a Latin letter a

and a combining character for the acute. You can see them both in the following program.

```
import unicodedata

s1 = 'á'
s2 = 'a\u0301' # combining acute

print(s1 == s2) # False

s3 = unicodedata.normalize('NFC', s2)
print(s1 == s3) # True now
```

In the Unicode, ‘complex’ characters can be represented either by their direct code points (if they exist) or by a sequence of a character modified by combining characters (such as acute). To compare strings encoded differently, you need to normalize them first. Use the `normalize` function from the `unicodedata` module.

```
$ python3 unicode-normalize2.py
False
True
```

Split words

Two methods to split a word into separate letters.

In this program, two methods of splitting a word into letters are demonstrated.

```
word = 'Hello'

# Method 1: converting to a list
letters1 = list(word)
```

```
# Method 2: using list comprehension
letters2 = [ch for ch in word]

print(letters1)
print(letters2)
```

In the first method, you just call the `list` function and convert a string to a list, which ends up with a list of characters.

In the second method, you use list comprehension to loop over the characters in a string (`ch in word`) and collect them to a list.

```
$ python3 split-word.py
['H', 'e', 'l', 'l', 'o']
['H', 'e', 'l', 'l', 'o']
```