Python Basic Track

LCfP Staff

September 30, 2017

Contents

Introduction

0.1 About this book

0.2 About the authors

0.2.1 Vincent Velthuizen

0.2.2 Niels Wouda

0.2.3 Nick Szirbik

0.3 Acknowledgements

Chapter 1

What it is and what it isn't

1.1 Computers

1.2 Programming

1.3 Software engineering

1.4 This course

Part I

Codecademy course

Chapter 2

Python syntax

2.1 Variables

2.1.1 Datatypes

int, float, bool

2.1.2 Duck typing

`https://en.wikipedia.org/wiki/Duck_typing`

2.2 Whitespace

2.2.1 Keep code together

2.3 Comments

2.4 Arithmetic operations

2.5 Apply these concepts

2.5.1 Tip calculator

Chapter 3

Strings & Console Output

TL;DR

- We can combine simple variables, like characters;
- into more complex ones, like strings.
- Use `len()` to get the length of a string.
- Use `.upper()` and `.lower()` to get upper- and lowercase versions of a string.

3.1 Strings

In the previous chapter you have seen some basic data types. One of the basic types is the 'character'. As the name suggests this type of variable can represent any[1]single character. Often being able to represent a single character will not be enough. After all, characters are usually combined to form words, sentences, paragraphs, etc.

To help us do this a new type of variabel was created. It is called a because it represents a string of characters. This concept is available in most (if not all) programming language but can have slight variations. Here we will focus on how strings work in python.

To create a string we need to tell the system where the string starts and were it ends. Like in most languages you can use `"` and `'`. Since these are characters themselves we cannot just use them inside of a string. We need to 'escape' them by putting a \ in front of them. That makes \ a special character in its own right requiring it to be escaped as well. An overview of common escape sequences is given in Table 3.1.

Table 3.1: Common escape sequences

Sequence	Represents
`"\'"`	'
`"\""`	"
`"\\"`	\
`"\n"`	newline
`"\t"`	tab

3.2 Index

As we have established a string is a list of characters. Since it is a list it makes since to think of concepts like: 'first', 'second' and 'last' element. It is important to note that computer scientists count slightly different compared to what you may be used to. The 'first' element of the list is considered element 0, the 'second' element 1, etc.

Look at the code in Listing 3.1. Between the square brackets (`[ ]`) we specify the index of the character we want. An alternative to thinking about this as 'counting from 0', is to look at it as an offset (which it is). The variable `language` points to the 'p' (start of the list). Thus, if we want that 'p' we want `language` + 0. For the second character we want `language` +1, etc.

p	y	t	h	o	n
0	1	2	3	4	5

[1]Clearly there is a limited 'characterset', but if you stay within the characters used in English you should be safe. More about this topic later.

Listing 3.1: Getting characters from a string.

```
language = 'Python'
first = language[0]
second = language[1]
last = language[5]
```

Also note that the highest index equals the length of the list MINUS 1. Since the string 'python' has 6 characters the highest index is 5. If we ask for 6th element in this string our program will crash! Don't worry when this happens, but try to understand what happend (and how to solve it).

3.3 (String) Methods

The designers of python have written pieces of code so you do not have to start from scratch. A piece of reusable code is called a method. Using a method is referred to as 'calling' a method. We 'call' upon the method to do its job after which we can continue doing what we were doing. Often we 'call' a method to answer a question, in that case the method will 'return' an answer.

Let's look at this in action with a view examples.

`len()` The method `len()` can give us the length of a string. We give it a string between the parentheses, as shown in Listing 3.2. It will return an integer which is the length of the list.

Listing 3.2: Getting the length of a string.

```
language = 'Python'
length = len(language) # length = 6
last = language[length - 1]
```

When the method finishes imagine it being replaced by the answer it returned. So when the computer arrives at line 2 of Listing 3.2 it notices the method, jumps out of our code and into the `len()` code, computes the length and finally jumps back to our line 2 replacing '`len(language)`' with 6. Then line 2 gets execute assigning 6 as the value of the variable `length`.

`len()` actually works on any list in python.

`upper()` and `lower()` These methods work slightly differently from `len()`. You can think of them as actions a string can perform on itself.[2] Instead of passing the string

[2]Don't worry if that doesn't make sense. We will spend extensive time on this concept at a later point.

between the parentheses we use a dot (.) after the variable name. What gets returned is a changed version of the variable. As the names of these variables suggest we get an upper- or lowercase version of the string respectively. See Listing 3.3 for an example.

Listing 3.3: Getting the upper- and lowercase versions of a string.

```
language = 'Python'
upper = language.upper() # upper = 'PYTHON'
lower = language.lower() # lower = 'python'
```

3.4 Print

3.4.1 Concatenation

3.4.2 Explicit string conversion

3.4.3 String formatting

3.5 Apply these concepts

3.5.1 Date and Time

Chapter 4

Conditionals and Control Flow

Chapter 5

Functions

Chapter 6

Lists & Dictionaries

Index