Part 1: Python basics

Topics

- Variables
- Operators
- Selection
- String functions

Statements

A computer program is essentially a series of instructions which enables a computer to perform a certain action. In the Python language, such instructions are referred to as statements. They are delineated by hard returns. The end of the statement (i.e. the hard return) is a cue which triggers the Python parser to interpret and to execute the statement.

Printing output

Programs are generally written to solve specific problems, and they usually produce some output. This output can be a number, resulting from a calculation, for instance. The print() function can be used to communicate the output of the program to the user of the program. As will be explained in more detail later, a function is a named set of statements which collectively represent a certain action. The print() function is always used with parentheses. Inside the parentheses, you can provide the text that you would like the computer to display.

```
print('This program works!')
```

You can also format the text, to some extent. The character '\t' prints a tab, and '\n' creates a hard return.

Variables

Virtually all programming languages make use of variables of some sort. Variables can be thought of as containers or as boxes in which information can be stored temporarily. Variables are always given a name, and they can also be given a value.

Variables names can consist of any combination of alphanumerical characters that you can think of. Underscores are also allowed in variable names. One important rule, however, is that variable names cannot begin with a number. Although it is possible to work with any sequence of characters, your script will be most understandable, evidently, when you make use of variable names which are

meaningful. Unlike other languages, there is no special symbol which indicates that the variable is indeed a variable.[1] Note that there are a number of reserved words which have a specific meaning in the Python language, and which cannot be used as variable names for this reason. The list of reserved words include 'for', 'in', 'is', 'print', 'global', 'def', 'I'f, 'elif' and 'else'.

As mentioned, variables can be given a value. The act of giving a value to a certain variable is called assignment. In the first example, the variable named placeName is assigned the value 'Leiden'.

```
placeName = 'Leiden'
numberOfinhabitant = 123924
```

Values are always of a specific type. In the example above, the variable placeName is assigned a string value. Simply put, a string is a sequence of characters and spaces. Strings can be created either with single quotes or with double quotes. The variable numberofInhabitants is an example of an integer (i.e. a natural number). Next to such integers, Python can also work with floating point numbers.

```
pi = 3.14159265359
```

The type of a variable can be found by using the type() function, as follows:

```
age = 20
print( type(age) )
## this will print <class 'int'>
```

Comments

Incidentally, the example above also illustrates another feature of the Pyhton language. Lines can be preceded by the hash ('#') symbol. Lines that start with a hash are called comments. Such comments are often very useful when you want to document your code. They can be used to explain what happens exactly in specific parts of the code. Comments are ignored by the Python interpreter.

Operators

[1] In the Perl programming language, for instance, all variable names must be preceded by a percentage sign, e.g. $age

Computer are generally written to process certain data values. Values can be manipulated by making use of so-called operators. They are symbols that represent specific actions. They can be applied to values, or to variables representing certain values. The values that are combined with these operators are called the operands.

Numerical values can be used in combination with mathematical operators.

+	Addition
/	Division
*	Multiplication

Listing 3. illustrates the function of a number of mathematical operators.

```
 1  # Firstly, we create two values a and b
 2  # and give them numerical values
 3
 4  a = 4
 5  b = 3
 6
 7  c = a + b
 8  # c now has value 7
 9
10  c = a * b ;
11  #c now has value 12
12
13  c = a - b
14  # c has value 1
15
16  c = c+1
17  #c now has value 2
18
19  c+= 1
20  #c now has value 3. The syntax used in line 19 is
21  #a shorthand notation for the statement given on line 16.
```

Listing 1.1.

Boolean operators

Boolean operators are operators which compare values. The results of such comparisons can be either true or false. The following Boolean operators can be used in combination with numerical values:

==	Equal to

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

When operands are combined with operators, this combination is referred to as an expression. This expression can be evaluated to a new value. An expression is not yet a statement in itself, however; it is not an independent instruction for the computer. An expression can be used in an assignment operation, as was illustrated in listing 1.1. Boolean expressions are commonly used in statements which determine the flow control.

Flow Control

By default, the Python moves through the code in a linear way. It simply executes all the statements in the order in which they are found in the programme. In some cases, however, a statement only needs to be executed under a certain condition. Alternatively, statements, or clusters of statements, may also need to be repeated as long as a certain condition applies.

In Python, there are various keywords that you can use to specify if, and how many times, a statement needs to be executed. These keywords are referred to collectively as flow control structures. These flow control structures typically make use of certain tests or conditions. This is typically a Boolean expression which, in a given situation, can either be true or false.

If a certain section of your program must only be executed if a certain condition is true, you can use the keyword 'if'. When the expression following 'if' is true, the statements underneath the expression are executed. If not, these statements are ignored. The listing below gives an example:

```
1   age = 25
2
3   if age > 20:
4       print("you are older than 20!")
5
```

Listing 1.2.

Importantly, the spatial layout of the code is not arbitrary. The block of code that must executed when the condition is true is indented using four spaces. In most code editors, the indentation that is created when you hit the tab button has exactly the length as four spaces. The statements that have the same indentation are all assumed to belong to the same block of code.

Listing 1 only contains one condition. It is also possible, however, to let Python evaluate a series of conditions. In this case, you need to use ‘if, elif and else. The basic structure is as follows.

```
1   time = 22
2
3   if time < 12:
4       print('Good morning!')
5   elif time < 18:
6       print('Good afternoon!')
7   else:
8       print('Good evening')
9
```

Listing 1.3.

If the condition that follows the if keyword can be evaluated as "true", the first code block will be executed. If not, the condition that is given after elif will be evaluated. Python will execute the second block of code if that second condition is found to be “true”. The final set of statements will be executed only if all the two earlier conditions are evaluated to “false”.

Note that only the keywords ‘if’ and ‘elif’ can be followed by a condition. The keyword ‘else’ always appears WITHOUT such a condition. The code block given after else contains the actions that must be performed if all the earlier conditions are false.

There are also flow control structures that can be used for repetitions of statements. These will be discussed in part 2 of this tutorial.

Operators for strings

The operators discussed above focused on operators which can be used to manipulate the value of numbers. One of the operators that you can use in combination with strings is the concatenation operator. Its symbol is the plus (‘+’). You can use this operator to combine two existing strings into a longer string.

Listing 4 is a simple example.

```
1   first = "Jane"
2   last = "Austen"
3   full = first + " " + last
4
5   print(‘Pride and Prejudice was written by ' + full)
```

Listing 1.4.

Listing 1.4 actually contains a number of concatenations. The variables 'first' and 'second' are combined into the variable named full. A very short string, consisting only of a space, is placed in between these two strings. Line 4 uses this full variable in a sentence which is printed.

As was explained, a string is essentially a sequence of characters. All the characters in such strings are also numbered. These numbers are referred to as indices. Python start counting at 0. Using these indices, it becomes possible to extract specific characters from strings. For this purpose, the index needs to provided in square brackets. When you use negative numbers (e.g. -1 or -2), Python starts selecting characters at the end of the string and moves back.

```
 1  author = 'William Shakespeare'
 2
 3  print( author[0] )
 4  ## This prints 'W'
 5
 6  print(author[4])
 7  ## This prints 'i'
 8
 9  print (author[-1])
10  ## This prints 'e'
```

Listing 1.5.

You can extract a range of characters from a given string by providing the start position and the end position of the substring that you want to create. These two indices must be separated by a colon.

```
1  title = 'Romeo and Juliet'
2
3  print( title[0:5] )
4  # this prints 'Romeo'
5
6  print( title[10:16])
7  # this prints 'Juliet'
```

Listing 1.6.

Functions and methods for strings

The len() is an in-built function which can return the length of a string. This function measures the length by counting the number of characters.

The value a of string variable can also be manipulated by making use of methods. A method is similar to a function. A method, like a function, is a word that represents a certain action. While functions can be used independently, methods are always associated with an object such as a string. We can call a method by appending the name of the method to the name of the variable. The variable and the method must be delimited by a period. The following methods are available for string objects:

upper()	Converts all the characters of the string into upper case.
lower()	Converts all the characters of the string into lower case.
strip()	Removes all white space (such as spaces, hard returns or tabs) from the beginning and the end of a string

Listing 1.7 Gives an indication of how these methods may be used.

```
 1  title = "   The Hitchhiker's Guide to the Galaxy   "
 2
 3  title = title.strip()
 4  #this removes the spaces before and after the original
 5  string
 6
 7  print( len(title) )
 8  ## This prints 36
 9
10  print( title.upper() )
11  ## This prints 'THE HITCHHIKER'S GUIDE TO THE GALAXY'
```

Listing 1.7.