PYTHON COMPENDIUM

WRITTEN BY

ANRICH TAIT

7 FEBRUARY 2023

Contents

1 SYNOPSIS

This document is a compendium of my python studies. It's main purpose is to act as study notes and a quick reference for specific information. The document is split into defined headings and sub-headings relating to each topic.

1.1 HOW TO USE THIS DOCUMENT

- For digital users: You will see that each source code block starts with a heading. In this heading you need to ensure that both "python" and ":results output" are included.
- "python" states what language the block is.
- ":results output" is the command that outputs the code in a block underneath the main block.
- For hard-copy users: The code blocks are used in this capacity as both examples and a learning oppurtunity.
- I suggest reading a code block and then prediciting what the output will be to practice your code recognition.

1.2 EXAMPLE TEMPLATE

```
print("THIS IS A TEMPLATE")
```

Output:

```
THIS IS A TEMPLATE
```

2 THE BASICS

2.1 INTRO

2.1.1 What is python?

Python is a popular programming language created by Guido van Rossum and released in 1991. `https://en.wikipedia.org/wiki/Guido_van_Rossum`

It is used for:

- web development (server-side),
- software development,
- mathematics,
- sytem scripting

What can python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping or for production-ready software development.

Why python?

- Python works on different platforms (Windows, Mac, Linux, Rasberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know:

- The most recent major version of Python is Python 3, which we shall be using in this tutotial. However, Python 2, although not being updated with anything other than security updates is still popular.

Python Syntax compared to other programming languages:

- Python was designed for readability and has some similarities to the English language with influence from mathematics.

- Python uses new lines to complete a command, as opposed to other languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

2.2 THE COMMAND LINE

- To test a small amount of code it is sometimes best to use the python command line instead of creating a file.
- You can type the following into a command line to open the python command line: "python" or "py" Then enter your code, for example: print("This is an example!") To exit the python command line enter: "exit()".

2.3 SYNTAX

2.3.1 Indentation

Indentation refers to the spaces at the beginning of a line of code. Indentation is very important in Python due to it using this to determine a block of code. The number of spaces is up to the programmer but four is the most common with one being the least possible. You have to set the same number of spaces in a block of code or python will output an error. Example of indentation:

```
if 5 > 2:
    print("Five is greater than two!")
```

Output:

```
Five is greater than two!
```

Notice the indentation before "print". If that indentation was not there you would receive an error.

3 S

3.1 WHAT ARE COMMENTS?

- Comments can be used to explain python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

3.2 CREATING A COMMENT

- Comments start with "#".
- They can be placed at the end of a line.
- Comments can also be used to prevent code from being run.
- Alternatively you can use a multiline string that is not assigned to a variable to make multiline comments.

```
# You can add multiple comments simply by inserting "#" before the text.
print("See all the uses for comments?") # Even at the end of lines
#print("This line won't be output!")
"""
This is also a comment
but uses multiline
strings that are not
assigned a variable!
"""
```

Output:

```
See all the uses for comments?
```

4 VARIABLES

Variables are containers for storing data values.

4.1 CREATING VARIABLES

Python has no command for declaring a variable. A variable is created simply by assigning a value to it. Variables don't need to be declared with any types (like C++ where int or string must be declared) and variables can even change types after being set.

```
x = 5       # x is an int type
y = "John"  # y is a str type
y = 5       # y is now also an int type
print(x)
print(y)
```

Output:

```
5
5
```

4.2 CASTING

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)  #x will be '3' (string)
y = int(3)  #y will be 3 (integer)
z = float(3)#z will be 3.0 (float)

print(x)
print(y)
print(z)
```

Output:

```
3
3
3.0
```

4.3 GET TYPE

You can get the data type of a variable with the "type()" function.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Output:

```
<class 'int'>
<class 'str'>
```

It is important to note that you can use either single or double quotes to declate a string variable, and variables are case sensitive. This means that variable "A" and variable "a" are different and will not overwrite each other.

```
a = 4
A = "John"
print(a)
print(A)
```

Output:

```
4
John
```

4.4 VARIABLE NAMES

Rules:

- A variable name must start with a leter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores(A-z, 0-9 and _).
- Variable names are case-sensitive (age, Age, and AGE are different variables)

```
# These are different ways to set variable names:
myvar = "myvar"
my_Var = "my_Var"
_my_var = "_my_var"
myVar = "myVar"
MYVAR = "MYVAR"
myvar2 = "myvar2"
# These are ways NOT to set variable names:
# 2myar
# my-var
#my var
print(myvar)
print(my_Var)
print(_my_var)
print(myVar)
print(MYVAR)
print(myvar2)
```

Output:

```
myvar
my_Var
_my_var
myVar
MYVAR
myvar2
```

4.5 ASSIGN MULTIPLE VALUES

4.5.1 MANY VALUES TO MULITPLE VARIABLES

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cheery"
print(x)
print(y)
print(z)
```

Output:

```
Orange
Banana
Cheery
```

4.5.2 ONE VALUE TO MULTIPLE VARIABLES

You can also assign the same value to multiple variables in one line.

```
x = y =z = "Orange"
print(x)
print(y)
print(z)
```

Output:

```
Orange
Orange
Orange
```

4.5.3 UNPACKING A COLLECTION

If you have a collection of values in a list, tuple, etc. You can extract the values into variables, this is called unpacking.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Output:

```
apple
banana
cherry
```

4.6 OUTPUT VARIABLES

The Python "print()" function is often used to output variables.

- In the "print()" function you can output multiple variables by seperating them with a comma:

```
x = "Python"
y = "is"
z = "tasty!"
print(x, y, z)
```

Output:

```
Python is tasty!
```

4.6.1 + OPERATOR

You can also use the "+" operator to output multiple variables. But keep in mind that then you need to include a space after each variable to ensure that the output remains coherent. This operator is mainly used for mathematical purposes.

```
#remember to add a space at the end of a string!
textOne = "Python "
textTwo = "is "
textThree = "tasty!"
print(textOne + textTwo +textThree)
#for mathematics
x = 5
y = 10
print(x + y)
```

Output:

```
Python is tasty!
15
```

Keep in mind that you cannot use the "+" operator to combine a string and a number. The best way to output multiple variables in the "print()" function is to seperate them with commas.

4.7 GLOBAL VARIABLES

Variables that are created outside of a function are known as global variables. Global variables can b used by everyone both inside and outside of functions.

If you create a variable with the same name inside a function this variable will be local and can only be used inside the function. The global variable with the same name will remain as it was.

```
x = "tasty" #this is the global variable

def myfunc():
    x = "fantastic" #this is the local variable
    print("Python is " + x)

myfunc()

print("Python is " + x)
```

Output:

```
Python is fantastic
Python is tasty
```

4.7.1 GLOBAL KEYWORD

Normally when you create a variable inside a function that variable is local and can only be used inside that function. To create a global variable inside a function you use the "global" keyword.

```
#To create a global variable inside a function you use the "global" keyword.
def myFuncOne():
    global x
    x = "tasty"

myFuncOne()
print("x: Python is " + x)
#You can also use the global keyword if you want to change a global variable inside a #fun
y = "awesome"

def myFuncTwo():
    global y
    y = "awesome"

myFuncTwo()
print("y: Python is " + y)
```

Output:

```
x: Python is tasty
y: Python is awesome
```

5 DATA TYPES

5.1 BUILT IN DATA TYPES

Variables can store data of different types and different types in turn do different things. Python has the following data types built-in.

TYPE	IDENTIFIER
text	str
numeric	int, float, complex
sequence	list, tuple, range
mapping	dict
set	set, frozenset
boolean	bool
binary	bytes, bytearray,memoryview
none	NoneType

5.2 GETTING THE DATA TYPE

You can get the data type of any object by using the "type()" function.

```
x = 5
print(type(x))
```

Output:

```
<class 'int'>
```

5.3 SETTING THE DATA TYPE

In python variables are automatically assigned a type when you assign them a value.

EXAMPLE	DATA TYPE
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = frozenset ({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview
x = None	NoneType

5.4 SETTING A SPECIFIC DATA TYPE

If you want to specify a data type you can use the following constructor functions:

EXAMPLE	DATA TYPE
x = str(“Hello World”)	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list((“apple”, “banana”, “etc”))	list
x = tuple((“apple”, “banana”, “etc”))	tuple
x = range(6)	range
x = dict(name=“John”, age=36)	dict
x = set((“apple”, “banana”, “etc”))	set
x = frozenset((“apple”, “banana”, “etc”))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

6 NUMBERS

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric typs are created when you assign values to them.

```
x = 1       #int
y = 2.8     #float
z = 1j      #complex

#Remember to use the "type()" function to output a variable's type.
print(type(x))
print(type(y))
print(type(z))
```

Output:

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

6.1 int

Int or integer is a whole number, positive or negative withut decimals of unlimited length.

```
x = 1
y = 123456789
z = -123456789

print(type(x))
print(type(y))
print(type(z))
```

Output:

```
<class 'int'>
<class 'int'>
<class 'int'>
```

6.2 float

Float or floating point number is a number, positive or negative containing one or more decimals. Floats can also be scientific numbers with an "e" to indicate the power of 10.

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(z))
print(type(y))

a = 35e3
b = 12E4
c = -87.7e100

print(type(a))
print(type(b))
print(type(c))
```

Output:

```
<class 'float'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'float'>
```

6.3 complex

Complex numbers are written with a "j" as the imaginary part.

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

6.4 TYPE CONVERSION

You can convert from one type to another with the "int()" "float()" and "complex()" methods.

```
x = 1   #int
y = 2.8 #float
z = 1j  #complex

#convert from int to float
a = float(x)
```

```
#convert from float to int
b = int(y)

#convert from int to complex
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

Output:

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Note that you cannot convert complex numbers into another number type.

6.5 RANDOM NUMBERS

Python does not have a "random()" function to make a random numer, but you can use the built-in "random" module to do this.

```
import random

print(random.randrange(1, 10)) #prints a random number within the range of 1 and 10
```

7 CASTING

When you need to specify a type on to a variable you need to use constructor functions. Python is an object-orientated language and therefore uses classes to define data type, including it's primitive types.

- int(): contructs an integer number from an integer literal or float literal (removes all decimals), or a string literal (providing the string represents a whole number) -float(): constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer). -str(): constructs a string from a wide variety of data type, including strings, integer literals and float literals.

7.1 INTEGERS EXAMPLE

```
x = int(1)      #x will be 1
y = int(2.8)    #y will be 2
z = int("3")    #z will be 3

print(x)
print(y)
print(z)
```

Output:

```
1
2
3
```

7.2 FLOATS EXAMPLE:

```
w = float("4.2")    #w will be 4.2
x = float(1)        #x will be 1.0
y = float(2.8)      #y will be 2.8
z = float("3")      #z will be 3.0

print(w)
print(x)
print(y)
print(z)
```

Output:

```
4.2
1.0
2.8
3.0
```

7.3 STRINGS EXAMPLE

```
x = str("s1")   #x will be 's1'
y = str(2)      #y will be '2'
z = str(3.0)    #z will be '3.0'

print(x)
print(y)
print(z)
```

Output:

```
s1
2
3.0
```

8 STRINGS

Strings in python are surrounded by either single quotation marks or double (' ') (“ ”). Therefore 'test' is the same as “test”. You can display a string literal with the “print()” function.

```
print('test')
print("test")
```

Output:

```
test
test
```

8.1 ASSIGN STRING TO A VARIABLE

Assigning a string to a variable is done with the variable name followed by an equal sign and the string.

```
a = "Test"
print(a)
```

Output:

```
Test
```

8.2 MULTILINE STRINGS

You can assign a multiline string to a variable by using three quotes:

```
a = """There is nothing outside of yourself that can ever enable you to get better, strong
print(a)
```

Output:

```
There is nothing outside of yourself that can ever enable you to get better, stronger, ric
```

Note that in the output the line breaks are inserted at the same position as the code.

8.3 STRINGS ARE ARRAYS

Like many other popular programming languages, strings in Python are are arrays of bytes representing unicode characters. However Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string. !!!Remember that the first character in a string has the value of “0”.

```
a = "Strings are arrays!"
print(a[1])
print(a[5])
```

Output:

```
t
g
```

8.4 LOOPING THROUGH STRINGS

Since strings are arrays we can loop through the characters in a string with a "for" loop.

```
for x in "banana":
    print(x)
```

Output:

```
b
a
n
a
n
a
```

8.5 STRING LENGTH

To get the length of a string, use the "len()" function.

```
a = "Testing string length"
print(len(a))
```

Output:

```
21
```

8.6 CHECK STRING

To check if a certain phrase or character is present in a string we can use the keyword "in".

```
txt = "The best things in life are free!"
print("free" in txt)
```

Output:

```
True
```

Using "in" with an "if" statement:

```
txt = "The best things in life are free!"
if "free" in txt:
    print("Yes, 'free' is present!")
```

Output:

```
Yes, 'free' is present!
```

8.7 CHECK “if” NOT

To check if a certain phrase or character is NOT present in a string we use the “not in” keyword.

```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

Output:

```
True
```

Using “if not” with an “if” statement:

```
txt = "The best things in life are free!"
if "expensive" not in txt:
    print("No, 'expensive' is NOT present.")
```

Output:

```
No, 'expensive' is NOT present.
```

8.8 SLICING STRINGS

You can return a range of characters by using the slice syntax. Specify the start index and the end index, seperated by a colon to return a part of the string.

```
b = "Slicing strings!"
print(b[2:7])
```

Output:

```
icing
```

8.8.1 SLICE FROM START

By leaving out the start idex, the range will start at the first character (0).

```
b = "Slicing strings!"
print(b[:7])
```

Output:

```
Slicing
```

8.8.2 SLICE TO END

By leaving out the end index, the range will end at the last character.

```
b = "Slicing strings!"
print(b[8:])
```

Output:

```
strings!
```

8.8.3 NEGATIVE INDEXING

Use negative indexes to start the slice from the end of the string. Example: Get the characters: From: “o” in “World” (position 5) TO, but not included: “d” in “World” (position 2)

```
b = "Hello World!"
print(b[-5:-2])
```

Output:

```
orl
```

8.9 MODIFY STRINGS

Python has a set of built-in methods that you can use on strings.

8.9.1 UPPER CASE

The “upper()” method returns the string in upper case:

```
a = "Upper case!"
print(a.upper())
```

Output:

```
UPPER CASE!
```

8.9.2 LOWER CASE

The “lower()” method returns the string in lower case:

```
a = "Lower case."
print(a.lower())
```

output:

```
lower case.
```

8.9.3 REMOVE WHITESPACE

Whitespace is the space before and after the actual text and very often you want to remove this. This is done by using the “strip()” method.

```
a = " Remove all whitespace! "
print(a.strip())
```

Output:

```
Remove all whitespace!
```

8.9.4 REPLACE STRING

The "replace()" method replaces a string with another string.

```
a = "Replace a string!"
print(a.replace("Replace", "Cut"))
```

Output:

```
Cut a string!
```

8.9.5 SPLIT A STRING

The "split()" method returns a list where the text between the specified seperator becomes the list items. The "split()" method splits the string into substrings if it finds instances of the seperator.

```
a = "Split this, string!"
print(a.split(","))
```

Output:

```
['Split this', ' string!']
```

8.10 CONCATENATION

To concatenate or combine two strings use the "+" operator.

```
a = "Hello"
b = "world"
c = a + " " + b
print(c)
```

Output:

```
Hello world
```

Note the use of " " to create a space between the strings.

8.11 STRING FORMAT

- Remember that strings and numbers cannot be combined using the "+" operator.
- To combine strings and numbers we need to use the "format()" method.
- The "format()" method takes the passed arguments, formats them and places them in the string where the placeholders "{ }" are.
- The "format()" method takes unlimited number of arguments and are placed into the respective placeholders.

- You can use index numbers {0} to be sure the arguments are placed in the correct placeholders.

```
quantity = 3
itemNo = 567
price = 49.95
myOrder = "I want to pay {2} Rand for {0} pieces of item {1}."
print(myOrder.format(quantity, itemNo, price))
```

Output:

```
I want to pay 49.95 Rand for 3 pieces of item 567.
```