Python Programming

Chapter 3

Start using Python

In this chapter we will start to use Python in some simple examples.

3.1 Python IDE

The basic code editor, or an integrated development environment, called IDLE. See Figure 3.1.

Other Python Editors will be discussed more in detail later. For now you can use the basic Python IDE (IDLE) or Spyder if you have installed the Anaconda distribution package.

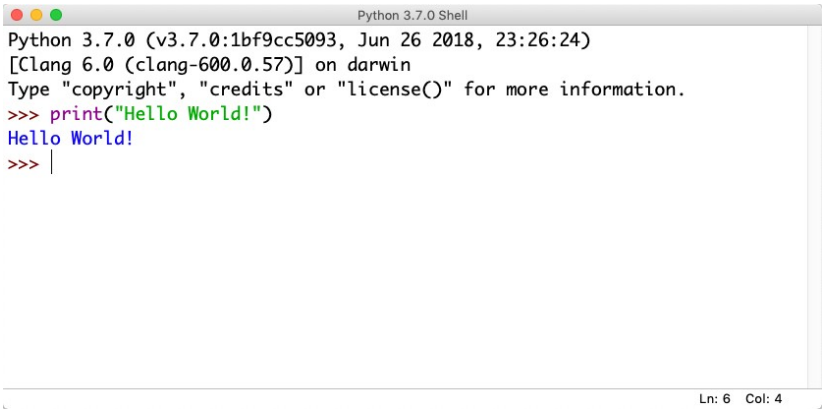

Figure 3.1: Python Shell / Python IDLE Editor

3.2 My first Python program

We will start using Python and create some code examples.

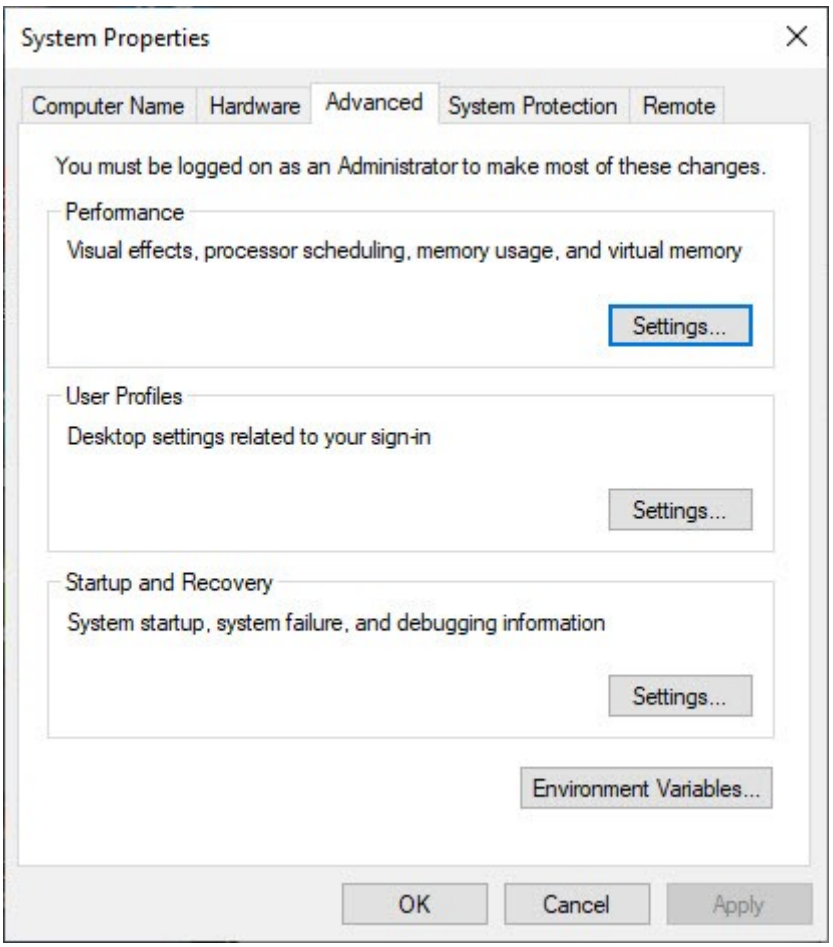

Figure 3.4: Windows System Properties

In the next window, find and select the user variable named Path and click Edit. . . to change its value. See Figure 3.5.

Select ”New” and add the path where ”python.exe” is located. See Figure 3.6.

The Default Location is:

```
C: \ Users \ user \AppData\ Local \ Programs\ Python\ Python37 -32\
```

Click Save and open the Command Prompt once more and enter ”python” to verify it works. See Figure 3.3.

Chapter 4

Basic Python Programming

4.1 Basic Python Program

We will start using Python and create some code examples.

We use the basic IDLE editor (or another Python Editor)

Example 4.1.1. Hello World Example

Lets open your Python Editor and type the following:

```
1 print("Hello World!")
```

Listing 4.1: Hello World Python Example

[End of Example]

4.1.1 Get Help

An extremely useful command is **help()**, which enters a help functionality to explore all the stuff python lets you do, right from the interpreter.

Press q to close the help window and return to the Python prompt.

4.2 Variables

Variables are defined with the assignment operator, "=". Python is dynamically typed, meaning that variables can be assigned without declaring their type, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.
Python

Example 4.2.1. Creating and using Variables in Python

We use the basic IDLE (or another Python Editor) and type the following:

```
>>> x = 3
>>> x
3
```

Listing 4.2: Using Variables in Python

Here we define a variable and sets the value equal to 3 and then print the result to the screen.

[End of Example]

You can write one command by time in the IDLE. If you quit IDLE the variables and data are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script.

Python scripts or programs are save as a text file with the extension **.py**

Example 4.2.2. Calculations in Python

We can use variables in a calculation like this:

```
x = 3
y = 3*x
print(y)
```

Listing 4.3: Using and Printing Variables in Python

We can implementing the formula $y = ax + b$ like this:

```
a = 2
b = 5
x = 3

y = a*x + b

print(y)
```

Listing 4.4: Calculations in Python

As seen in the examples, you can use the *print()* command in order to show the values on the screen.

[End of Example]

Variables of numeric types are created when you assign a value to them, so in normal coding you don't need to bother.

Example 4.2.3. Numeric Types in Python

```
x = 1       # int
y = 2.8     # float
z = 3 +2j   # complex
```

Listing 4.5: Numeric Types in Python

This means you just assign values to a variable without worrying about what kind of data type it is.

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

Listing 4.6: Check Data Types in Python

If you use the Spyder Editor, you can see the data types that a variable has using the Variable Explorer (Figure 4.2):

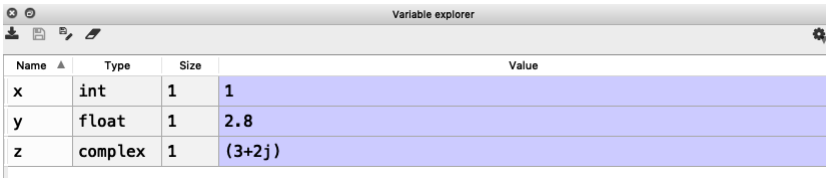

Figure 4.2: Variable Editor in Spyder

[End of Example]

4.2.2 Strings

Strings in Python are surrounded by either single quotation marks, or double quotation marks. 'Hello' is the same as "Hello".
Strings can be output to screen using the print function. For example: print("Hello").

Example 4.2.4. Plotting in Python

Below we see examples of using strings in Python:

```
a = " He l l o World ! "

print (a)

print(a[1])
print(a[2:5])
print(len(a))
print(a.lower())
```

```
print(a.upper())
print(a.replace("H", "J"))
print(a.split(" "))
```

Listing 4.7: Strings in Python

As you see in the example, there are many built-in functions form manipulating strings in Python. The Example shows only a few of them.

Strings in Python are arrays of bytes, and we can use index to get a specific character within the string as shown in the example code.

[End of Example]

4.2.3 String Input

Python allows for command line input.

That means we are able to ask the user for input.

Example 4.2.5. Plotting in Python

The following example asks for the user's name, then, by using the input() method, the program prints the name to the screen:

```
print(" Enter your name : ")
x = input()
print(" Hello , " + x)
```

Listing 4.8: String Input

[End of Example]

4.3 Built-in Functions

Python consists of lots of built-in functions. Some examples are the print(9 function that we already have used (perhaps without noticing it is actually a Built-in function).

Python also consists of different Modules, Libraries or Packages. These Modules, Libraries or Packages consists of lots of predefined functions for different topics or areas, such as mathematics, plotting, handling database systems, etc. See Section 4.4 for more information and details regarding this.

In another chapter we will learn to create our own functions from scratch.

4.4 Python Standard Library

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs.
The **Python Standard Library** consists of different modules for handling file I/O, basic mathematics, etc. You don't need to install these separately, but you need to important them when you want to use some of these modules or some of the functions within these modules.

The math module has all the basic math functions you need, such as: Trigonometric functions: *sin(x)*, *cos(x)*, etc. Logarithmic functions: *log()*, *log10()*, etc. Constants like *pi*, *e*, *inf*, *nan*, etc. etc.

Example 4.4.1. Using the math module

We create some basic examples how to use a Library, a Package or a Module:

If we need only the sin() function we can do like this:

```
from math import sin

x = 3.14
y = sin(x)

print(y)
```

If we need a few functions we can do like this

```
from math import sin, cos

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

If we need many functions we can do like this:

```
from math import *

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

We can also use this alternative:

```
import math

x = 3.14
y = math.sin(x)

print(y)
```

We can also write it like this:

```
import math as mt

x = 3.14
y = mt.sin(x)

print(y)
```

[End of Example]

There are advantages and disadvantages with the different approaches. In your program you may need to use functions from many different modules or packages. If you import the whole module instead of just the function(s) you need you use more of the computer memory.

Very often we also need to import and use multiple libraries where the different libraries have some functions with the same name but different use.

Other useful modules in the **Python Standard Library** are **statistics** (where you have functions like *mean()*, *stdev()*, etc.)

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/index.html

4.5 Using Python Libraries, Packages and Mod- ules

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This approach has advantages and disadvantages. An disadvantage is that you need to install these packages separately and then later import these modules in your code.

Some important packages are:

- **NumPy** - NumPy is the fundamental package for scientific computing with Python
- **SciPy** - SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.
- **Matplotlib** - Matplotlib is a Python 2D plotting library

Plotting functions that you will use a lot:

- plot()
- title()
- xlabel()
- ylabel()
- axis()
- grid()
- subplot()
- legend()
- show()

Lets create some basic plotting examples using the Matplotlib library:

Example 4.6.1. Plotting in Python

In this example we have to arrays with data. We want to plot x vs. y. We can assume x is a time series and y is the corresponding temperature i degrees Celsius.

```
import matplotlib.pyplot as plt

x = [1,2,3,4,5,6,7,8,9,10]

y = [5,2,4,4,8,7,4,8,10,9]

plt.plot(x,y)
plt.xlabel('Time(s)')
plt.ylabel('Temperature(degC)')
plt.show()
```

We get the following plot:
We can also write like this:

```
from matplotlib.pyplot import *

x = [1,2,3,4,5,6,7,8,9,10]
y = [5,2,4,4,8,7,4,8,10,9]

plot(x,y)
xlabel('Time(s)')
ylabel('Temperature(degC)')
show()
```

This makes the code simpler to read. one problem with this approach appears assuming we import and use multiple libraries and the different libraries have some functions with the same name but different use.

Part II

Python Programming

Chapter 5

Python Programming

We have been through the basics in Python, such as variables, using some basic built-in functions, basic plotting, etc.

You may come far only using these thins, but to create real applications, you need to know about and use features like:

- If ... Else
- For Loops
- While Loops
- Arrays ...

If you are familiar with one or more other programming language, these features should be familiar and known to you. All programming languages has these features built-in, but the syntax is slightly different from one language to another.

5.1 If ... Else

An ”if statement” is written by using the **if** keyword.

Here are some Examples how you use a If sentences in Python:

Example 5.1.1. Using For Loops in Python

```
a = 5
b = 8

if a > b:
    print("a is greater than b")

if b > a:
    print("b is greater than a")

if a == b:
    print("a is equal to b")
```

Listing 5.1: Using Arrays in Python

Try to change the values for a and b.

Using **If - Else**:

```
a = 5
b = 8

if a > b:
    print("a is greater than b")
else:
    print("b is greater than a or a and b are equal")
```

Listing 5.2: Using Arrays in Python

Using **Elif**:

```
a = 5
b = 8

if a > b:
    print("a is  greater than b")
elif b > a:
    print("b is  greater than a")
elif a == b:
    print("a is  equal to b")
```

Listing 5.3: Using Arrays in Python

Note! Python uses "elif" not "elseif" like many other programming languages do.

[End of Example]

5.2 Arrays

An array is a special variable, which can hold more than one value at a time.

Here are some Examples how you can create and use Arrays in Python:

Example 5.2.1. Using For Loops in Python

```
data = [1.6, 3.4, 5.5, 9.4]

N = len(data)

print(N)

print(data[2])

data[2] = 7.3

print(data[2])


for x in data:
  print(x)
```

```

data . append ( 1 1 . 4 )


N = l e n ( data )

p r i n t (N)


f o r x in data :
  print (x)
```

Listing 5.4: Using Arrays in Python

You define an array like this:

```
data = [1.6, 3.4, 5.5, 9.4]
```

You can also use text like this:

```
carlist = ["Volvo", "Tesla", "Ford"]
```

You can use Arrays in Loops like this:

```
for x in data :
  print(x)
```

You can return the number of elements in the array like this:

```
N = len(data)
```

You can get a specific value inside the array like this:

```
index = 2
x = c ars [ index ]
```

You can use the append() method to add an element to an array:

```
data . append (11.4)
```

[End of Example]

You have many built in methods you can use in combination with arrays, like sort(), clear(), copy(), count(), insert(), remove(), etc.

You should look test all these methods.

5.3 For Loops

A For loop is used for iterating over a sequence. I guess all your programs will use one or more For loops. So if you have not used For loops before, make sure to learn it now.

Below you see a basic example how you can use a For loop in Python:

```
for i in range(1,10):
  print(i)
```

The For loop is probably one of the most useful feature in Python (or in any kind of programming language). Below you will see different examples how you can use a For loop in Python.

Example 5.3.1. Using For Loops in Python

```
data = [1.6, 3.4, 5.5, 9.4]

for x in data:
  print(x)


carlist = ["Volvo", "Tesla", "Ford"]

for car in carlist:
  print(car)
```

Listing 5.5: Using For Loops in Python

The range() function is handy yo use in For Loops:

```
N = 10

for x in range(N):
  print(x)
```

The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

You can also use the range() function like this:

```
start=4
stop= 12 #but not including

for x in range(start, stop):
  print(x)
```

Finally, you can also use the range() function like this:

```
start=4
stop = 12 #but not including
step = 2

for x in range(start, stop, step):
  print(x)
```

You should try all these examples in order to learn the basic structure of a For loop.

[End of Example]

Example 5.3.2. Using For Loops for Summation of Data

You typically want to use a For loop for find the sum of a given data set.

```
data = [ 1 , 5 , 6 , 3 , 12 , 3 ]

sum = 0

#Find the Sum o f a l l the numbers
f o r x in data :
    sum = sum + x

p r i n t ( sum)

#Find the Mean or Average o f a l l the numbers

N = l e n ( data )

mean = sum/N

p r i n t ( mean)
```

This gives the following results:

```
30
5 . 0
```

[End of Example]

Example 5.3.3. Implementing Fibonacci Numbers Using a For Loop in Python

Fibonacci numbers are used in the analysis of financial markets, in strategies such as Fibonacci retracement, and are used in computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure.
They also appear in biological settings, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone.

In mathematics, Fibonacci numbers are the numbers in the following sequence: 0, 1, 1, 2 ,3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two.

Some sources omit the initial 0, instead beginning the sequence with two 1s.

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$f_n = f_{n-1} + f_{n-2} \tag{5.1}$$

with seed values:

$$f_0 = 0, f_1 = 1$$

We will write a Python script that calculates the N first Fibonacci numbers. The Python Script becomes like this:

```
N = 10

fib1 = 0
fib2 = 1

print(fib1)
print(fib2)

for k in range(N-2):
    fib_next = fib2 + fib1
    fib1 = fib2
    fib2 = fib_next
    print(fib_next)
```

Listing 5.6: Fibonacci Numbers Using a For Loop in Python

Alternative solution:

```
N = 10

fib = [0,1]


for k in range(N-2):
    fib_next = fib[k+1] + fib[k]
    fib.append(fib_next)

print(fib)
```

Listing 5.7: Fibonacci Numbers Using a For Loop in Python - Alt2

Another alternative solution:

```
N = 10

fib = []

for k in range(N):
    fib.append(0)

fib[0] = 0
fib[1] = 1

```

```
for k in range (N-2):
    fib[k+2] = fib[k+1] +fib[k]


print(fib)
```

Listing 5.8: Fibonacci Numbers Using a For Loop in Python - Alt3

Another alternative solution:

```
import numpy as np


N = 10

fib = np.zeros(N)

fib[0] = 0
fib[1] = 1

for k in range (N-2):
    fib[k+2] = fib[k+1] +fib[k]


print(fib)
```

Listing 5.9: Fibonacci Numbers Using a For Loop in Python - Alt4

[End of Example]

5.3.1 Nested For Loops

In Python and other programming languages you can use one loop inside another loop.

Syntax for nested For loops in Python:

```
for iterating var in sequence:
    for iterating var in sequence:
        statements(s)
    statements(s)
```

Simple example:

```
for i in range(1,10):
    for k in range(1, 10):
        print(i, k)
```

Exercise 5.3.1. Prime Numbers

The first 25 prime numbers (all the prime numbers less than 100) are:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

By definition a prime number has both 1 and itself as a divisor. If it has any other divisor, it cannot be prime.

A natural number (1, 2, 3, 4, 5, 6, etc.) is called a prime number (or a prime) if it is greater than 1 and cannot be written as a product of two natural numbers that are both smaller than it.

Create a Python Script where you find all prime numbers between 1 and 200.

Tip! I guess this can be done in many different ways, but one way is to use 2 nested For Loops.

[End of Exercise]

5.4 While Loops

The while loop repeats a group of statements an indefinite number of times under control of a logical condition.

Example 5.4.1. Using While Loops in Python

```
m = 8

while m > 2 :
    print (m)
    m = m - 1
```

Listing 5.10: Using While Loops in Python

[End of Example]

5.5 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 5.5.1. Plot of Dynamic System

Given the autonomous system:

$$\dot{x} = ax \tag{5.2}$$

Where:

$$a = -\frac{1}{T}$$

Chapter 9

File Handling in Python

9.1 Introduction

Python has several functions for creating, reading, updating, and deleting files. The key function for working with files in Python is the **open()** function.

The open() function takes two parameters; Filename, and Mode.

There are four different methods (modes) for opening a file:

- ”x” - Create - Creates the specified file, returns an error if the file exists
- ”w” - Write - Opens a file for writing, creates the file if it does not exist
- ”r” - Read - Default value. Opens a file for reading, error if the file does not exist
- ”a” - Append - Opens a file for appending, creates the file if it does not exist

In addition you can specify if the file should be handled as binary or text mode

- ”t” - Text - Default value. Text mode
- ”b” - Binary - Binary mode (e.g. images)

9.2 Write Data to a File

To create a **New** file in Python, use the open() method, with one of the following parameters:

- ”x” - Create - Creates the specified file, returns an error if the file exists
- ”w” - Write - Opens a file for writing, creates the file if it does not exist
- ”a” - Append - Opens a file for appending, creates the file if it does not exist

To write to an **Existing** file, you must add a parameter to the open() function:

- ”w” - Write - Opens a file for writing, creates the file if it does not exist
- ”a” - Append - Opens a file for appending, creates the file if it does not exist

Example 9.2.1. Write Data to a File

```
f= open (" my f i l e . txt ", "x")

data = " Helo  World"

f. write (data)

f. c l o s e ()
```

Listing 9.1: Write Data to a File

[End of Example]

9.3 Read Data from a File

To read to an existing file, you must add the following parameter to the open() function:

- ”r” - Read - Default value. Opens a file for reading, error if the file does not exist

Example 9.3.1. Read Data from a File

```
f= open (" my f i l e . txt ", " r ")

data = f. read ()

print (data)

f. c l o s e ()
```

Listing 9.2: Read Data from a File

[End of Example]

9.4 Logging Data to File

Typically you want to write multiple data to the, e.g., assume you read some temperature data at regular intervals and then you want to save the temperature values to a File.

Example 9.4.1. Logging Data to File

Chapter 10

Error Handling in Python

10.1 Introduction to Error Handling

So far error messages haven't been discussed. You could say that we have 2 kinds of errors: syntax errors and exceptions.

10.1.1 Syntax Errors

Below we see an example of syntax errors:

```
>>> print(He llo World)
  File "<ipython-input-1-10cb182148e3>", line 1
    print(Hello World)
                    ^
SyntaxError: invalid syntax
```

In the example we have written print(Hello World) instead of print("Hello World") and then the Python Interpreter gives us an error message.

10.1.2 Exceptions

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called exceptions and are not unconditionally fatal: you will soon learn how to handle them in Python programs. Most exceptions are not handled by programs, however, and result in error messages as shown here:

```
>>> 10 * (1/0)
Traceback (most recent call last):

  File "<ipython-input-2-0b280f36835c>", line 1, in <module>
    10 * (1/0)

ZeroDivisionError: division by zero
```

or:

```
>>> '2'+2
Traceback (most recent call last):

```

```
  File "<ipython-input-3-d2b23a1db757>", line 1, in <module>
    '2' + 2

TypeError: must be str, not int
```

10.2 Exceptions Handling

It is possible to write programs that handle selected exceptions.

In Python we can use the following built-in Exceptions Handling features:

- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the **try** - **except** statements.

Some basic example:

```
try:
    10 * (1/0)
except:
    print("The calculation failed")
```

or:

```
try:
  print(x)
except:
  print("x is not defined")
```

You can also use multiple exceptions:

```
try:
  print(x)
except NameError:
  print("x is not defined")
except:
  print("Something is wrong")
```

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example: