**Scientific computing with Python**

**Chapter 1:**

**Getting Started**

In this chapter, we provide a concise overview of Python’s core syntactical elements.

1. **Installation and configuration**

This book focuses on the Python 3.x series, specifically up to version 3.7.

To follow along with this book, you will need to install the following components:

* **Anaconda**: Even if Python is already pre-installed on your computer, we strongly recommend creating a personal Python environment.
* **Spyder**:
* Open the Anaconda Prompt
* Launch the spyder with “start spyder”

Most Python codes will be collected in files. We recommend that you use the following header in all your Python files:

***from numpy import \****

***from matplotlib.pyplot import \****

* **Python shell**: IPython can be started in different ways
* In a terminal shell by running the following command: *ipython*
* By directly clicking on an icon called *Jupyter QT Console*
* When working with Spyder, you should use an *IPython console*
* **Executing Scripts:** You need to specify the full path or navigate to the directory containing the file before running it.
* Use the command cd in IPython in order to move to the directory where your file is located.
* To execute the contents of a file named myfile.py, just run the following command in the IPython shell: *run myfile*
* **Getting help:**
* To get help on an object, just type ? after the object's name and then press the Return key.
* Use the arrow keys to reuse the last executed commands.
* Use *Ctrl+D* to quit.
* Use IPython's magic functions. You can find a list and explanations by applying %magic on the command prompt.
* **Jupyter Notebook:** You can invoke the notebook by running the following command in the terminal window:

*jupyter notebook*

A browser window will open and you can interact with Python through your web browser.

1. **Program and program flow**

A program is a sequence of statements that are executed in top-down order. This linear execution order has some important exceptions :

* There might be a conditional execution of alternative groups of statements (blocks), which we refer to as branching.
* There are blocks that are executed repetitively, which is called looping.
* There are function calls that are references to another piece of code.
* **Comments**: If a line in a program contains the symbol **#**, everything following on the same line is considered as a comment.
* **Line joining**: A backslash **\** at the end of the line marks the next line as a continuation line, that is, explicit line joining.

1. **Basic data type in python**

* **Numbers**: A number can be an integer, a real number, or a complex number.
* **Strings:** Strings are sequences of characters, enclosed by Single, double or triple quotes: **‘ ’ , “ ”, “”” Long string “”” .**
* **Variables**: Variables are created when you assign a value to a name, and they can hold different types of data such as numbers, text, lists, or more complex objects.

The value of a variable can be displayed by the ***print*** function.

* **Lists:** A list is an ordered, mutable collection of items in Python. Lists can hold elements of different data types, including numbers, strings, and even other lists.

The indexing of the elements starts at zero. You can put objects of any type inside a list, even other lists. Some basic list functions are as follows :

* **list(range(n))** creates a list with n elements, starting with zero.
* **len** gives the length of a list.
* **append** is used to append an element to a list.
* **Boolean expressions:** A Boolean expression is a logical statement that evaluates to one of two values: **True** or **False**.
* **Comparison operators:  
  ==** (equal to), **!=** (not equal to), **<** (less than), **>** (greater than), **<=** (less than or equal to), **>=** (greater than or equal to)
* **Logical operators:**  
  **and** (logical AND), **or** (logical OR), **not** (logical NOT)

1. **Repeating statements with loops**

Loops allow you to execute a block of code repeatedly, either a specific number of times or as long as a condition remains true.

* **Repeating a task**: This code uses a for loop with the range function to repeat the print(i) statement five times, with i taking values from 1 to 5.

for i in range(1, 6):

print(i)

* **Break and else:** The **break** statement allows you to exit a loop immediately.

for number in range(1, 5):

if number == 3:

break # Exit the loop when number equals 3

print(number)

The finalizing else checks whether the for loop was broken with the break keyword. If it was not broken, the block following the else keyword is executed.

1. **Conditional Statements**

This section covers how to use conditions for branching, breaking, or otherwise controlling your code.

x = 10

if x >= 0:

print("x is positive")

else:

print("x is negative")

1. **Encapsulating code with functions**

Functions are useful for gathering similar pieces of code in one place.

**def f(x):**

**return 2\*x + 1**

1. **Understanding scripts and modules**

A script is a file containing a sequence of Python statements that are executed from top to bottom when run.

In a Python or IPython shell, such a script can then be executed with the exec command after opening and reading the file. Written as a one-liner, it reads as follows: **exec(open('smartscript.py').read()).**

The IPython shell provides the magic command %run as a handy alternative way to execute a script**: %run smartscript.**

1. **Python interpreter**

The Python interpreter executes the following steps:

* First, it checks the syntax.
* Then it executes the code line by line.
* . First, it checks the syntax. Then it executes the code line by line. The code inside a function or class declaration is not executed, but its syntax is checked.

**Chapter 2:**

**Variables and Basic Types**

In this chapter, we will present the most important and basic types in Python.

1. **Variables**

A variable in Python is a named reference to a value stored in memory. You create a variable using the assignment operator **=.**

The variables:

* can store any type of data
* Do not need explicit type declarations

1. **Numeric types**

In Python, like many other computer languages, we have numeric types: **int**, **float**, **complex**.

* **Integer (int):** The simplest numeric type.
* **Floating-point number (float)** : is a number that has a decimal point or is written using scientific notation.

Operations between floating-point numbers rarely return the exact result.

* **Complex numbers:** Complex numbers are an extension of the real numbers frequently used in many scientific and engineering fields.

1. **Booleans**

A Boolean variable can take only two values, True or False. The main use of this type is in logical expressions.

* **Boolean operators:** Boolean operations are performed using the keywords and, or, and not.
* **Boolean casting:** Most Python objects may be converted to Booleans; this is called Boolean casting.
* **Automatic Boolean casting :** Using an if statement with a non-Boolean type will cast it to a Boolean. In other words, the following two statements are always equivalent:

if a:

...

if bool(a): # exactly the same as above

...

1. **Strings:** The type string is a type used for text**.**

A string is enclosed either by single or double quotes. If a string contains several lines, it has to be enclosed by three double quotes """ or three single quotes '''.

* **Escape sequences and raw strings :** The string '**\n'** is used to insert a line break and '**\t**' inserts a horizontal tabulator (TAB) into the string to align several lines: **print('Temperature\t20\tC\nPressure\t5\tPa')**
* **Operations on strings and string methods:**

The addition of several strings results in their concatenation:

last\_name = 'Carlsson'

first\_name = 'Johanna'

full\_name = first\_name + ' ' + last\_name **# returns 'Johanna Carlsson'**

Consequently, multiplication by an integer is repeated addition:

game = 2 \* 'Yo' # returns 'YoYo'

Multiplication by floating-point or complex numbers is undefined and results in a TypeError.

**Chapter 3:**

**Container Types**

Container types are used to group objects together. The main difference between the different container types is the way individual elements are accessed and how operations are defined.

1. **Lists**

A list in Python is a flexible container that can store a collection of items in a specific order. You can add, remove, or change the items in a list at any time.

**L = ['a', 20.0, 5]**

**M = [3, ['a', -3.0, 5]]**

**L [1]** # returns 20.0

**L [0]** # returns 'a'

**M [1]** # returns ['a', -3.0,5]

**M [1][2]** # returns 5

A list containing subsequent integers can easily be generated by the command range: **L=list(range(4))** # generates a list with four elements: [0, 1, 2 ,3].

The command len returns the length of the list:

**len(L)** # returns 3

* **Slicing:** Like cutting a slice from a loaf of bread, lists can be cut into slices. Slicing a list between i and j creates a new list containing the elements starting at index i and ending just before j.

Here, L[i:] means remove the first elements and L[:i] means take only the first elements:

**L = ['C', 'l', 'o', 'u', 'd', 's']**

**L[1:5]** # remove one element and take four from there:

# returns ['l', 'o', 'u', 'd']

You may omit the first or last bound of the slicing:

**L = ['C', 'l', 'o', 'u', 'd', 's']**

**L[1:]** # ['l', 'o', 'u', 'd', 's']

**L[:5]** # ['C', 'l', 'o', 'u', 'd']

**L[:]** # the entire list

* **Strides:** When computing slices, you may also specify a stride, which is the length of the step from one index to the other. The default stride is 1. Here is an example:

**L = list(range(100))**

**L[:10:2] # [0, 2, 4, 6, 8]**

**L[::20] # [0, 20, 40, 60, 80]**

**L[10:20:3] # [10, 13, 16, 19]**

Note that the stride may also be negative:

**L[20:10:-3] # [20, 17, 14, 11]**

* **List methods:**

**list.append(x)** # Add x to the end of the list.

**list.extend(L) #** Extend the list by the elements of the list L.

**list.insert(i,x)** # Insert x at position i.

**list.remove(x) #** Remove the first item from the list whose value is x.

**list.sort() #** Sort the items of the list.

**list.reverse() #** Reverse the elements of the list.

**list.pop() #** Remove the last element of the list.

* **Merging Lists**: The **zip()** function in Python is used to combine two or more lists into a single list.

**ind = [0,1,2,3,4]**

**color = ["red", "green", "blue", "alpha"]**

**list(zip(color,ind))** # gives [('red', 0), ('green', 1), ('blue', 2), ('alpha', 3)]

* **List comprehension:**  is a compact way to create a new list by performing an operation on each item in an existing sequence.

The syntax of a list comprehension is :

**[<exp> for <variable> in <list> if <condition>]**

**L = [2, 3, 10, 1, 5]**

**L2 = [x\*2 for x in L] # [4, 6, 20, 2, 10]**

**L3 = [x\*2 for x in L if 4 < x <= 10] # [20, 10]**

1. **Arrays**

The NumPy package offers arrays, which are container structures for manipulating vectors, matrices, or even higher-order tensors in mathematics.

**v = array([1.,2.,3.])**

**A = array([[1.,2.,3.],[4.,5.,6.]])**

1. **Tuples**

A tuple is an immutable list. Immutable means that it cannot be modified.

**my\_tuple = 1, 2, 3** # our first tuple

**my\_tuple = (1, 2, 3)** # the same

**my\_tuple = 1, 2, 3,** # again the same

**len(my\_tuple)** # 3, same as for lists

**my\_tuple[0] = 'a'** # error! tuples are immutable

1. **Dictionaries**

* **Creating and altering dictionaries :**

**truck\_wheel = {'name':'wheel','mass':5.7, 'Ix':20.0,'Iy':1.,'Iz':17., 'center of mass':[0.,0.,0.]}**

The command dict generates a dictionary from a list with key/value pairs: **truck\_wheel = dict([('name','wheel'),('mass',5.7), ('Ix',20.0), ('Iy',1.), ('Iz',17.)**

* **Looping over dictionaries :**

There are mainly three ways to loop over dictionaries:

* **By Keys**
* **By Value**
* **By item**

1. **Sets**

The last container object we introduce in this section is defined by the data type **set**.

Sets are containers that share properties and operations with sets in mathematics. A mathematical set is a collection of distinct objects.

Like in mathematics, in Python the elements of a set are also listed within a pair of braces.