Chapter 35
Introduction to Python

35.1 Structure

The programming language Python is an interpreter language that allows both the interactive input of commands in a console as well as the execution of programs or scripts, which are saved as a sequence of commands in a file. Python is well-known for its good readability of programs, the implementation in common operating systems and the availability of extensive libraries and modules. Variables in Python do not need to be declared and can be used directly. It is important to maintain block structures by indenting commands and marking their beginning with a colon. In Fig. 35.1 two simple Python programs are shown, which are saved as text files with the specified filenames. They are for example started with the command

```
$ python3 comp_square.py
```

from a console.

Typically, each instruction is written into a separate line. A longer command can be continued into further lines with a backslash. To write multiple commands in one line, they are separated with a semicolon. Programs usually start with the inclusion of modules and the definition of functions.

S. Bartels, *Numerical Mathematics 3x9*, La Matematica per il 3+2 160,
https://doi.org/10.1007/978-3-662-70890-3_35

```python
# comp_square.py
def square(x):
    x_sq = x**2
    return x_sq
x = 3.8
y = square(x)
print(y)
```

```python
# simple_loop.py
import math
n = 10
delta_x = math.pi/n
for i in range(n):
    x = math.sin(i*delta_x)
    print(i,x)
```

Fig. 35.1 Elementary programs in Python; indentations are essential components for marking program blocks

35.2 Elementary Commands

The basic arithmetic operations and the input and output of variables are directly available in Python. The assignment of values is done with an equals sign. The command `print` displays variables and text flexibly and always ends automatically with a line break. Single-line comments are marked with a hash, multi-line comments with triple quotation marks. Further mathematical functions are provided by the modules `math` and `numpy`, which are included via the command `import`. To avoid cumbersome commands, a library can be renamed when included using `as`. The output of floating point numbers can be formatted, for example by:

```
print("x = {:>7.4f}".format(x))
```

Here, space for seven characters is reserved right-aligned and four decimal places of the variable `x` are displayed. Table 35.1 provides an overview of some commands.

Table 35.1 Input and output functions, comments, importing of modules as well as mathematical functions

`print`, `input`	Output and input of text and variables
`\n`	Generation of a line break
`"""..."""`, `#`	Multi-line and single-line comments
`**`, `%`	Power and remainder in division
`a+=b`	Short form for `a=a+b`
`import [as]`	Inclusion of libraries
`e`, `pi`	Euler's number and π
`cos`, `sin`, `tan`	Trigonometric functions
`exp`, `log`, `log10`	Exponential function and logarithms
`pow`, `sqrt`	Power and square root
`floor`, `ceil`, `abs`	Rounding to whole numbers and absolute function

Table 35.2 Immediately available variable types in Python

`int`	Integer machine numbers
`float`	Double precision floating point numbers
`bool`	Boolean variables with values `false` and `true`
`str`	Strings
`list`	Lists

```
if (condA): statementA elif (condB): statementB else: statementC
while (cond): statement
for i in <enumerating object>: statement
```

Fig. 35.2 Control structures in Python: conditional, repetition and enumeration

35.3 Types

Python provides common data types for working with variables, see Table 35.2. The conversion of a variable or object is done, for example, using `i = int(str)`.

Although Python automatically performs type conversions, so for example `1/2` yields the result `0.5`, Python relies on various C++ libraries where this is usually not the case. Floating point numbers should therefore be specified with a point for safety, such as `1./2.` or `1.0/2.0`.

35.4 Control Statements

Conditional statements and loops in Python have a special structure. The blocks to be executed are introduced with a colon. For `for`-loops, the specification of an enumerating object is required, over which the iteration is performed. For this, the command `range(start,stop,step)` is often used, which generates a sequence of numbers. The specifications of the start value and the step size are optional, if they are omitted, then 0 and 1 are used. The stop value is not included in the list, for example

$$\texttt{range(n)} \equiv (0, 1, \ldots, n-1)$$

is an enumeration of the integers from 0 to $n-1$. The syntactic structures of the most important control statements are shown in Fig. 35.2.

35.5 Logical Expressions

To formulate conditions in conditional statements and repetitions, logical expressions are needed, which can be constructed using the comparisons and conjunctions shown in Table 35.3.

35.6 Functions

Functions begin with the command `def` and can have no or several arguments. They usually return one or more values with the command `return`. This command can be omitted if a variable to be changed is passed by *call by reference* and directly modified by the function. Lists are always passed in this way, so the explicit return can and should be omitted. If a function does not contain a return command, the value `None` is automatically returned. The exemplary use of functions is shown in Fig. 35.3.

35.7 Lists

Python allows the direct definition of lists or vectors without special commands for declaration or changing their length. The elements of an array are separated by commas and enclosed in square brackets:

```
x = [1.0,2.3,0.7]
```

Access to one or more elements of a list is also done using square brackets, the indexing of lists starts with index zero. The formal addition of two lists using `x+y` results in their combination into a larger list. The most important list operations are

Table 35.3 Logical operations for formulating conditions

==, !=, >, >=, <, <=	Logical comparison of machine numbers
`in`	Logical query for containment
`and`, `or`, `not`	Logical conjunctions *and*, *or* and negation

```
# example_function.py
def two_values(x,y):
    u = x*y; v = x+y
    return u, v
a = 2.0; b = 1.0
[c,d] = two_values(a,b)
print(c,d)
```

```
# function_list.py
def several_squares(v):
    for i in range(len(v)):
        v[i] = v[i]**2
x = [1,2,3,4];
several_squares(x)
print(x)
```

Fig. 35.3 Definition and call of functions with and without explicit return of values

Table 35.4 Methods for editing lists

`[]`	Empty list
`[0]*n`	Create a list with n zero entries
`x[i]`, `x[-m]`, `x[i:s:j]`	Access to individual or multiple elements
`len(x)`	Returns the length of the list `x`
`x.append(a)`	Appends an element at the end of the list
`x.sort()`	Sorts a list
`max(x)`, `min(x)`	Maximum and minimum of a list
`del x[i:j]`	Deletes a range of a list

```
# read_vector.py
def scan_vector(x,n):
    for i in range(n):
        str = input("x[i] = ")
        x.append(float(str))
x = []
str =  input("dim = ")
n = int(str)
scan_vector(x,n)
print(x)
```

```
# numpy_matrices.py
import numpy as np
m = 5; n = 3
A = np.zeros((m,n))
x = np.ones(n)
for i in range(m):
    for j in range(n):
        A[i][j] = i*n+j
y = np.matmul(A,x)
print(A,x,y)
```

Fig. 35.4 Reading a vector and matrix-vector multiplication

shown in Table 35.4. The program shown in Fig. 35.4 uses the list commands for interactive input of a vector.

For working with matrices, the `numpy` module with the multidimensional fields and matrix operations defined therein is recommended, a program example is shown in Fig. 35.4.

35.8 Timing, Saving and Plotting

To determine the runtime of a program, the `time` module provides the function `t = time.time()`, which returns an absolute time in seconds. To access a file, the command `open('file.txt','r')` is used when data is to be read. If a file is to be created or overwritten, the argument `'w'` is to be used, when appending data the argument `'a'`. The addition of the command `with ... as` serves for error handling and ensures the correct closing of the file. The `matplotlib` module provides routines for displaying graphs. Examples of the use of the commands and modules can be found in Figs. 35.5 and 35.6. The module `scipy` contains methods for the numerical solution of differential equations.

```python
# runtime.py
import time
t1 = time.time()
for i in range(10**6):
    x = i*i # i**2
t2 = time.time()
print("runtime = ",t2-t1)
```

```python
# save_data.py
x = [1.0,3.0,6.0]
with open('var.dat','w') as f:
    for i in range(len(x)):
        f.write("{:>10.4f}\n".\
                format(x[i]))
#
```

Fig. 35.5 Runtime measurement and data saving in Python

```python
# plot_function.py
import math
import matplotlib.pyplot as plt
n = 100; x = [0]*n;
y = [0]*n; z = [0]*n;
for i in range(n):
    x[i] = i*2.0*math.pi/n
    y[i] = math.sin(x[i])
    z[i] = math.cos(x[i])
plt.plot(x,y,"-b",label="sin")
plt.plot(x,z,":k",label="cos")
plt.legend(loc="lower left")
plt.show()
```

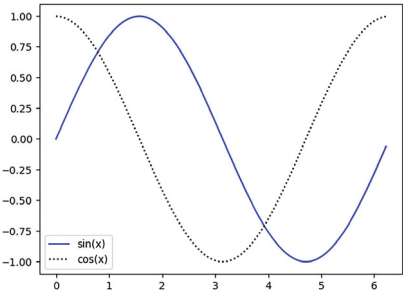

Fig. 35.6 Plotting of functions in Python