

Kapitel 1 (Allgemeines)

- Du kannst die Endung von Python-Programmen angeben.
- Du kannst das Kommentarsymbol von Python-Programmen angeben.
- Du kannst angeben, mit welcher Art von Programm man einen Python-Quelltext schreibt.
- Du kannst angeben, welcher Art von Programm einen Python-Quelltext ausführt.
- Du kannst die drei wesentlichen Typen von Programmierfehlern beschreiben und jeweils ein Beispiel dazu angeben.

Kapitel 2 (Einfache Datentypen)

- Datentypen: ganze Zahlen (integer), Gleitkommazahlen (float) und Zeichenketten (string) erkennen können.
- mathematische Operatoren: +, -, *, /, **, //, %
- mathematische Funktionen: derzeit nur `abs(...)`
- Wahrheitswerte: `True`, `False` und die Operatoren `not`, `and`, `or`
- Vergleichsoperatoren: `==`, `!=`, `<=`, `<`, `>=`, `>`
- Casting (Typumwandlungen): `int → float`, `int → bool`, `int → str`; `float → int`, `float → bool`, `float → str`; `bool → int`, `bool → float`, `bool → str`; `str → int`, `str → float`, `str → bool`;

Kapitel 3 (Variablen)

- Die Regel für die Bildung von Bezeichnern (Variablenamen): Das erste Zeichen muss ein Buchstabe oder ein Unterstrich sein. Danach dürfen weitere Buchstaben oder Unterstriche oder Ziffern folgen. Sonderzeichen wie +, -, *, #, ", @, ... sind nicht erlaubt.
- Zuweisungsoperator: `=`
- Mehrfachzuweisungen: z.B. `x, y, z = 5, 7, -3`
- Erweiterte Zuweisungsoperatoren: `+=`, `*=`, usw.

Kapitel 4 (Bedingte Anweisungen und Verzweigungen)

- bedingte Anweisungen:

```
if <bedingung>:
    <block>
```
- einfache Verzweigung:

```
if <bedingung>:
    <block1>
else:
    <block2>
```
- mehrfache Verzweigung:

```
if <bedingung1>:
    <block1>
elif <bedingung2>:
    <block2>
...
elif <bedingungN>:
    <blockN>
else:
    <block>
```
- Verschachtelte bedingte Anweisungen und Verzweigungen

Kapitel 5 (Schleifen)

- for-Schleife (Zählschleife): `for <var> in range(<start>,<stop>[,<step>]):`
 <block>
- for-Schleife für Listen `for <var> in [<item_1>, <item_2>,...,<item_n>]:`
 <block>
- while-Schleife: `while <bedingung>:`
 <block>
- Schleifenabbruch mit `break`
- Überspringen von Schleifendurchläufen mit `continue`

Kapitel 6 (Listen)

- Bildung von Listen mit `L = [...]`
- Listensyntax `L[0]`, `L[1]`, ...
- Bedeutung negativer Listenindizes wie `L[-1]`
- Slices: `L[a:b]`, `L[a:]`, `L[:b]`, `L[a:]`, `L[a:b:step]`
- Listenfunktionen: `len(L)`, `sorted(L)`, `max(L)`, `min(L)` `sum(L)`
- Listenoperatoren `+` und `*`
- Listenmethoden: `L.append(element)`, `L.pop()`, `L.pop(index)`, `L.insert(i, element)`, `L.index(element)`, `L.remove(element)`, `L.reverse()`,
Die Namen *element*, *index* und *list* stehen jeweils für ein beliebiges Element, einen erlaubten Index und eine beliebige Liste.
- Du weißt, dass das Kopieren einer Listenvariablen nur das Kopieren einer Listenreferenz bedeutet und dass eine Änderungen an der Kopie eine Änderung am Original bewirkt.
- Du weißt, dass eine Slice jeweils eine „echte“ Kopie einer (Teil-)Liste im Speicher erzeugt.
- Einfache List-Comprehensions

Kapitel 7 (Funktionen)

- Du kannst mindestens zwei Vorteile von Funktionen im Hinblick auf die Programmierung aufzählen.
 - Funktionen ermöglichen die *Wiederverwendung* von Programmcode.
 - Mit Funktionen lässt sich ein langes Programm in kleinere, besser lesbare Teile zerlegen (*Strukturierung*).
 - Sind Funktionen an einem zentralen Ort gespeichert, dann vereinfachen sie die *Pflege* (Verbesserungen und Anpassungen) von Programmcode.
- Du kannst die Syntax von Funktionsdefinitionen

```
def funktionsname(parameterliste):  
    funktionsblock  
    return wert
```

nachvollziehen und selber einfache Funktionen definieren.
- Du kennst den Unterschied zwischen *Funktionsdefinition* und *Funktionsanwendung* und kannst die folgenden zwei Mechanismen (*Positionsparameter*, *Schlüsselwortparameter*) anwenden, mit denen die *formalen* durch die *aktuellen* Parameter ersetzt werden. *Beispiel*:

```
def f(a, b, c):  
    return 100*a + 10*b + c
```

 - Positionsparameter (Reihenfolge muss stimmen)
`f(7, 1, 2) ⇒ a=7, b=1, c=2`
 - Schlüsselwortparameter (Reihenfolge ist „egal“)
`f(b=1, c=2, a=7) ⇒ a=7, b=1, c=2`
- Du weißt, dass Funktionsparameter und innerhalb einer Funktion definierte Variablen nur innerhalb dieser Funktion (*lokal*) sichtbar sind und allfällige ausserhalb der Funktion definierte Variablen gleichen Namens während der Ausführung der Funktion „überschatten“. *Ausnahme*: Steht in der Parameterliste eine Variable, die auf ein veränderliches Objekt verweist (z. eine Liste), dann wird zwar eine lokale Variable erzeugt, die jedoch auf das gleiche Objekt verweist, so dass eine Veränderung des

- Du kannst Funktionen mit Rückgabewerten innerhalb von zusammengesetzten Ausdrücken auswerten.
- Du kannst einfache rekursive (sich selbst aufrufende) Funktionen nachvollziehen.

Kapitel 8 (Zeichenketten)

- Syntax von Zeichenketten mit `'...'` oder `"..."`
- String-Operatoren: `+` und `*`
- String-Funktion: vorläufig nur `len(<str>)`
- Die String-Methode `str.format(...)` mit `{...}`-Ersetzungen

Kapitel 9 (Ein- und Ausgabe)

- `print()`-Anweisung (zusammen mit der `format()`-Methode)
- Funktion `int(<str>)` zur Umwandlung einer Zeichenkette in eine ganze Zahl
- Funktion `float(<str>)` zur Umwandlung einer Zeichenkette in eine Gleitkommazahl
- Lesen von Dateien und Schreiben in Dateien mit der Funktion `open(...)` und den Methoden `fd.write(...)`, `fd.read(...)` und `fd.close()`.