Einführung

- Installation
- Informelle Grundlagen
- Datentypen und Operatoren

Installation

Im Folgenden wird der CPython-Interpreter auf dem verwendeten Computer installiert.

Um auf einem Computer mit Python programmieren zu können, muss zuerst ein Interpreter heruntergeladen und installiert werden.
Es gibt einige 'Varianten' der Programmiersprache Python, jeweils mit verschiedenen Einsatzzwecken und Eigenheiten: CircuitPython und MicroPython sind für Verwendung mit Microcontrollern optimiert und Jython erlaubt von Haus aus Interaktion mit Java-Bibliotheken.

Die bekannteste und beliebteste 'Variante' ist CPython, die in der Programmiersprache C geschriebene, offizielle Referenzimplementation. Die aktuellste Version hiervon wird auf der Website python.org für Windows und MacOS als Installer zum Download angeboten. Es wird empfohlen, im Installer keine Optionen zu verändern, und die Schnellinstallation auszuwählen. (3.14 ist die letzte Version, die über diesen Weg installiert werden kann: Das Werkzeug `pymanager` wird diese Aufgabe in Zukunft übernehmen.)

Unter Linux ist zwar oft möglich, Python über den Paketmanager der jeweiligen Distribution herunterzuladen, jedoch ist diese Version der Aktuellen fast immer um einige Jahre hinterher, und oft wird das Installieren von Paketen über den eingebauten Paketmanager `pip` unterbunden. Um Python unter Linux effizient nutzen zu können, muss es oftmals selbst kompiliert werden (Es wird angenommen, dass Linux-Benutzer hiermit schon Erfahrung haben).

Nach Abschluss der Installation (und eventuellem Systemneustart) müsste unter Windows der Eingabe des Befehls `py` in das Terminal (CMD) zu einer solchen (oder ähnlichen) Ausgabe führen:

```
Python 3.14.0 <Informationen zur verwendeten Toolchain>
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Unter Linux und MacOS wird hierfür stattdessen der Befehl `python3` verwendet. Im Verlauf dieses Buches wird vom Befehl `py` gesprochen, dieser muss auf diesen Plattformen mental 'übersetzt' werden, sofern man keinen Alias anlegen möchte.

Ebenso kann unter Windows die 'IDLE' als App eingetragen worden sein. Diese ist eine im Installationsumfang enthaltene, (vollständig in Python geschriebene) Entwicklungsumgebung für Python. Sie kann auf jeder Plattform ebenso mit dem Befehl `py -m idlelib` aufgerufen werden.

Aktuelle Informationen zur Installation und Verwendung von Python sind auf Englisch unter docs.python.org einzusehen.

Nun kann in einem Verzeichnis, auf das man einfach zugreifen kann (Desktop, Dokumente, etc.), eine Datei mit dem Namen `helloworld.py` erstellt werden (alternativ kann der Name frei gewählt werden). Diese Datei wird nun in einem Texteditor geöffnet (etwa Notepad, Visual Studio Code oder der soeben installierten IDLE) und der folgende Text hereingeschrieben:

```
print("Hello, world!")
```

Wird nun in der CMD in den entsprechenden Ordner navigiert, und der Befehl `py helloworld.py` eingegeben, so erscheint als Ausgabe des Programms der Text `Hello, world!`.

Informelle Grundlagen

Es folgt eine kurze Einleitung in einige Teile der Programmiersprache Python.

Ausgabe von Text

Wie wir im vorigen Kapitel gesehen haben, gibt der Programmcode

```
print("Text Hier Einfügen")
```

den eingefügten Text über die Standardausgabe (CMD, Linux-Terminal etc.) aus. Dies funktioniert, indem die Funktion `print` mit einem Text als einzigen Parameter aufgerufen wird. `print` tut folgendes:

- Jeder Parameter wird, sofern er keiner ist, in einen Text umgewandelt
- Jeder Parameter wird der Reihe nach über die Standardausgabe ausgegeben
- Zwichen den Parametern wird je ein Leerzeichen ausgegeben
- Am Schluss wird eine Neuzeile ausgegeben

Somit gibt das Programm

```
print("jedes", "Wort", "ist", "getrennt")
```

den Text `jedes Wort ist getrennt` aus.
Die CMD erwartet, dass von jedem Programm als letztes Zeichen die Neuzeile ausgegeben wird. Dieser Punkt spielt somit bei einem einzigen print-Aufruf keine Rolle, wird jedoch auffällig, sobald wir mehrere davon haben.

Anders als in Sprachen wie C und Java muss eine Anweisung nicht mit einem Semikolon (`;`) abgeschlossen werden, es reicht hierfür das Zeilenende aus. Somit kann man mehrere Zeilen Text der Reihe nach ausgeben, indem man mehrere `print`-Aufrufe in aufeinanderfolgende Zeilen schreibt:

```
print("Dieser Text kommt in Zeile 1")
print("Dieser Text kommt in Zeile 2")
```

Python ignoriert Leerzeilen bei der Ausführung. Ebenso wird, sobald es auf das `#`-Zeichen stößt, der Rest der Zeile (bis zum Zeilenende) als Kommentar gelesen und ignoriert. Somit verhalten sich folgende (durch `# ---` getrennte) Programme vollkommen gleich:

```
print("abc")
print("def")
# ---


print("abc")


print("def")


# ---
# diese Anweisung gibt "abc" aus
print("abc")
# diese Anweisung gibt "def" aus
print("def")
# ---
print("abc") # diese Anweisung gibt "abc" aus
print("def") # diese Anweisung gibt "def" aus
```

Variablen und Zuweisungen

Will ich beispielsweise dreimal den Text `Dieser Text wird dreimal ausgegeben` ausgeben, so kann ich dies durch das Programm

```
print("Dieser Text wird dreimal ausgegeben")
print("Dieser Text wird dreimal ausgegeben")
print("Dieser Text wird dreimal ausgegeben")
```

erreichen. Ist der Text jedoch deutlich länger (beispielsweise `Es ist Ziel dieses Programms, diesen Text siebenmal, also einmal für jede Zahl von 1 bis 7, auf die Standardausgabe auszugeben`), wird das Programm unnötig lang, und wünscht man, den Text zu ändern, so muss man das mehrmals tun. Python erlaubt es, dies zu vereinfachen: der Text kann einer Variable zugewiesen werden, welche daraufhin als Platzhalter für den Text verwendet werden kann:

```
my_text = "Es ist Ziel dieses Programms, diesen Text siebenmal, also einmal für jede Zahl von
1 bis 7, auf die Standardausgabe auszugeben"
print(my_text)
print(my_text)
```

```
print(my_text)
print(my_text)
print(my_text)
print(my_text)
print(my_text)
```

Es ist auch möglich, den Wert der Variable im Verlauf des Programmes zu ändern:

```
mein_text = "Es ist Ziel dieses Programms, diesen Text siebenmal, also einmal für jede Zahl
von 1 bis 7, auf die Standardausgabe auszugeben"
print(my_text)
print(my_text)
print(my_text)
print(my_text)
print(my_text)
print(my_text)
print(my_text)
mein_text = "Es ist Ziel dieses Programms, diesen Text dreimal, also einmal für jede Zahl von
1 bis 3, auf die Standardausgabe auszugeben"
print(my_text)
print(my_text)
print(my_text)
```

Ebenso können mehrere Variablen definiert werden.

```
even = "ist eine gerade Zahl"
uneven = "ist eine ungerade Zahl"
print("0", even)
print("1", uneven)
print("2", even)
print("3", uneven)
print("4", even)
print("5", uneven)
print("6", even)
```

Auch können Variablen bei Bedarf gelöscht werden

```
x = "Hallo!"
print(x)
del x
```

(Wird auf eine Variable zugegriffen, die nicht definiert oder schon gelöscht wurde, so stoppt Python die Ausführung des Programms und wirft einen Fehler)

Der Name einer Variable darf aus

- Groß- und
- Kleinbuchstaben
- Ziffern
- Unterstrichen (`_`)

bestehen, und nicht mit einer Ziffer anfangen. Ebenso gibt es bestimmte sog. "keywords", welche besondere Bedeutung haben und nicht zur Benennung von Variablen verwendet werden dürfen. Bisher ist uns nur `del` bekannt, es gibt jedoch sehr viele solcher Worte in Python (beispielsweise `in`, `if`, `while`, `match`, `class`, `def`, `from`). Es ist üblich, innerhalb eines Programmes dasselbe Format für Variablennamen zu verwenden: so kann ich eine Variable, die die Position des Spielers in einem Videospiel speichert, `spielerposition`, aber auch `spieler_position`, `SpielerPosition` oder `_Z7spieler8position` nennen. Python lässt diese Namen allesamt zu, jedoch führt Verwendung von mehr als einem Format oft zu Verwirrung und schlecht lesbarem Code. Es ist daher üblich, sich an den offiziellen Python-Styleguide PEP 8 zu halten, nach welchem Variablen Namen im `snake_case` (Worte klein geschrieben und mit Unterstrichen getrennt) tragen sollen. "Konstante Variablen", deren Wert im Laufe des Programms nicht geändert wird, werden im `SCREAMING_SNAKE_CASE` benannt, also großgeschrieben und mit Unterstrichen getrennt.

Es wird auch zumeist auf Englisch programmiert. Alle eingebauten Funktionen und Klassen sind beispielsweise auf Englisch benannt und dokumentiert. Auch ist davon abzuraten, nicht-ASCII-Buchstaben und Zeichen (wie `äöü`, aber auch `î` und `á`) in Variablennamen zu verwenden. In allem folgenden Programmcode werden Variablen somit auf Englisch benannt (Kommentare bleiben jedoch deutsch).

Benutzereingaben

Für etwas Interaktvität kann es nützlich sein, von dem Benutzer des Programms eine Eingabe zo fordern. Hierfür wird die Python-Funktion `input` verwendet. Das einfachste hiermit realisierbare Programm lautet

```
print(input())
```

Es wartet, bis der Benutzer im Terminalfenster eine Zeile Text eingibt und gibt diese Zeile sofort wieder aus. Natürlich kann dieser Wert auch in einer Variable gespeichert werden.

```
print("Wie heißt du?")
name = input()
print("Hallo,", name)
```

```
print("Ich weiß jetzt, dass du", name, "heißt!")
```

Optional kann dieser Funktion auch ein Parameter übergeben werden. Wird zum Beispiel `input("Wie heißt du?")` aufgerufen, so wird die Zeile mit diesem Text begonnen, wobei der Nutzer unmittelbar darauf seinen Namen eingeben kann. Das Ergebnis liest sich dann als `Wie heißt du?Max Mustermann` (wobei nur der Teil nach dem Fragezeichen eingegeben werden musste). Generell wird empfohlen, an das Ende dieses Textes ein Leerzeichen zu hängen. Die Benutzererfahrung kann dazu deutlich verbessert werden, wenn alle 'leeren Aufrufe' von `input()` durch eine einfache Eingabeaufforderung wie `input("> ")` ersetzt werden.

Datentypen und Operatoren

Im Folgenden wird in die Datentypen und arithmetischen Operatoren der Programmiersprache Python eingeführt.

Zahlen

Bisher haben wir nur mit sogenannten Strings gearbeitet, also Zeichenketten. Python nennt diesen Datentyp `str`. Um jedoch komplexere Programme zu schreiben, muss man mit Zahlen rechnen können. Python stellt uns dazu zwei andere Datentypen zur Verfügung.

Ganze Zahlen

Eine einfache Zahl als Folge der Ziffern `0` bis `9` im Programm liest Python diese als eine ganze Zahl, auf Englisch 'integer' und in Python `int`.

```
some_int = 0
some_other_int = 100000000000000000000000000000
yet_another_int = 3
```

Zu beachten ist, dass anders als die Programmiersprache C (welches wohl die erste war, die 'ganze Zahlen' `int` nannte),

- Python keine Zahl-Datentypen mit verschiedenen 'Größen' hat
- `int` Zahlen in einem unglaublich großen Bereich darstellen kann (mehrere Millionen Ziffern im Dezimalsystem).

Der Unterstrich `_` kann in einer Zahl in jeder Stelle (außer ganz am Anfang) eingefügt werden, und wird ignoriert. Beginnt eine Zahl mit `0x`, wird sie als Hexadezimalzahl gelesen (Ziffern `0`-`9` und `a`-`f` bzw. `A`-`F`). Beginnt sie mit `0o` oder `0b`, ist sie zur Basis 8 (oktal) oder 2 (binär) geschrieben.

```
some_int = 100
the_same_int = 1_0_0
the_same_again = 0x64
```

Gleitkommazahlen

Es ist dem Computer nicht möglich, Reelle Zahlen zu speichern. Rationale Zahlen sind als Paar zweier `int`-Werte möglich, jedoch ist dies langsam und speicherintensiv (und noch langsamer, wenn man versucht, jeden Bruch auf den keinsten Nenner zu bringen). Eine Alternative dazu sind fixed-point-Zahlen, wobei mit ganzen Zahlen gerechnet wird, die jeweils den Zähler eines Bruches mit festem Nenner beschreiben. Eine nicht ganz offensichtliche (uhd häufig nicht ideale, aber gut funktionierende) Methode ist die Verwendung von Gleitkommazahlen, welche in Python durch den Datentyp `float` beschrieben werden. Diese haben intern die Form `s * m * 2^e`, wobei

- `s`, das Vorzeichen, -1 oder 1 ist
- `m`, die Mantissa, als Binärzahl in der Form von `1.xyz` gespeichert wird (mit 52 Bits als Nachkommastellen)
- `e`, der Exponent, einen Wert zwischen -1022 und 1023 hat und als Binärzahl mit Offset 1023 gespeichert wird

Fast alle Operationen mit Gleitkommazahlen sind ungenau, insbesondere das Umwandeln zwischen ihnen und Zahlen im Dezimalsystem. Ein bekanntes Beispiel dafür ist auf 0.30000000000000004.com zu finden.