COMPUTERSCHULUNGEN

Rat und Schulung

Python - Seminar

Einblick in die Programmiersprache

Dr.sc.nat. Michael J.M. Wagner, AS Computer*

Revision 300

*michael@wagnertech.de

Inhaltsverzeichnis

1 Einführung in Python[1]

1.1 Grundlagen

Python ist ein Interpreter-Sprache mit vielen Vorteilen:

- Eine einfache, eindeutige Syntax
- Klare Strukturen: Die Anordnung der Programmzeilen ergibt gleichzeitig die logische Struktur des Programms.
- Unabhängigkeit vom Betriebssystem

Entwicklungsgeschichte:[2]

- 1990: Erste Anfänge durch Guido van Rossum (Amsterdam)
- 1994: Python 1.0. Anfänglich mit Konzepten der funktionalen Programmierung
- 2000: Python 2.0. Garbage Collection, Unicode-Zeichensatz
- 2008: Python 3.0. Entfernen von Redundanzen bei Befehlssätzen und veralteten Konstrukten

  ABER: Verlust der Abwärtskompatibilität. Daher wird die Version 2.7 weiter unterstützt.

Da die graphischen Oberflächen der meisten Linux-Distributionen Python verwenden, ist der Python-Interpreter dort stets installiert. In diesem Kurs wird die Entwicklungumgebung *Eclipse* verwendet, die sich auch für viele andere Programmiersprachen eignet.

Aufgabe:

- Prüfen Sie die installierten Python-Versionen, indem Sie in der Shell folgende Kommandos absetzen:

```
$ python -V
$ python3 -V
```

- Starten Sie den Python-Interpreter und führen Sie „Übung u_grundrechenarten"[3]aus.

Wie in Skriptsprachen üblich, ist das „Hauptprogramm" dadurch gekennzeichnet, dass die Befehlszeilen direkt in der Datei, ohne weitere Angaben einer Funktion o.ä. stehen. Ein „Hallo Welt"-Programm besteht also nur aus der Zeile:

```
print ("Hallo Welt")
```

Zur Ausführung von Python-Skripten gibt es folgende Möglichkeiten:

- Ausführung in der Entwicklungsumgebung: *Ausführen als* → *Python Skript*

[1] Theis: Kap. 1.
[2] https://de.wikipedia.org/wiki/Python_(Programmiersprache) (25.7.2017)
[3] Theis: S. 25f.

- Ausführung mit dem Aufruf des Interpreters:

  ```
  python3 HalloWelt.py
  ```

- Unter POSIX-Systemen (Linux, ...) besteht als weitere Möglichkeit den Interpreter im Skript selbst anzugeben. Die erste Zeile des Skripts lautet dann:

  ```
  #!/usr/bin/python3
  ```

  Das Skript muss dann ausführbar gemacht werden (`$ chmod 755 HalloWelt.py`). Danach kann das Skript direkt aufgerufen werden:(`$ ./HalloWelt.py`).

Die offizielle Python-Dokumentation findet sich unter `http://docs.python.org`.

Aufgabe:

- Legen Sie in der Entwicklungsumgebung ein neues Projekt an und fügen Sie eine Datei `HalloWelt.py` hinzu.
- Geben Sie in dieser Datei den Code für die Ausgabe ein.
- Führen Sie die Datei auf die drei oben beschriebenen Weisen aus.

  Stellen Sie in Eclipse `/usr/bin/python3` als Standardinterpreter ein.

1.2 Python Sprachelemente

Kommtare

```
# Mein erstes Programm
print("Hallo Welt")   # Eine Ausgabe
"""Kommentar in
mehreren Zeilen"""
```

Variablen

Variablen sind in Pyton getypt. Der Definition einer Variablen erfolgt durch Zuweisung. Es gibt keine implizite Typumwandlung einer Variable: → [T:3.2.1, `zuweisung.py`]

Als Basistypen stehen *integer*, *float*, *string* und *boolean* zur Verfügung.

Die Typumwandlung erfolgt durch entsprechende Funktionen (z.B. `int()`):
→ [T:3.2.3, `eingabe_zahl.py`]

Soll der Typ einer Variable zur Laufzeit ermittelt oder überprüft werden, kann dies mit der Funktion `type()` erfolgen: → [T:4.1.3, `zahl_type.py`]

Verzweigungen

Vergleichsoperatoren, Einfache Verzweigungen: → [T:3.3.2, `verzweigung_einfach.py`]

Zur Beachtung: Einrücken gehört zur Syntax!

Abweichend von den meisten anderen Programmiersprachen kann Python auch ein „zwischen" formulieren: `if x < y < z:`[4]

Mehrfache Verzweigung: → [T:3.3.4, `verzweigung_mehrfach.py`]

Logische Operatoren: `and or not`
→ [T:3.3.5, `operator_logisch.py`]

Schleifen

Iteration über eine Anzahl von Elementen: `for i in 2, 7.5, -22:`[5]
`break` springt aus der Schleife.

Zählschleife: `for i in range(3,11,2):`
Werte für `i: 3,5,7,9`
1. Parameter: Startindex (optional, default: 0)
2. Parameter: Ende-Index. Dieser Index wird nicht mehr bearbeitet!
3. Parameter: Schrittgröße (optional, default: 1)

```
for i in range(3):
    print("Zahl:", i)
```

Ergebnis:

```
Zahl: 0
Zahl: 1
Zahl: 2
```

Kopfgesteuerte Schleife: `while summe < 30:` → [T:3.4.7, `schleife_while.py`]

Eine fußgesteuerte Schleife existiert in Python nicht.

> Aufgabe:
>
> Schreiben Sie ein Programm (Datei `u_while.py`), das den Anwender wiederholt dazu auffordert, einen Wert in Inch einzugeben. Der eingegebene Wert soll anschließend in Zentimeter umgerechnet und ausgegeben werden. Das Programm soll nach der Eingabe des Werts 0 beendet werden.
>
> 1 inch = 2,54 cm

[4] Theis: S. 48.
[5] Theis: S. 51.

Funktionen

Einfache Funktionen: [T:3.7.1, `funktion_einfach.py`]

Funktionen mit Paramtern: `def berechnung(x,y,z):`[6]

Funktionen mit Rückgabewert: [T:3.7.4, `rueckgabewert.py`]

Python kennt bei einfachen Variablen nur den *call by value*. Bei komplexen Daten wird in der Variable die Referenz verwaltet. *Immutable objects* (z.B. Zeichenketten) werden aber bei einer Änderung kopiert, so dass sich diese wie einfache Variablen verhalten.

Sollen aus einer Funktion mehrere Variablen an den Aufrufer zurück gegeben werden, können diese kommaseperiert angegeben werden:

```
def f():
   a = 3
   b = 2.5
   c = "Hallo"
   # f gibt 3 Werte zur"uck
   return a,b,c

# Aufruf
x,y,z = f()
```

Hier noch ein paar Varianten der Variablenübergabe:

- Default-Parameter

  ```
  def printer(zeile = ""):
     print(zeile)

  printer("Hallo")        # Schreibt Hallo
  printer()               # Schreibt Leerzeile
  ```

- Benannte Parameter

  ```
  def printer(a, b):
     print(a+" "+b)

  printer("Hallo", "Welt")        # Schreibt Hallo Welt
  printer(b="Welt", a="Hallo")    # Schreibt Hallo Welt
  ```

Pakete und Module

Befindet sich die zu rufende Funktion in einem anderen Modul (= andere Datei), muss diese zuvor importiert werden. Beim Aufruf muss der Name des Moduls dem Funktionsnamen voran gestellt werden:

```
import mein_modul  # falls die Datei mein_modul.py heisst

# Funktionsaufruf, falls in mein_modul.py eine Funktion f() definiert ist:
mein_modul.f()
```

[6] Theis: S. 75.

Soll die Funktion `f` ohne explizite Modulangabe aufrufbar sein, so kann diese oder alle Funktionen des Moduls in den aktuellen Kontext eingebunden werden:

```
from mein_modul import f
# oder
from mein_modul import *
```

Enthält ein Modul direkt ausführbaren Code, so wird dieser beim Import-Statement ausgeführt. Soll dies verhindert werden und der Code nur dann ausgeführt werden, wenn das Modul als Hauptprogramm aufgerufen wird, kann dies mit folgender Konstruktion erreicht werden:

```
def f():
    ...

if __name__ == "__main__":
    # Dieser Code wird nicht beim Import ausgef"uhrt
    print ("Modul wurde als Hauptprogramm gerufen")
    f()
```

Module wiederum lassen sich in Paketen (=Unterverzeichnisse) zusammenfassen. Liegt die Datei `mein_modul.py` im Unterverzeichnis `pack`, so gibt es folgende Varianten für den Import:

- Import des Moduls

  ```
  from pack import mein_modul
  mein_modul.f()
  ```

- Import der Funktion `f` aus `mein_modul`:

  ```
  from pack.mein_modul import f # oder *
  f()
  ```

2 Datei-Operationen

Das Öffnen einer Datei erfolgt mit:

```
with open(<Dateiname>[, <modus>]) as datei_objekt:
```

Anmerkung: Allokiert ein Objekt Resourcen und implementiert das Objekt eine `__exit__`-Methode zur Freigabe der Resourcen, kann durch Verwendung des `with` gesichert werden, dass mit Verlassen des Blocks auf jeden Fall die Resourcen freigegeben werden. Im Fall des `open` ist damit kein `close` mehr nötig.

`<modus>`: `r` (default), `w`, `a`[7]

Für die zeilenweise Verarbeitung einer Datei empfiehlt sich folgende Formulierung:

```
with open("readfile.txt") as d:

    # Ausgabe und Addition aller Elemente
    sum = 0
    for line in d:
        print(line, end="")
```

[7] Theis: S. 259.

```
        sum += float(line)

# Ausgabe der Summe
print("Summe:", sum)
```

Das Schreiben einer Zeile erfolgt mit:

```
d.write( string_var )
```

Alternativ kann auch die Funktion `print` verwendet werden, wenn der benannte Parameter `file` versorgt wird. `print` hat den Vorteil, dass

- mehere Parameter übergeben werden können,
- mit dem benannten Parameter `sep` bestimmt werden kann, welches Zeichen zwischen den übergebenen (normalen) Parametern eingefügt werden soll,
- am Zeilenende standardmäßig ein Zeilenumbruch angefügt wird.

```
with open("writefile.txt", "w") as d:
   print(file=d,"Zeile zum Schreiben")
```

Aufgabe:

Mit den nächsten Übungen soll Schritt für Schritt eine Bücherverwaltung für eine Bücherei erstellt werden. Führen Sie folgende Schritte aus:

- Legen Sie ein neues Projekt *Buecherei* an.
- Legen Sie die Datei `Medienverwaltung.py` an.
- Schreiben Sie eine Funktion `addMedium`, die sechs Parameter nimmt:

  `addMedium(signatur, autor, titel, typ, seitenzahl, spieldauer)`
- `addMedium` soll die Daten an die Datei `medien.csv` kommasepariert anhängen.
- Rufen Sie die Prozedur aus dem Hauptprogramm (mit konstanten Werten) auf.

3 Zeichenketten[8]

In Python gibt es mächtige Werkzeuge zur Bearbeitung von Zeichenketten.

Zur Definition von Zeichenketten können verschiedene Hochkommata verwendet werden: [T:4.2.1, `text_eigenschaft.py`

Die Operatoren `+`, `*`, `in`: [T:4.2.2, `text_operator.py`]

Indizierung der Zeichen innerhalb einer Zeichenkette (Abb. 1)

Bildung von Slices: [T:4.2.3, `text_operation.py`]

[8] Theis: Kap. 4.2
[9] Theis: p. 100.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Element	R	o	b	i	n	s	o	n		C	r	u	s	o	e
Negativer Index	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Abbildung 1: Sequenz mit Index[9]

Es gibt in Python viele Funktionen zur Bearbeitung von Zeichenketten.[10] Die folgenden seien herausgehoben:

- `find`: [T:4.2.4, `text_suchen.py`]
- `replace(old, new[, count])`: Liefert eine Kopie der Zeichenkette, bei der alle Zeichenketten `old` durch `new` ersetzt sind. Wenn das optionale Argument `count` gesetzt ist, werden nur die ersten `count` Stellen ersetzt.
- `startswith(prefix)`: Liefert `True`, wenn die Zeichenkette mit `prefix` beginnt, sonst `False`.
- `split(sep=None, maxsplit=-1)`: Liefert eine Liste von Wörtern aus der Zeichenkette mit `sep` als Trenner. Wenn `maxsplit` gegeben ist, wird die Zeichenkette in maximal `maxsplit` Bestendteile zerlegt.
- `rstrip, lstrip, strip`: Entfernt Leerzeichen

Aufgabe: Schreiben Sie im Modul `Medienverwaltung.py` eine Funktion `isSignatureInFile(signatur)`, die

- die Datei `medien.csv` Zeile für Zeile ausliest,
- jede Zeile prüft, ob sie mit der übergebenen Signatur beginnt,
- als Ergebnis einen bool'schen Wert zurückgibt: `True`, falls die Signatur bereits in der Datei vorhanden ist, sonst `False`.
- Falls die Eingabedatei nicht vorhanden ist, soll dies auch zu „Signatur nicht vorhanden" (`False`) führen.
- Überlegen Sie sich für das Hauptprogramm eine Benutzerführung, mit der Sie zwischen den angebotenen Funktionen wählen können.

[10] https://docs.python.org/3.5/library/stdtypes.html#string-methods (14.3.2018)

4 Fehlerbehandlung

Zur Fehlerbehandlung gibt es verschiedene Ansätze:

- Rückgabewerte vom Typ `int`
- Rückgabewerte eines speziellen Fehlertyps
- Exceptions
- Eigene Routine zur Ermittlung des Fehlerstatus

Vorteil der Rückgabewerte:

- Einfachere lokale Fehlerbehandlung

Vorteil der Exceptions:

- Fehlerbehandlung stört nicht den logischen Programmfluss.
- Fehlerbehandlung wird nicht vergessen.

Empfehlung:

- Zu erwartende (oft fachliche) Fehler werden auf Rückgabewerte abgebildet.
- Unerwartete Fehler (oft technische Fehler, Logikfehler) werden über Exceptions behandelt.

Beim Werfen und Fangen von Ausnahmen wird der Typ dem Schlüsselwort `raise` hintenangestellt:

```
raise RuntimeError("Da gibt es ein Problem")
```

In Python gibt es zahlreiche vordefinierte Ausnahmen. Der Obertyp aller Ausnahmen ist `Exception`. Hier weitere Ausnahmetypen:[11]

- `EnvironmentError`: Fehler in der Umgebung, in der das Programm gestartet wurde, z.B. fehlende Dateien
- `ValueError`: Fehler, wenn auf einen unerwarteten Attributwert gestoßen wird
- `RuntimeError`: Sonstige zur Laufzeit auftretende Fehler

Beim Fangen der Ausnahmen beginnt man mit den speziellen Typen und endet mit dem Obertyp `Exception`:

```
try:
        ...
except ValueError:
    print("Fehler: keine Zahl")
except ZeroDivisionError:
    print("Fehler: Zahl 0 eingegeben")
except RuntimeError as e:
    print("Fehler:", e)
# Fangen aller sonstigen Ausnahmen
except Exception as e:
    print ("Ein Fehler ist aufgetreten:", e)
```

[11] https://docs.python.org/2/library/exceptions.html (1.8.2017)

Aufgabe:

Ergänzen Sie die Medienverwaltung:

- In der Funktion `addMedium` um den Aufruf von `isSignatureInFile`. Falls die Signatur schon vorhanden ist, geben Sie einen entsprechenden Integer-Fehlercode an das Hauptprogramm zurück. Definieren Sie dazu entsprechende Konstanten (was leider vom Python nicht wirklich unterstützt wird).
- Werten Sie im Hauptprogramm die Fehlercodes aus.

5 Objektorientierung

Der rein prozedurale Ansatz kommt in großen Projekten an seine Grenzen. Ein Problem ist die Verantwortlichkeit für den Inhalt von Datenstrukturen.

Die Objektorientierung galt in den 90er-Jahren als Allheilmittel. Daher wurden Klassendefinitionen als Behälter für Prozeduren missbraucht. Es zeigte sich aber bald, dass auch Klassennamen zu Namenskonflikten führen können. In Python wird daher der Namensraum in Module (=Dateien) zerlegt, die explizit importiert werden müssen.

Folgene Erfahrungen in der Softwareentwicklung haben zur Idee der Objektorientierung geführt:

- Strukturen sind eine sehr nützliche Sache, um Daten, die logisch zusammen gehören, zusammen zu verwalten.
- Wird eine Struktur im Speicher angelegt, wurde es in großen Programmwerken schnell unübersichtlich, wer diese Struktur für welchen Zweck gebraucht und wer Änderungen daran vornimmt.
- Unklar war oft, ab welchem Zeitpunkt welche Bestandteile einen gültigen Wert besitzen.

Vor diesem Hintergrund kam die Idee auf, die Zugriffe auf Strukturen (lesend, wie schreibend) zu kontrollieren. Der allgemeine Zugriff auf die Datenstruktur wurde also verboten, stattdessen wurden Funktionen geschaffen, über die auf die Daten zugegriffen werden konnte. Eine Datenstruktur mit den dazugeförigen Zugriffsfunktionen nennt sich *Klasse*.

Die Instanzierung einer Klasse bedeutet, einer Struktur einen konkreten Speicherbereich zuzuordnen. Nur eine instanzierte Struktur (=Objektinstanz) kann auch verwendet werden.

Bei einer Klasse sind im Gegensatz zu einer Struktur die interen Datenelemente von außen nicht zugreifbar. Alle Zugriffe erfolgen prozedural über öffentliche *Methoden*. So weit die Theorie: Python ist da nicht so streng.

5.1 Klassen

In Python werden Objektinstanzen ausschließlich im Freispeicher angelegt und vom *garbage collector* wieder frei gegeben. Darüber hinaus kann eine Instanz durch das explizite Aufrufen von `del` freigegeben werden. Eine Variable im Programm stellt daher lediglich einen Verweis in den Freispeicher dar. Über eine Zuweisung wird nur der Verweis kopiert, nicht die Instanz selbst. Um die Instanz selbst zu kopieren stellt Python die Funktion `copy.deepcopy(<obj>)` zur Verfügung.

Klassendefinition, Erzeugen und Verwenden von Instanzen: [`Fahrzeug.py`]

In streng objektorientierten Sprachen darf von außen nicht auf Elemente eines Objekts (Eigenschaften) zugegriffen werden. Diese sind „privat". Datenzugriff erfolgt ausschließlich über *getter*. In Python ist aber der direkte Zugriff auf Eigenschaften üblich.

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    geschwindigkeit = 0               # Eigenschaft
    def getGeschwindigkeit(self):     # getter
       return self.geschwindigkeit
```

Konstruktor (aus `Fahrzeug.py`):

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __init__(self, bez, ge):       # Konstruktormethode
        self.bezeichnung = bez
        self.geschwindigkeit = ge

# Objekte der Klasse Fahrzeug erzeugen
opel = Fahrzeug("Opel Admiral", 40)
volvo = Fahrzeug("Volvo Amazon", 45)
```

Anmerkung: Im Unterschied zu Programmiersprachen wie C++, C# oder Java können in Python Funktionen/Methoden nicht überladen werden. Es gibt also stets genau einen Konstruktor.

Neben Konstruktor und Destruktor gibt es noch weitere `__<NAME>__`-Methoden. Besonders nützlich sind:

- `__str__`: Die Umwandlung in eine Zeichenkette
- `__repr__`: Soll eine für die Objetinstanz eindeutige Zeichenkette liefern. `__repr__` wird bei der Ausgabe der in Listen enthaltenen Objekten verwendet.
- `__eq__`: Definition, wann zwei Instanzen gleich sind, definiert den ==-Operator

```
# Definition der Klasse Fahrzeug
class Fahrzeug:
    def __str__(self):                 # Ausgabemethode
        return self.bezeichnung + " " \
           + str(self.geschwindigkeit) + " km/h"

# Objekte der Klasse Fahrzeug erzeugen
opel = Fahrzeug("Opel Admiral", 40)
volvo = Fahrzeug("Volvo Amazon", 45)

# Objekte ausgeben
print(opel)
```

```
print(volvo)
```

Anmerkung: Eine Methode, die auf einer Instanz arbeitet, hat `self` als ersten formalen Parameter. Eine Klassenmethode ohne `self` könnte *statisch* genannt werden.

5.2 Vererbung

Vererbung ist ein nützliches Merkmal der Objektorientierung um gemeinsame Eigenschaften aus Klassen herauszufaktorieren (s. Abb. 2).

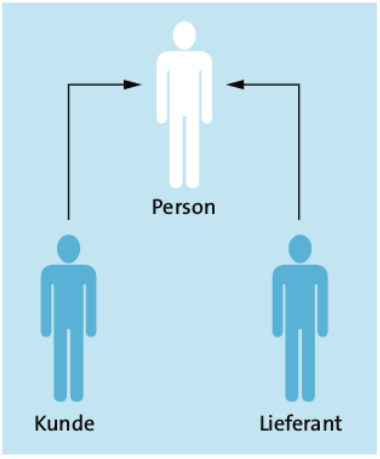

Abbildung 2: Vererbungsbeziehung[12]

Syntax der Vererbung: [T:6.7, `oop_vererbung.py`]

Die Methode `__str__` ist sowohl in `Fahrzeug` als auch in `PKW` definiert. Dies nennt man „eine Methode überschreiben". Der Python-Interpreter entscheidet anhand der tatsächlichen Klasse, auf die die Referenz zeigt, welche Implementierung aufzurufen ist. Dieses Verhalten nennt man „polymorphes Überschreiben".

Wie auch in C++ erlaubt Python die Mehrfachvererbung: [T:6.8, `oop_mehrfach.py`]

Aufgabe:

Ergänzen Sie die Bibliotheksanwendung:

- Ergänzen Sie das Modul `MediumModul.py`. Leiten Sie von der Klasse `MediumBase` die Klassen `Buch` und `CD` ab.
- Verteilen Sie die Attribute sinnvoll auf die Klassen.
- Ergänzen Sie jede Klasse um eine Methode `__str__`, die den Datensatz als Zeichenkette in dem Format zurückgibt, wie er in der Eingabedatei erwartet wird.
- Testen Sie den Code durch direkte Verwendung im Hauptprogramm.
- Ergänzen Sie die Medienverwaltung um eine Funktion

[12] Jürgen Wolf, Grundkurs C++: S. 346.

addMediumHira(mediumBase),

die die neue __str__-Methode zum Schreiben in die Datei verwendet. Zur Duplikatsprüfung kann die bestehende isSignatureInFile verwendet werden.

- Verwenden Sie die Methode im Hauptprogramm, indem Sie einmal ein Buch, einmal eine CD übergeben.
- Sollen, wie in kompilierten Sprachen üblich, Methodennamen überschrieben werden, kann dies in Python dadurch erreicht werden, dass die übergebenen Parameter mit isinstance auf ihre Zugehörigkeit zu einer Vererbungshierarchie überprüft werden.

  Ergänzen Sie die Funktion addMedium in der Weise, dass

  – nur der erste Parameter Pflicht ist,

  – wenn sich der erste Parameter von MediumBase ableitet, in die Funktion addMediumHira verzweigt wird.

6 Weitere Typen

6.1 Enumeration

Aufzählungen können in Python von der Basisklasse enum.IntEnum abgeleitet werden. Diese Klasse ermöglicht es, sowohl eine Darstellung als Integer zu erhalten, was zum Abspeichern in Datenbanken oder zum Transport über Schnittstellen praktisch ist, als auch eine Rückumwandlung vom Integerwert zur textuellen Darstellung zu bieten.
→ [T:6.9, enumeration.py]

Aufgabe:

- Definieren Sie im Hauptprogramm einen enum für die Rückgabewerte.
- Weisen Sie die zurückgegebenen Ganzzahlwerte einer enum-Variable zu:
  rc_enum = RC_Typ(rc_int)
- Geben Sie den Rückgabewert „sprechend" aus.

6.2 Listen

Listen sind Folgen von Objekten, die von verschiedenen Typen sein können. Da Listen geordnet sind, kann mittels Index auf die einzelnen Elemente zugegriffen werden.
→ [T:4.3.1, liste_eigenschaft.py]

Listen können mit + konkatiniert und mit * vervielfältigt werden.
→ [T:4.3.2, liste_operator.py]

Im Gegensatz zu Zeichenketten sind Listen veränderbar: [T:4.3.3, `liste_element.py`]

Weitere nützliche Listenoperationen: [T:4.3.3, `liste_aendern.py`]

Python kennt auch eine nicht veränderbare Variante der Liste, das Tupel.[13] Tupel werden in dieser Form definiert:

```
t1 = 13, "Hallo", 15.0
t2 = ( 13, "Hallo", 15.0 )
```

Tupel können auch über die Parameterübergabe erzeugt werden:

```
def printer(*tup):
   print(tup)

printer("Hallo", "Welt")      # Schreibt ("Hallo","Welt")
```

Aufgabe:

- Ergänzen Sie das Modul `Medienverwaltung` um die Funktion `getMediaList()`, die eine Liste von Medien zurückgibt.
- Geben Sie diese Liste im Hauptprogramm aus.

6.3 Verzeichnisse und Mengen

Verzeichnisse

Schlüssel-Wert-Paare werden als Verzeichnisse, Wörterbücher oder *dictionaries* bezeichnet.
→ [T:4.5.1, `dictionary_eigenschaft.py`]

Operationen auf Verzeichnisse: [T:4.5.2, `dictionary_funktion.py`]

Über Verzeichnisse lässt sich nicht direkt iterieren. Dazu müssen *views* erzeugt werden. Für ein Verzeichnis lassen sich drei *views* erzeugen: Schlüssel `keys()`, Werte `values()`, Paare `items()`.

Anmerkung: Wenn das Verzeichnis doppelte Werte enthält, sind diese auch in `values()` doppelt enthalten.
→ [T:4.5.3, `dictionary_view.py`]

Verzeichnisse können auch über die Parameterübergabe erzeugt werden:

```
def printer(**verz):
   print(verz)

printer(a="Hallo", b="Welt")      # Schreibt {a:"Hallo",b:"Welt"}
```

[13] Theis: Kap. 4.4

Mengen

> Mengen (englisch: *sets*) unterscheiden sich von Listen und Tupeln dadurch, dass jedes Element nur einmal existiert. Außerdem sind Mengen ungeordnet, daher ist auch die Reihenfolge bei der Ausgabe eines gesamten Sets nicht festgelegt.[14]

Syntax: `s = {'a', 12, 4.5}`

Listen lassen sich mit der Funktion `set()` in Mengen umwandeln. Duplikate werden dabei entfernt.

Aufgabe:

Ergänzen Sie die Bibliotheksanwendung. Um die Datei nicht jedes mal auf's Neue lesen zu müssen, sollen die Daten in einem Verzeichnis abgelegt werden. Dazu bekommt die Medienverwaltung jetzt interne Daten, wird vom Funktionsmodul zur Klasse.

- Legen Sie die Klasse `MedienverwaltungClass` mit einem internen Verzeichnis an.
- Schreiben Sie eine Methode `load()`, die die Datei einliest und das Verzeichnis füllt. Als Schlüssel soll dabei die Signatur dienen, als Wert eine Buch- oder CD-Instanz.

  Nehmen Sie die Implementierung von `isSignatureInFile(signatur)` als Vorlage.
- Der Konstruktor der Klasse ruft die `load()`-Methode auf.
- Schreiben Sie eine Methode `checkDuplicate(signatur)`, die prüft, ob eine bestimmte Signatur im Verzeichnis vorhanden ist.
- Schreiben Sie eine Methode `addMedium(medium)`, die `checkDuplicate` aufruft und das Medium an die Datei anhängt, danach das Verzeichnis mit `load` aktualisiert.
- Verwenden Sie den neuen Code im Hauptprogramm.

7 Python Standard-Bibliothek

Die Implementierung der verschiedenen Typen ist in Python in der Standard-Bibliothek organisiert. Selbst für die eingebauten Datentypen definiert der Sprachkern lediglich die Schnittstelle. Die Standard-Bibliothek ist sehr umfangreich.

Inhalt der Standard-Bibliothek[15]

Auf Linux-Systemen wird die Bibliothek üblicherweise in mehrere Pakete zerlegt, die ggf. nachinstalliert werden müssen.

Im Folgenden werden nun einige Elemente der Bibliothek näher vorgestellt.

[14] Theis: S. 126

[15] https://docs.python.org/3.3/library/ (25.11.2019)

7.1 Datum und Zeit

Aktueller Zeitpunkt mit `time` und `localtime`: [T:7.1.1, `zeit_localtime.py`]

Formatieren der Zeitpunkte mit `strftime`: [T:7.1.2, `zeit_strftime.py`]

Zeitpunkt erzeugen mit `mktime`: [T:7.1.3, `zeit_erzeugen.py`]

Mit Zeitangaben rechnen: [T:7.1.4, `zeit_rechnen.py`]

> Aufgabe:
>
> Bringen Sie das Beispiel `spiel_zeit.py` zum Laufen.

7.2 Container „double-ended queue"

Operationen: [T:7.2.1, `container_deque1.py`]

Veränderungen: [T:7.2.2, `container_deque2.py`]

> Aufgabe:
>
> Ergänzen Sie die Klasse `MedienverwaltungClass`:
>
> - um die Eigenschaft `sortedMedia`, eine *double ended queue*
> - Im Konstruktor soll Medium für Medium nach der Signatur sortiert eingegefügt werden.
> - um die Methode `printMediaSorted(reverse=False)`, die die Medien sortiert ausgibt. Falls der `reverse`-Parameter auf `True` steht, sollen die Medien in umgekehrter Reihenfolge ausgegeben werden.

7.3 Brüche

→ [T:4.1.9, `zahl_bruch.py, zahl_bruch_naehern.py`]

7.4 Reguläre Ausdrücke

Suchen von Teiltexten: [T:7.4.1, `regexp_suchen.py`]

Ersetzen von Teiltexten: [T:7.4.2, `regexp_ersetzen.py`]

> Aufgabe:
>
> - Ändern Sie `isSignatureInFile` so ab, dass die Prüfung der Signatur über einen regulären Ausdruck erfolgt.

- Ändern Sie `getMediaList` so ab, dass die Zerlegung der Zeile aus der Datei über einen regulären Ausdruck erfolgt.

7.5 Multithreading

Threading

Gerade bei GUI-Anwendungen ist die Abspaltung der Bearbeitung in einen eigenen Thread wichtig, da die GUI sonst bei länger laufenden Aktionen „einfriert".

Starten eines Threads: [T:7.3.3, `thread_ident.py`]

Anmerkung: Ausnahmen (*exceptions*) werden nicht über Threads hinweg übertragen. Jeder Thread sollte daher seine eigene Ausnahmebehandlung haben.

Multithreading erfordert den Schutz sensibler Ressourcen. Dies können globale Variablen innerhalb des Prozesses oder Systemressourcen sein. Für erstere wird ein `Lock` verwendet, für zweitere ein `Mutex`:

```
lock = threading.Lock() # oder Mutex

with lock:
   # access Ressource
```

Oft ist es nötig, Threads zu synchronisieren. Dabei warten die Threads an einer `Condition` und laufen nach einer Benachtichtigung (*notification*) weiter:

```
condition = threading.Condition()

with condition:
   condition.wait()               # wartet auf eine Benachritigung
   condition.wait_for(predicate) # wartet auf eine Benachritigung + predicate
   notify()                       # benachrichtigt einen der wartenden Threads
   notify_all()                   # benachrichtigt alle der wartenden Threads
```

Koroutinen

Ein modernerer Ansatz ist die Verwendung von Koroutinen. Koroutinen sind unterbrechbare Routinen. Python läuft grundsätzlichlich *single theaded*. Wird aber eine Koroutine vertagt, kann der Interpreter bei einer anderen Koroutine mit der Ausführung fortfahren. Diese Eigenschaft des Interpreters nennt sich *global interpreter lock* (GIL).

Hier ein Beispiel für asynchrone Koroutinen:

```
import asyncio

async def sleep_6():
  await asyncio.sleep(5)
  print('5 done')
  await asyncio.sleep(3)
```

```
  print('3 done')

async def sleep_7():
  await asyncio.sleep(7)
  print('7 done')

async def main():
  await asyncio.gather(sleep_6(), sleep_7())

asyncio.run(main)
```

Aufgabe:

Schreiben Sie das Spiel „Concentration":

- Starten Sie N Spieler, jeder Spieler kennt seine Nummer.
- Jeder Spieler warten auf eine Benachritigung, läuft aber nur los, wenn die globale Variable `next_player` mit der eigenen Nummer übereinstimmt.
- Ein Spieler würfelt den nächsten Spieler, setzt die Variable `next_player` auf diesen Wert und benachtichtigt alle anderen Spieler.
- Macht ein Spieler einen Fehler (ruft sich selbst auf oder ruft einen Spieler, der bereits ausgeschieden ist), scheidet er aus. Daher ist eine Liste der noch aktiven Spieler zu führen.
- Der Spieler, der als letztes über ist, hat gewonnen.

Aufgabe:

Starten Sie für die Aufrufe der `Medienverwaltung` in der GUI eignene Threads. Überlegen Sie sich, welche Resourcen zu schützen sind.

8 Quellen

Theis, Thomas Einstieg in Python. Ideal für Programmiereinsteiger, 5. Auflage, 2017