Inhaltsverzeichnis

[1. Kommandos und Programme 2](#_Toc92873771)

[a. Interaktive Shell 2](#_Toc92873772)

[2. Programme schreiben 2](#_Toc92873773)

[3. Bytecode und Maschinencode 2](#_Toc92873774)

[4. Datentypen und Variablen 3](#_Toc92873775)

[5. Sequenzielle Datentypen 3](#_Toc92873776)

[6. Dictionaries 3](#_Toc92873777)

[a. Definition 3](#_Toc92873778)

[b. Fehlerfreie Zugriffe auf Dictionaries 4](#_Toc92873779)

Quelle: Einführung in Python 3 von Bernd klein, Hanser-Verlag 2013

# Kommandos und Programme

## Interaktive Shell

Er steht zwischen den Anwender und dem Betriebssystem. Dabei schützt der Shell das Betriebssystem vor dem Anwender und anderer Seite erspart er dem Anwender von dem Basis-Funktionen des Betriebssystems.

Eigenschaft:

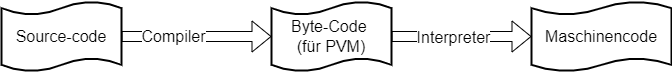
* Ein Befehl wird nach dem Drücken der Eingabetaste direkt ausgeführt.
* Der Unterstrich(\_) wird als Variable zum Aufrufen des vorherigen Ergebnisses verwendet.
* Vorheriger Befehl mit (↑) aufrufbar
* Verlassen des Shells: Ctrl-D oder mit der Funktion exit()
* Variable können deklariert werden und bleiben während der ganzen Session erreichbar.
* Schleife sind auch möglich

# Programme schreiben

* Ein Programm von der Datei hello.py 🡪 in dem Linux oder Windows Shell: python3 hello.py

# Bytecode und Maschinencode

Ablauf zur Ausführung eines Python-Script



PVM: Python-Virtual-Maschine

Fragen:

* Handelt es sich bei Python um eine Programmier- oder eine Skriptsprache?

Programm: Vor Ausführung wird es übersetzt bzw. kompiliert und dann ausgeführt. Kann mehrere Zeilen(tausend, Million) haben.

Skript: wird unmittelbar ausgeführt bzw. interpriert.

Python ist keine Programmiersprache und auch keine Skript Sprache. Aus der oberen Abbildung kann man entnehmen, dass ein Source code zuerst in Byte-Code in einer PVM übersetzt wird und in einer bestimmten Maschinencode übersetzt.

* Wird ein Python-Programm übersetzt oder interpretiert?

Siehe obige Abbildung

* Kann ein Python-code auf verschiedenen Rechnern oder Betriebssystemen laufen?

Ja, sie ist Betriebssystem unabhängig

* Worin liegt der Zusammenhang zwischen Python, Jython oder Java?

Der Ausführungsablauf entspricht dem von Java.

# Datentypen und Variablen

In Python wird der Datentyp einer Variablen automatisch erkannt und kann im Programmablauf geändert werden: Dynamische Typdeklaration.

*Type(i)*

*Isinstance(object,klasse)*

Die Funktion Typ gibt der aktuelle Datentyp von der Variabel i.

Die Funktion isinstance überprüft die Variable/Objekt auf die vorgegebene Klasse/Typ und gibt entweder „True“ oder „False“ zurück. Man kann auch mehrere Datentypen eingeben:

*Isinstance(object,(typ1, typ2))*

Typverletzung: Wenn ungültige Operationen zwischen Variablen verschiedener Datentyp auftreten. ZB. Addieren eines Strings und Integer.

# Sequenzielle Datentypen

Sequenzielle Datentypen sind Variablen, die mehrere Werte haben oder sequenziell angeordnet sind. Man unterscheidet:

* Strings: Anordnung beliebiger Zeichen. Ein Zeichen kann nicht durch Zuweisung geändert werden
* Lists: Anordnung beliebigen Objekten in eckigen Klammern: x = [4, 4.6, “text“]
* Tupel:

Anordnung beliebigen Objekten in runden Klammern: x =(4, 4.6, “text“) und kann nicht mehr im Programm verändert werden.

Tupel-Entpacken:

*T =(“Philip”, “Glass”)*

*(first,last))= T*

*print(first) 🡪 Philip*

*first, \*remainder = 35,99,27,18*

*print(first) 🡪 35*

*print(remainder) 🡪 [99,27,18]*

*first, \*middle, last = 35,99,27,18*

*print(first) 🡪 35*

*print(middle) 🡪 [99,27]*

*print(last) 🡪 18*

Eigenschaften von sequenziellen Datentypen:

* Indexierung ist gleich.
* Die Slice- oder Ausschneiden-Funktionen sind bei allen gültig.

# Dictionaries

## Definition

Es gibt in Python noch die Kategorie Mapping als Datentyp und diese Kategorie hat einen einzigen implementierten Typ: Dictionary. Dictionary beinhaltet mehrere Objekte, die durch Schlüsselwörter abgebildet sind. Sie sind erweiterbar und veränderbar.

*Empty={} 🡪 Leer Dictionary*

*Empty = {“ham” : ‘yes’ , “spam” : ‘no’}*

*print(Empty) 🡪 {‘ham’ : ‘yes’ , ‘spam’ : ‘no’}*

*print(Empty[“ham”]) 🡪 ‘yes’*

*Empty[“spam”]=”yes”*

*print(Empty) 🡪 {‘ham’ : ‘yes’ , ‘spam’ : ‘yes’}*

* Python kann die Reihenfolge der Dictionary beliebig ausgeben.
* Der Zugriff auf einem wert ist durch das Schlüsselwort.
* Bei Zugriff auf einem nicht existierenden Schlüsselwort wird den Error: KeyError erzeugt.

## Fehlerfreie Zugriffe auf Dictionaries

Vor dem Zugriff auf einem Element der Dictionary kann mithilfe des Schlüsselwortes „in“ überprüft werden, ob dieses Element vorhanden ist.

*Empty={} 🡪 Leer Dictionary*

*Empty = {“ham” : ‘yes’ , “spam” : ‘no’}*

*if “ham” in Empty:*

*print(Empty[“ham”]*

*else:*

*print(“Eintrag nicht bekannt”)*

## Zulässige Type für Schlüssel und Werte

Für die Werte gibt es keine Einschränkungen. Für den Schlüssel allerdings können folgende Datentypen nicht verwendet werden: Lists und Dictionary.

## Verschachtelte Dictionaries

*en\_fr={“red” : “rouge”, “green” : “vert”}*

*de\_fr={“rot” : “rouge”, “grün” : “vert”}*

*# Verschachtelte*

*dictionary={“en\_fr” : en\_fr, “de\_fr” : de\_fr}*

*# Zugriff*

*print(dictionary[en\_fr][“red”]) 🡪 rouge*

## Methode auf Dictionaries

D ist im Folgenden eine Variable des Datentyps Dictionary

* D.Clear(): Löschen alle Einträge.
* D.copy(): Erzeugt eine flache Kopie.
* D2= D.fromkeys(D, “yellow”): liefert ein Dictionary zurück, bei dem alle Einträge von dem Dictionary D den Wert „yellow“ haben. Sollte der zweite Parameter fehlen, werden die Werte des Dictionary auf „None“ gesetzt.
* D.get(“Schlüssel“): Gibt den Wert vom „Schlüssel“ oder None, wenn der Eintrag nicht vorhanden ist.
* D.items(): liefert ein Objekt vom Typ „dic\_item“ zurück , das vom ähnlichen Typ wie Dictionary ist: Ein Feld mit Tupeln aus Schlüssel und Werte vom D als Element.
* D.keys(): Liefert ein Objekt vom Datentyp „dic\_item“ zurück: Ein Feld mit Werten vom D als Einträge.
* D.pop(“Schlüssel“): Entfernt den Schlüssel mit seinem Wert und gibt ihn dabei der Wert zurück zur Bestätigung.
* D.popitem(): Entfernt einen beliebigen Schlüssel mit seinem Wert und gibt beide zurück
* D.setDefault(“Schlüssel”,”wert”): Fügt den Schlüssel mit dem Wert hinzu und gibt den Wert zurück. Wenn der Wert nicht eingegeben wird, wird der Schlüssel auf None gesetzt.
* D.update(D2): Fügt die Elemente von D2 in D hinzu. Dabei werden die Schlüssel von D, die bereits in D2 existiert, überschrieben.

## Operatoren auf Dictionaries

D ist eine Variable des Datentyps Dcitionary.

* len(D): gibt die Anzahl der Schlüssel-Werte-Paare zurück.
* Zip(lists1, lists2,lists3):

Lists1, lists2 und lists3 bilden hier eine Matrix. Durch die Funktion Zip wird diese Matrix transponiert. Sind die Listen verschieden lang, bestimmt der kleinste Vektor die Länge der transponierten Matrix.

Zum Entpacken der Transponierten Matrix: Zip(\*T)

Z1=[11,12,13]

Z2=[21,21,23]

Z3=[31,32,33]

T=zip(Z1,Z2,Z3)

print(list(T)) 🡪[(11, 21, 31), (12, 21, 32), (13, 23, 33)]

## Dictionaries aus Listen erzeugen

Die Liste muss zweier Tupel als Element haben.

Z1=["Pizza","Sauerkraut","Paela"]

Z2=["Italien","Deutschland","Spanien"]

T=zip(Z1,Z2)

print(list(T)) 🡪[('Pizza', 'Italien'), ('Sauerkraut', 'Deutschland'), ('Paela', 'Spanien')]

T=list(T)

Dictionary=dict(T)

print(Dictionary) 🡪 {'Pizza': 'Italien', 'Sauerkraut': 'Deutschland', 'Paela': 'Spanien'}

# Mengen

In Python entspricht der Datentyp „set“ („Collection“-Typ) eine Menge, bei der ein Element nur einmal vorkommt.

## Deklaration

Sollte bei der Deklaration mit „set“ Elemente doppelt vorkommen, werden sie bei Speichern vereinzeln.

laender= {"Italien","Deutschland","Spanien"}

laender= set(("Italien","Deutschland","Spanien","Italien"))

print(laender) 🡪{"Italien","Deutschland","Spanien"}

* Element einer Menge können nicht veränderlichen (mutable) Objekte sein. Wie Zum Beispiel Liste oder Dictionary

NB.: „frozensets“ sind wie „set“ aber nicht veränderbar.

## Operationen auf „set“-Objekten

Methode:

S eine Variable des Datentyps „set“

* S.add(element): Fügt ein Element in der Menge hinzu, falls es nicht existiert
* S.clear(): leert die Menge
* S.copy(): Erzeugt eine flache Kopie.
* S1.difference(S2): liefert die Elemente in Menge von S1, die sich nicht in S2 befinden.
* S1.difference(S2): Liefert eine Menge, wobei die Elemente von S2, die sich in S1 befinden, gelöscht werden.

x={"a","b","c", "d","e"}

y={"b","c"}

z={"c","d"}

x.difference\_update(z)

print(x) 🡪 {'a', 'b', 'e'}

* S.discard(element): Löscht das Element in S. Wenn das Element nicht vorhanden ist, passiert nichts.
* S.remove(element): Löscht das Element in S. Wenn das Element nicht vorhanden ist, wird ein Fehler generiert.
* S1.intersection(S2) oder S1 & S2: Gibt die Schnittmenge von S1 und S2: S1∩S2
* S1.isdisjoint(S2): Gibt True Zurück, wenn die Schnittmenge leer ist. Andersfalls False.
* S1.issubset(S2) oder S1<S2: Gibt True zurück, wenn S1 eine Untermenge von S2 ist. Andersfalls False.
* S1.issuperset(S2) oder S1>S2: Gibt True zurück, wenn S1 eine Obermenge von S2 ist. Andersfalls False.
* S.pop(): Löscht ein beliebiges Element in der Menge und gibt es zurück.

# Verzweigungen

# Schleifen

## While

Neu: Es ist möglich „else“ bei einer „While“-Schleife hinzufügen. Der Teil wird ausgeführt, wenn die Bedingung der „While“-Schleife nicht erfüllt wird und die Schleife nicht durch eine Break-Anweisung abgebrochen wurde.

x= "YES"

y="No"

while x!=y:

    y=input()

    print("raté")

else:

    print("trouvé")

## For- Schleife

Die For-Schleife in Python dient dazu eine Datensequenz zu iterieren. Eine Variable (index) wird bei jeder Iteration den Wert des entsprechenden Elements sein.

Es ist auch möglich einen else-Block hinzufügen und funktioniert wie bei der While-Schleife.

x= ["C", "C+", "Perl", "Python"]

for index in x:

    print(index)

else:

    print("Terminé")

🡪 C

C+

Perl

Python

Terminé

# Module importieren

Import math

math.sin(zahl)

Pi= math.pi

dir(math) 🡪 Zeigt alle Funktionen der Klasse math

help(math.tan) 🡪 Gibt Information zu der Fonction tan in der klasse math

# Dateien lesen und schreiben

Datei: ist eine Menge von logisch zusammenhängenden und meist sequentiell geordneten Daten, die auf einem Speichermedium dauerhaft gespeichert werden und mittels eines Bezeichners bwz. Namen wieder identifizierbar und damit ansprechbar sind.

Der Inhalt jeder Datei besteht grundsätzlich aus einer eindimensionalen Aneinanderreihung von Bits, die normalerweise in Byte-Blöcke zusammengefasst interpretiert werden. Dabei wird durch das Anwendungsprogramm und das Betriebssystem entschieden, ob es sich um eine Textdatei, ausführbares Programm, Musikstück oder Bild handelt.

## Text aus einer Datei lesen

Zum Lesen einer Textdatei wird die Funktion open(filename, mode) verwendet:

* Filename: Datentyp Sting und kann auch ein Pfad sein.
* Mode: „r“, „w“. Ohne diese Angabe wird der Mode automatisch „r“.

Beispiel:

fobj=open("C:\\Users\\E1376231\\Desktop\\Document.txt", "r")

iterator=0

for line in fobj:

    if iterator==0:

        print(line.rstrip())

        break

    iterator +=1

fobj.close()

Dieser Code liest die erste Zeile der Textdatei „Document.txt“.

* Die Funktion open() erzeugt eine Dateiobjekt und liefert eine Referenz zu diesem Objekt als Ergebniswert zurück.
* Die For-Schleife bearbeitet die Textdatei hier Zeile pro Zeile. Das Ende einer Zeile wird durch Zeilenumbruch identifiziert.
* rstrip() ist eine Funktion der Klasse String und entfernt Leerzeichen und newlines vom rechten Rand
* Durch die Funktion close() wird die Datei geschlossen.

## Schreiben in eine Datei

Beispiel:

fobj\_in=open("C:\\Users\\E1376231\\Desktop\\Document.txt", "r")

fobj\_out=open("C:\\Users\\E1376231\\Desktop\\Document2.txt", "w")

iterator=0

for line in fobj\_in:

    lineout= str(iterator)+ " "+ line.rstrip() +"\n"

    fobj\_out.write(lineout)

    iterator +=1

    print(lineout)

fobj\_in.close()

fobj\_out.close()

* Mit der Mode „w“ (write) wird die Datei „Document2.txt“ erstellt, wenn er im Ordner nicht vorhanden ist. Wenn die Datei schon existiert, wird der Inhalt gelöscht.
* "\n" ermöglicht den Zeilenumbruch.
* Mit der Funktion write() wird in der Datei geschrieben.
* Am Ende müssen die zwei Dateien geschlossen werden.

## Die Methoden read() und readlines()

* Mit readlines () wird die Datei gelesen und der Inhalt als Feld mit Zeilen der Textdatei als Elemente zurückgeliefert. Somit können die Zeilen der Textdatei durch Indexierung angesprochen werden.

Readlines() öffnet und schließt direkt die Datei.

Beispiel:

text=open("C:\\Users\\E1376231\\Desktop\\Document.txt").readlines()

print(text)

print(text[0])

Ausgabe

["je m'appele Franc. \n", "C'est bien moi\n", 'et on va manger. ']

je m'appele Franc.

* Mit read() wird die Datei gelesen und der Inhalt als String zurückgeliefert. Somit können die Zeichen der Textdatei durch Indexierung angesprochen werden.

text=open("C:\\Users\\E1376231\\Desktop\\Document.txt").read()

print(text)

print(text[0])

Ausgabe

je m'appele Franc.

C'est bien moi

et on va manger.

j

Leerzeichen am Ende der Zeilen werden aufgenommen.