Python Merkblatt-1/3

Python Grundlagen

Hinweise

Mit Leerzeichen aufpassen - sie verursachen große Unterschiede im Code.

Insbesondere die Einrückung von Codeblöcken beinflusst die Ausführung!

Mit **#** werden Kommentarzeilen markiert, die nicht ausgeführt werden.

”””
Drei aufeinanderfolgende Anführungszeichen werden für mehrere Zeilen umfassende Strings verwendet.
”””

Objekttypen

Mittels **type** kann angezeigt werden, zu welcher Klasse ein Objekt gehört:

type(3) Ausgabe: **<type ’int’>**
type(3.14) Ausgabe: **<type ’float’>**

Mit **isinstance** kann geprüft werden, ob ein Objekt eine Instanz der angegebenen Klasse ist:

isinstance(”, str) Ausgabe: **True**
isinstance(1, str) Ausgabe: **False**

Dateneingabe

Die Dateneingabe kann in Python im Terminal über den Befehl **input** ausgeführt werden:

a = input()
Speichere die eingegebenen Zeichen in **a**

b = int(input())
Speichere die eingegebene Ganzzahl in **b**

a, b = map(int,input().split())
Trenne die Eingabe an Leerzeichen und speichere das Ergebnis als Ganzzahlen in **a** und **b**

input(”Drücke ENTER”)
Fahre fort durch das Drücken von **ENTER**.

Programmlogik

if

- **if test:**
  ……..# Führe dies aus, wenn **test** wahr ist
  elif test2:
  ……..# Führe dies aus, wenn **test2** wahr ist
  else:
  ……..# Führe ansonsten dies aus

while

- **while test:**
  ……..# Wiederhole dies, solange **test** wahr ist

for

- **for x in sequence:**
  ……..# Führe dies für jedes Element **x** in
  ……..# **sequence** (Liste, String, ...) aus
- **for x in range(10):**
  ……..# Wiederhole 10 mal (**x** ist 0 bis 9)
- **for x in range(5,10):**
  ……..# Wiederhole 5 mal (**x** ist 5 bis 9)

Ausnahmen

- **try:**
  ……..# Führe diesen Code aus
  except:
  ……..# Führe diesen Code aus, falls eine Exception aufgetreten ist

- **else:**
  ……..# Führe diesen Code aus, falls keine Exception aufgetreten ist
- **finally:**
  ……..# Führe anschließend immer diesen Code aus

Schleifenkontrolle

- **break** Schleifenabbruch

- **continue** Gehe an den Anfang der Schleife

Strings

Ein String ist eine unveränderliche (”immutable”) Folge von Zeichen.

Erzeugung

ein_string = ”Hello World!”
anderer_string = ’Hallo Welt!’

Zugriff

ein_string[4] Ausgabe: **’o’**
(Dies gibt das fünfte Zeichen im Text wieder - Indizes beginnen mit Null!)

Teilen

ein_string.split() Ausgabe **[’Hello’,’World’]**
(Dies teilt den String an Leerzeichen in eine Liste von Teilstrings)
ein_string.split(’r’) Ausgabe **[’Hello Wo’,’ld’]**
(Dies teilt den String am Buchstaben **”r”**)

Vereinigung

Verwende die Funktion **join()** zum Vereinigen einer Liste von Strings:

eine_liste = [”dies”,”ist”,”eine”,”Liste”,”von”,”Strings”]
’ ’.join(eine_liste) Ausgabe: **”dies ist eine Liste von Strings”**
”.join(eine_liste) Ausgabe: **”diesisteineListevonStrings”**

Stringoperationen

Gegeben seien **a = [’Hello’]** und **b = [’Python’]**

Op.	Beschreibung	Beispiel
+	Konkatenation - füge die beiden Strings zu einem neuen String zusammen	**a + b** Ausgabe: **HelloPython**
*****	Repetition - wiederhole den String N mal	**a*2** Ausgabe: **HelloHello**
[ ]	Slice - gib das Zeichen am gegebenen Index zurück	**a[1]** Ausgabe: **”e”**
[:]	Slice Range - Gib alle Zeichen zwischen Index 1 (inklusive) und Index 2 (exklusive) zurück	**a[1:4]** Ausgabe: **”ell”**
in	Vorkommen - gib **True** zurück, wenn das Zeichen im String enthalten ist	**”H”in a** Ausgabe: **True**
not in	Mitgliedschaft - gib **True** zurück, wenn das Zeichen nicht im String enthalten ist	**”M”not in a** Ausgabe: **True**

Stringformatierung

Mit dem Operator **%** können Variablen im String eingefügt werden:
print(”Hallo ihr %s!” % ”alle”) Ausgabe: **”Hallo ihr alle!”**

Symbol	Bedeutung	Symbol	Bedeutung
%c	Zeichen	%i	Ganzzahl (Integer) mit Vorzeichen
%o	Oktale Integer	%x	Hexadezimale Integer (Kleinbuchstaben)
%f	Gleitkommazahl	%X	Hexadezimale Integer (Grossbuchstaben)
		%s	Konvertierung zu einem String

f-Strings

Ab Python3.6 gibt es sogenannte f-Strings zur einfacheren Stringformatierung; Dabei können beliebige Python-Variablen mit ihrem Namen in Strings eingebunden werden:

name, alter = ”Helga”, 32
f”Hallo, {name}. Du bist {alter}.”
Ausgabe: **’Hallo, Helga. Du bist 32.’**

Python Merkblatt-2/3

Weitere Sprachelemente

Wichtige Befehle

Befehl	Beschreibung
is	Teste die Identität eines Objekts
return	Beende die Funktion und gib gegebenenfalls einen Wert zurück
lambda	Erzeuge eine anonyme Funktion
global	Definiere eine global im gesamten Script verfügbare Variable
raise	Erzeuge eine Exception (Ausnahme)
del	Lösche ein Objekt
pass	**pass** tut nichts und kann verwendet werden, wenn ein Befehl syntaktisch notwendig ist, ohne dass das Programm etwas tun muss
assert	Überprüfe eine Bedingung und erzeuge gegebenenfalls eine Exception
exec	Führe einen String von Python-Code aus
yield	Erstellen von Generatoren

Module

Aufbau eines Moduls

Module sind einfache Python-Dateien, die Konstanten, Funktionen und Klassen zum Importieren bereitstellen. Dabei wird üblicherweise die Python-Datei in einem gleichnamigen Verzeichnis abgelegt, beispielsweise **modul/modul.py**. In dem Verzeichnis muss ausserdem eine leere Datei namens **__init__.py** liegen.

import modul.modul
Importiere die gesamte Datei als **modul.modul**

from modul.modul import konstante
Importiere nur **konstante**

from modul.modul import konstante as k
Importiere **konstante** mit dem neuen Namen **k**

from modul.modul import *
Importiere alle Konstanten, Funktionen und Klassen

Operatoren

Gegeben seien **a=10** und **b=20**:

Arithmetische Operatoren

Op.	Beschreibung	Beispiel
+	Addition	**a + b** Ausgabe: **30**
-	Subtraktion	**a - b** Ausgabe: **-10**
*	Multiplikation	**a * b** Ausgabe: **200**
/	Division	**b / a** Ausgabe: **2**
%	Modulo	**a % b** Ausgabe: **0**
**	Exponentiell	**a**b** Ausgabe: 10^{20}
//	Ganzzahlteilung	**9 // 2** Ausgabe: **4**

Vergleichsoperatoren

Die Standard-Vergleichsoperatoren können auf alle Datentypen angewendet werden - Zahlen, Strings, Listen, etc.. Sie geben immer **True** oder **False** zurück.

Op.	Beschreibung	Beispiel
<	Kleiner	**a < b** Ausgabe: **True**
<=	Kleiner gleich	**a <= b** Ausgabe: **True**
==	Gleich	**a == b** Ausgabe: **False**
>	Größer	**a > b** Ausgabe: **False**
>=	Größer gleich	**a >= b** Ausgabe: **False**
!=	Ungleich	**a != b** Ausgabe: **True**
<>	Ungleich	**a <> b** Ausgabe: **True**

Logikoperatoren

Die Logikoperatoren **and** und **or** geben ebenfalls einen boolschen Wert abhängig von der Kondition zurück.

Op.	Beschreibung
and	Ausgabe **True** wenn beide Operatoren wahr sind
or	Ausgabe **True** wenn mindestens einer der beiden Operatoren wahr ist
not	Invertiere die Logik eines boolschen Operators

Listen

Listen

Listen können verschiedene Datentypen beinhalten.

Listenbearbeitung in Python

Erzeugung	
eine_liste = [5,3,'p',9,'e']	Erzeugt: **[5,3,'p',9,'e']**
Zugriff	
eine_liste[0]	Ausgabe: **5**
Slices (Abschnitte)	
eine_liste[1:3]	Ausgabe: **[3,'p']**
Länge	
len(eine_liste)	Ausgabe: **5**
Anzahl eines Elementes in der Liste.	
eine_liste.count('p')	Ausgabe: **1**
Iteriere über die Länge der Liste:	
while x < len(eine_liste):	
Sortieren mit sort()	
eine_liste.sort()	Ausgabe: **[3,5,9,'e','p']**
Erzeuge eine sortierte Kopie der Liste (Original bleibt unverändert)	
print(sorted(eine_liste))	Ausgabe: **[3,5,9,'e','p']**
Ein Element anhängen - append(item)	
eine_liste.append(37)	Ausgabe: **[5,3,'p',9,'e',37]**
Ein Element entfernen - pop(position)	
eine_liste.pop()	Ausgabe: **'e'** und Liste jetzt **[5,3,'p',9, 'e']** - Letztes Element entfernen
eine_liste.pop(1)	Ausgabe: **3** und Liste jetzt **[5,'p',9,'e']** - Zweites Element entfernen
Ein bestimmtes Element entfernen - remove(item)	
eine_liste.remove('p')	Ausgabe: **[5,3,9,'e']**
Einfügen	
eine_liste.insert(2,'z')	Ausgabe: **[5,'z',3,'p',9,'e']** - An bezeichneter Stelle einfügen
Invertieren - reverse()	
reverse(eine_liste)	Ausgabe: **['e',9,'p',3,5]**
Anhängen	
eine_liste+[0]	Ausgabe: **[5,3,'p',9,'e',0]**
eine_liste+eine_liste	Ausgabe: **[5,3,'p',9,'e',5,3,'p',9,'e']**
Mitgliedschaft	
9 in eine_liste	Ausgabe: **True**
Tupel	
tupel = ('a','b','c','d','e')	Tupel sind unveränderliche (immutable) Listen

Python Merkblatt-3/3

www.datamics.com

Python Builtins

Builtins sind wichtige, immer in Python verfügbare Funktionen.

Ausgabe von Variablen

print(”Hallo”)
Ausgabe: **Hallo**

Absoluter Wert

abs(-3)
Ausgabe: **3**

Objektfunktionen und -attribute

dir(’a’)
Ausgabe: **[..., ’partition’, ’replace’, ’rfind’, ...]**

Ist mindestens ein Element wahr?

any([1==2, 1==1])
Ausgabe: **True**

Sind alle Elemente wahr?

all([1==2, 1==1])
Ausgabe: **False**

Attribut vorhanden?

hasattr(’a’, ’replace’)
Ausgabe: **True**

Instanz einer Klasse?

isinstance(’a’, str)
Ausgabe: **True**

Zähle über die Elemente einer Liste

list(enumerate(range(3)))
Ausgabe: **[(0, 0), (1, 1), (2, 2)]**

Erzeuge ein Set

set([1, 1, 2, 3])
Ausgabe: **{1, 2, 3}**

Invertiere eine Liste

list(reversed([1, 2, 3]))
Ausgabe: **[3, 2, 1]**

Runde eine Gleitkommazahl

round(1.34, 1)
Ausgabe: **1.3**

Maximum und Minimum einer Liste

print(max([1, 2]), min([1, 2]))
Ausgabe: **2 1**

Funktion auf eine Liste anwenden

list(map(str, [1, 2]))
Ausgabe: **[’1’, ’2’]**

Listen zusammenfassen

list(zip([’1’, ’2’, ’3’], [’a’, ’b’, ’c’]))
Ausgabe: **[(’1’, ’a’), (’2’, ’b’), (’3’, ’c’)]**

Dictionaries

Dictionaries sind assoziative Speicher des Typs Schlüssel/Wert. Wert kann hierbei verschiedene Datentypen beinhalten.

Erzeugen

d = {’name’: ”franz”, ”alter”: 121}

Zugriff

d[’name’]
Ausgabe: **’franz’**

Wert hinzufügen (oder ueberschreiben)

d[’kontostand’] = 10.99

Schlüssel vorhanden?

”id”in d
Ausgabe: **False**

Zugriff auf alle Schlüssel/Wert-Paare

d.items()
Ausgabe: **dict_items([(’name’, ’franz’), ...])**

Element löschen

del d[’kontostand’]

Zwei Dictionaries vereinigen

d.update({”hausnr”: 3})

Schlüssel anzeigen

d.keys()
Ausgabe: **dict_keys([’name’, ’id’, ...])**

Werte anzeigen

d.values()
Ausgabe: **dict_values([’franz’, 121, 10.99, 3])**

Datum und Zeit

Python verwendet unter anderem das Modul **time** zum Behandeln von Datum und Zeit.

TimeStruct

import time
time.localtime().tm_year
Ausgabe (z.B.): **2019**

Formatierte Zeit

time.asctime(time.localtime(time.time()))
Ausgabe (z.B.): **’Tue Jun 25 11:02:41 2019’**

Vergleich

a = time.localtime()
time.sleep(1)
a > time.localtime()
Ausgabe: **False**
a < time.localtime()
Ausgabe: **True**

Fortgeschrittene Themen

Ternäre Operatoren

Ternäre Operatoren führen if...else-Bedingungen in einer Zeile aus:
print(True if 1 == 2 else False) Ergebnis: **False**

Listenabstraktion

Mit Listenabstraktion (list comprehension) können Listen in einer Zeile erzeugt und bearbeitet werden:
print([xx for x in [1, 2, 3] if x != 2])** Ergebnis: **[1, 27]**

Definieren von Funktionen

Funktionen dienen der wiederholten Ausführung von Programmcode und werden mit dem Schlüsselwort **def** gekennzeichnet:

```
def entferne_strings(liste):
    ”Entferne alle Strings aus einer Liste. ”
    return [x for x in liste if type(x) != str]

print(entferne_strings([’aa’, 1, ’2’]))
```

Ergebnis: **[1]**
Unterhalb der Funktionsdefinition steht hier ein mit drei öffnenden und drei schliessenden Anführungszeichen gekennzeichneter **Docstring** mit einer Erläuterung der Funktion. Dieser kann mit **help(entferne_strings)** angezeigt werden.

Definieren von Klassen

Klassen dienen der Verwaltung zusammengehöriger Datentypen und Funktionen und werden mit dem Schlüsselwort **class** gekennzeichnet:

```
class Familie:
    mitglieder = []
    def mitglied_hinzufuegen(self, name):
        self.mitglieder.append(name)
```

Dabei ist **self** ein Parameter, der die Instanz einer Klasse bezeichnet:

```
familie = Familie()
familie.mitglied_hinzufuegen(”Hans”)
print(familie.mitglieder)
```

Ergebnis: **[”Hans”]**

Debugging mit IPDB

Nach der Installation des Debuggers IPDB (mit pip3 install ipdb) kann dieser mit dem folgenden Befehl an einer beliebigen Stelle im Programmcode aufgerufen werden, um beispielsweise Variablen und Konditionen zu überprüfen:

```
import ipdb; ipdb.set_trace()
```