Inhaltsverzeichnis

1 Allgeimes

Die Hauptseite ist `https://pypi.org/`, der Python Package Index.
Als Entwicklungsumgebung sind jupyter und spyder die bekanntesten, jedoch reicht ein besserer Texteditor und ipython auch aus. Auf Windows wird empfohlen Python via Anaconda zu installieren

2 Datentypen

- Python erkennt den Datentyp automatisch
- Variablen sind **immer** Referenzen auf Objekte

```
1 ct = int
```

SubTex/Code/DatenTypen.py

- Wert- und Typänderung während der Laufzeit erlaubt, da ein neues Objekt referenziert wird
- Datentypen prüfen:

```
type(objekt)  # returns Type
2 isinstance(objekt, ct)  # checks if
```

SubTex/Code/DatenTypen.py

- Python achtet auf Typverletzungen
- Python kennt keine implizite Typumwandlung

Datentyp	**Beschreibung**	**False-Wert**
NoneType	Indikator für nichts, keinen Wert	None
Numerische Datentypen		
int	Ganze Zahlen	0
float	Gleitkommazahlen	0.0
bool	Boolesche Werte	False
complex	komplexe Zahlen	0 + 0j
Sequenzielle Datentypen		
str	Zeichenketten oder Strings	’ ’
list	Listen (veränderlich)	[]
tuple	Tupel(unveränderlich)	()
bytes	Sequenz von Bytes (unveränderlich)	b’ ’
bytearray	Sequenz von Bytes (veränderlich)	bytearray(b’ ’)
Mengen		
set	Menge mit einmalig vorkommenden Objektion	set()
frozenset	Wie set jedoch unveränderlich	frozenset()
Assoziative Datentypen		
dict	Dictionary (Schlüssel-Wert-Paare)	{}

Nummerische Datentypen können „nur“ Zahlenwerte annehmen.
Sequenzielle Datentypen sind vergleichbar mit Arrays. Werte können mehrmals vorkommen und jeder einzelne besitzt einen spezifischen Index.
Mengen sind vergleichbar mit mathematischen Mengen, jedes Element ist einzigartig und die Reihenfolge ist willkürlich.
Assoziative Datentypen sind ähnlich wie die Sequenziellen Datentypen, jedoch besitzen sie keinen nummerischen Index, aber Keys aus (unveränderlichen) Datentypen.

2.1 Nummerische Datentypen - Arithmetische Operatoren

Operator	Beschreibung
$x + y$	Summe von x und y
$x - y$	Differenz von x und y
$x * y$	Produkt von x und y
x/y	Quotient von x und y
$x//y$	Ganzzahliger Quotient von x und y
$x\%y$	Rest der Division von x durch y
$+x$	Positives Vorzeichen
$-x$	Negatives Vorzeichen
$abs(x)$	Betrag von x
$x**y$	Potenzieren, x^y

Achtung: $x++$ und $x--$ existieren **nicht**, aber $x+=1, x-=1, x*=2, \cdots$
Bitweise Operatoren sind vergleichbar mit C:

- Vergleichende Operatoren $(==, !=, <=, \dots)$
- Bitweise Operatoren $(\&, |, \wedge, \dots)$

Zahlen werden nicht verändert, es wird ein neues Python-Objekt erstellt.

2.2 Sequentielle Datentypen

```
1 string1 = "Ich bin ein String"
  string2 = 'Ich bin auch ein String!'
3 string3 = """Ich bin ein
  mehrzeiliger String."""
5 string4 = '''Ich bin auch ein mehrzeiliger String!'''
```

SubTex/Code/DatenTypen.py

Wenn ein String verändert wird, wird ein neues Python-Objekt erstellt.

```
1 liste1 = [1, 1, 2, 3, 'vier']
  liste2 = [liste1, 5, 'sechs']
```

SubTex/Code/DatenTypen.py

Listen können beliebige Datentypen enthalten

```
tuple1 = (1, 1, 2, 3, 'vier')
tuple2 = 1, 1, 2, 3, 'vier'
```

SubTex/Code/DatenTypen.py

Tuples sind wie Listen, jedoch sind Tuples fix und können nicht mehr verändert werden. Aber die Werte einer z.B. Liste die ein Element des Tuples ist kann verändert werden, da das Element an sich nicht ändert

```
zahl1, zahl2, zahl3, zahl4, string1 = liste1  # oder tupel
```

SubTex/Code/DatenTypen.py

Tupel/Liste unpacking, funktioniert wie bei Funktionen, für jeden Rückgabewert kann man eine Variable (mit Komma abgetrent) hinzufügen, die diesen Wert annimmt. Sequentielle Datentypen sind alle indexierbar. Der Index startet immer bei Null. Man indexiert mit eckigen Klammern (

). Man kann über negative Indices Rückwärts indizieren.

2.2.1 Sliccing

Der Startwert ist inklusive, der Endwert ist exklusive.

```
slice = x[start: end: step]
```

SubTex/Code/DatenTypen.py

Beim Slicing gibt es keine Fehlermeldung wie Out of Bounds es gibt (falls der Bereich kommplett ausserhalb liegt) nur den Leeren Typ zurück.
Beim Slicing kann wie folgt gekürzt werden:

```
x[:end:step] == x[0:end:step]
x[start::step] == x[start:0:step]
x[start:end] == x[start:end:1]
```

SubTex/Code/DatenTypen.py

2.3 set/frozenset

- ungeordnete, unindexierte, veränderliche Sammlung von unveränderlichen Elementen
- Jedes Element kommt nur einmal vor
- Frozensets sind unveränderlich

```
set1 = {1, 1, 2, 3, 'vier'}
set2 = frozenset(set1)
```

SubTex/Code/DatenTypen.py

2.3.1 „set“ Funktionen

```
primes.add(11)  # Element wird hinzugefuegt (solange nicht vorhanden)
# .add veraendert den set

x = n.intersection(primes)
# gibt set zurueck mit Elementen, die in beiden sets vorkommen

n.intersection_update(primes)
# Vergleichbar mit:
# y = n.intersection(primes)
# n = y

y = n.difference(primes)
# gibt set zurueck mit Elementen, die nur in "n" vorkommt

n.difference_update(primes)
# Vergleichbar mit:
# y = n.difference(primes)
# n = y
```

SubTex/Code/DatenTypen.py

2.4 Assoziative Datentypen - Dictionary

```
steckbrief = {
        "Vorname": "Pippi",
        "Nachname": "Langstrumpf",
        "Alter": 9,
        "Hobbies": ["Reiten", "Spielen"],
}
```

SubTex/Code/DatenTypen.py

- Ugeordnete Sammlung von Schlüssel-Wert-Paaren
- Jeder Schlüssel kommt nur einmal vor
- Schlüssel-Objekt muss immutable(unveränderlich) sein (int, float, string, bool, tuple)

2.4.1 „Dictionary“ Funktionen

Die Elemente eines Dictionary können ähnlich wie bei einem Array addreassiert werden, nur dass Keys und nicht indizes verwendet werden.

```
buch["Erscheinungsjahr"] = 1980  # wird ueberschrieben
print(buch["Titel"])
```

SubTex/Code/DatenTypen.py

Jedoch gibt diese Art vom addressieren einen Key-Error. Besser ist die get-Funktion.

```
zahl = buch.get("Seitenzahl", 0)
```

SubTex/Code/DatenTypen.py

Mit dieser Funktion kann man eine Rückgabe definieren, falls der Key nicht vorhanden ist.

3 Python Funktionen

Dieses Kapitel befasst sich mit Python-Standart-Funktionen.

3.1 Typen

Da Python Typen nicht implizit umwandeln kann,müssen sie explizit umgewandelt werden.

list	wandelt iterierbare Datentypen in eine Liste um
tuple	wandelt iterierbare Datentypen in ein Tupel um
string	wandelt Datentypen in Strings um (bei Listen u.ä. werden die Entsprechenden Klammern und Kommatas mit übernommen)
int	wandelt (Strings und Floats) in Integer um (Strings müssen zahlen mit Basis 10 sein)
float	wandelt (Strings und Integer) in Floats um (Strings müssen Basis 10 sein und wenn mit Nachkommastellen, muss es mit einem Dezimalpunkt getrennt werden
set	wandelt in sets um
frozenset	wandelt in frozensets um
bin	wandelt Integer in ein binären String der Form „0b...“ um
oct	wandelt Integer in ein oktalen String der Form „0o...“ um
hex	wandelt Integer in ein hexadezimalen String der Form „0x...“ um
len	gibt die Anzahl der Elemente von Iterierbaren Datensätzen zurück („nur die erste Stufe“)

3.2 IO

print	schreibt in die Konsole
input	gibt einen String zurück mit der Konsoleneingabe, der Wert in der Klammer wird zuerst in die Konsole geschrieben

```
zahl1 = input("1. Zahl: ")
```

SubTex/Code/DatenTypen.py

3.3 „range“

Die range()-Funktion kreirt ein range-Objekt, welches iterierbar bar ist. Mit dieser Funktion lassen sich sehr gut Listen initialisieren.

```
range(stop) == range(0, stop, 1)
range(start, stop) == range(start, stop, 1)
range(start, stop, step)
```

SubTex/Code/DatenTypen.py

Start ist inklusive, stop ist exklusive. Step gibt die Schrittweite an, der Standart ist 1. Kann auch in negativer richtung (mit einem minus) durchlaufen werden (start und stop müssen angepasst werden).

4 for/if/while

In Python existieren for, while und if-else, jedoch fehlt do-while.