Datenstrukturen

Inhalt:

Im letzten Programm wurde eine Variable `sum` benutzt.
Ihr wurde der Wert für die Berechnung `1 + 2` zugewiesen.

Jede Variable muss einen Datentyp besitzten. In Python muss dieser aber nicht deklariert werden, sondern ergibt erstmall aus den enthaltenen Werten.

In [ ]:
```
# program.py
sum = 1 + 2 # Das Ergebnis der Summenberechnung von 1 und zwei wird an die Variabl
print(sum) # Der Wert 3 von sum wird auf dem Bilschirm ausgegeben.
```

Die Variable `sum` hat also nie einen anderen Wert gehabt als 3.

Da die Zahl 3 eine Ganzzahl ist, ergibt sich daraus, das die Variable den Datentype *integer* hat.

Abfragen des Datentypes mit der Type-Funktion `type()`

In [ ]:
```
# program2.py
sum = 0.5
print(type(sum))
sum = 1 + 2
print(type(sum))
```

```
<class 'float'>
<class 'int'>
```

Die Datentypen von Variablen können sich im Verlauf eines Programms ändern.

1. Übersicht über die wichtigsten Datentypen

Bezeichnung	Abkürzung	Bedeutung	Beispiel
integer	int	ganze Zahl	3
float	float	Fließkommazahl	3.1
string	str	Zeichenkette	"Paprika, Salami"
boolean	bool	boolesche Werte (Wahrheitswerte)	True oder False
list	list	Liste	["Salami", "Margherita"]
tuple	tuple	Tupel (Elemente nicht veränderbar)	('Pizza','Salat','Eis')
dictionaries	dict	Dictionaries (Schlüssel/ Wert Paare)	{'car':'Auto', 'dog': 'Hund'}

Eine vollständige Liste der Datentypen in Python:
https://docs.python.org/3/library/stdtypes.html#

1.1 Integer - ganze Zahlen

Integers sind ganze Zahlen, also positive und negative Zahlen ohne Komma oder Null.
Ab Python 3 können integers so lang sein, wie der Arbeitsspeicher hergibt, es gibt keine Unterscheidung mehr in integer und long integer wie in Python das bis Version 2.7 der Fall war.

In [ ]:
```
# Beispiel 1 Integer
pizza_Salami = 1
print(pizza_Salami)
```

```
1
```

In [ ]:
```
# Beispiel 2 Integer
pizza_Bestellung = pizza_Salami * 2
print(pizza_Bestellung)
```

```
2
```

1.2 Float - Fließkommazahlen

In [ ]:
```
# Beispiel Float
cocaCola = 1.5
print(cocaCola)
```

```
1.5
```

In [ ]:
```
# Beispiel 2 Float
cocaColaBestellung = cocaCola * 2
print(cocaColaBestellung)
```

```
3.0
```

1.3 String - Zeichenkette

In [ ]:
```
# Beipiel 1 String
pizzaBelagStephanie = "Paprika, Salami, Thunfisch, Oliven, Zwiebeln, Käse"
print(pizzaBelagStephanie)
```

```
Paprika, Salami, Thunfisch, Oliven, Zwiebeln, Käse
```

In [ ]:
```
# Beipiel 2 String
pizzaBelagStephanie = "Paprika, Salami, Thunfisch, Oliven, Zwiebeln, Käse"
print(pizzaBelagStephanie)
print(type(pizzaBelagStephanie))

pizzas = "5"
print(pizzas)
print(type(pizzas))
```

```
Paprika, Salami, Thunfisch, Oliven, Zwiebeln, Käse
<class 'str'>
5
<class 'str'>
```

Mit Strings können keine Rechenoperationen durchgeführt werden.

In [ ]:
```
# Beipiel 3 String
bestellung = pizzas * 3
```

```
print(bestellung)
```

```
555
```

Konvertieren des Datentypes

Funktion	Ergebnis
int(value)	konvertiert in ein Integer
float(value)	konvertiert in ein Float
str(value)	konvertiert in ein String

Bevor mit der Variable `pizzas` gerechnet werden kann, muss eine Datentyp-Umwandlung in entweder integer oder float erfolgen.

In [ ]:
```
# Beipiel 3 String
bestellung = int(pizzas) * 3
print(bestellung)
```

```
15
```

1.4 Boolean - boolesche Werte

Boolesche Werte sind Ausdrücke, die entweder *True* (wahr) oder *False* (falsch) sind.

In [ ]:
```
pizzaSalami = 3
pizzaMozarella = 1
pizza_Salami = pizzaMozarella
print(pizzaMozarella == pizza_Salami)
```

```
True
```

1.5 List - Liste

Eine Liste ist zunächst erstmal ein zusammengesetzter Datentyp.
In einer Liste können Strings, Floats und Integers kombiniert werden.
Listen-Elemente habe eine feste Reihenfolge und eine Indexnummer für jedes Element, mit Start `[0]`. Neue Elemente werden ans Ende gesetzt.

Die Elemente einer Liste können verändert werden und es können Elemente doppelt vorkommen.

In [ ]:
```
# Beispiel 1 Liste
pizzas = [3, "Salami", 4, "Margeherita"]
print(pizzas)
```

```
[3, 'Salami', 4, 'Margeherita']
```

In [ ]:
```
print(type(pizzas))
```

```
<class 'list'>
```

In [ ]:
```
# Beipiel 2 Liste
print(pizzas[1]) # das 2. Element der Liste ausdrucken
```

```
Salami
```

In [ ]:
```
# Beispiel 3 Liste
pizzas[3] = "Peperoni" # ändern des 4. Listen-Elementes
```

```
print(pizzas) # Ausdruck der neuen Liste
```

```
[3, 'Salami', 4, 'Peperoni']
```

In [ ]:
```
# Beipiel 4 Liste - die Länge der Liste ausdrucken
print(len(pizzas))
```

```
4
```

Methoden für die Arbeit mit Listen:
https://www.w3schools.com/python/python_lists_methods.asp

1.6 Tuple - Tupel

Tupels sind ebenfalls eine Art Liste. Sie unterscheiden sich vom Type `list` dadurch, das sie unveränderlich sind. Man erkennt sie dadurch das die Elemente in runden Klammern stehen.

In [ ]:
```
# Beispiel 1 Liste
pizzas = (3, "Salami", 4, "Hawaii")
print(pizzas)
```

```
(3, 'Salami', 4, 'Hawaii')
```

In [ ]:
```
print(type(pizzas))
```

```
<class 'tuple'>
```

In [ ]:
```
print(pizzas[3])
```

```
Hawaii
```

In [ ]:
```
# Beispiel 3 Liste
pizzas[3] = "Peperoni" # wir versuchen das 4. Element zu ändern
print(pizzas) # Ausdruck der neuen Liste
```

Methoden um mit Tupels zu arbeiten:
https://www.w3schools.com/python/python_tuples_methods.asp

1.7 Dictionaries

Man kann sich Dictionaries als eine Menge von Schlüssel-Wert-Paaren vorstellen. Während bei Listen der Zugriff auf ein Element über die Position (Index) erfolgt, geschieht die Zuordnung und der Zugriff bei Dictoionaries über Schlüssel (key).
Ein Schlüsselpaar steht in Anführungsstrichen, einfach oder doppelt wenn es sich um einen String handelt und hat den Doppelpunkt als Trennungszeichen zwischen sich.
Für das Dictionarie werden geschweifte Klammern verwendet. **Dictionaries können keine doppelten Keys enthalten.** Doppelte Werte dagegen schon. Der Key steht links, der Wert rechts.

In [ ]:
```
# Beispiel 1 Dictionaries - erstellen, zugreifen

woerterbuch = {"tree":"Baum", "house":"Haus", "car":"Auto"}

print(woerterbuch["tree"])

# zugreifen mit get
woerterbuch.get("house")
```

In [ ]:
```
# Beispiel 2 Dictionaries - hinzufügen

woerterbuch["street"]="Strasse"

print(woerterbuch)
```

In [ ]:
```
# Beispiel 3 Dictionaries - ändern

woerterbuch["car"]= "Automobil"

print(woerterbuch)
```

In [ ]:
```
# Beispiel 3 Dictionaries - löschen

#del woerterbuch["street"]

print(woerterbuch)
```

Dictionaries können auch verschachtelt werden, wie es in dem Spiel Escape The Room gemacht wird.

2. Operatoren

2.1 Arithmetische Operatoren

Mit Hilfe von *arithmetischen* Operatoren können Berechnungen wie Addition, Subtration, Multiplikation und Division durchgeführt werden. Hier die wichtigsten arithmetischen Operatoren:

Type	Bewchreibung	Beipiel	Ergebnis
+	Additionsoperator	1 + 1	2
-	Subtraktionsoperator	2 - 1	1
/	Divisionsoperator	10 / 2	5
//	Floor Divisionsoperator	11 / 4	2
%	Modulusoperator	11/4	3
*	MMultiplikationsoperator	2 * 2	4

2.2 Zuweisungsoperatoren

Mit Hilfe von Zuweisungsoperatoren können sie einer Variablen während ihres gesamten Lebenszyklus Werte zuzuweisen.

Operator	Beispiel	Beschreibung	Ergebnis
=	x = 2	x enthält jetzt "2"	2
+=	x += 2	x wird um 2 erhöht	4
-=	x -= 2	wird um 2 verringert	2
/=	x /= 2	wird durch 2 dividiert	1
*=	x *= 2	wird mit 2 multipliziert	2

3. Datum und Zeit

Um mit Datum oder Zeit in einem Python-Programm arbeiten zu können, muss `date` oder `time` importiert werden.

3.1 Datum

In [ ]:
```
# Beispiel 1 - importieren von date

from datetime import date
```

Anschließend können alle Funktionen, die in `date` enthalten sind genutzt werden.

In [ ]:
```
# Beispiel 2 - aufrufen der Funktion today()
date.today()
```

Out[ ]:
```
datetime.date(2022, 5, 18)
```

In [ ]:
```
# Beispiel 3 - ausgeben des Datums auf dem Bildschirm
print(date.today())
```

```
2022-05-18
```

In [ ]:
```
# Beispiel 4 - ein bestimmtes Datum ausgeben

weihnachten2022 = date(2022, 12, 24)
print(weihnachten2022)
```

```
2022-12-24
```

In [ ]:
```
#  Beispiel 5 - Formatiertes Datum ausgeben

print(weihnachten2022.strftime("%d.%m.%Y"))
```

```
24.12.2022
```

In [ ]:
```
# Beispiel 6 - Wochentag anhand einer Liste als Text ausgeben

from datetime import date
aktuellesDatum = date.today()
wochentag_nr = aktuellesDatum.isoweekday()
print(wochentag_nr)

wochentage_kuerzel = ["So", "Mo", "Di", "Mi", "Do", "Fr", "Sa"]
print("aktueller Wochentag: ", wochentage_kuerzel[wochentag_nr])
```

```
3
aktueller Wochentag:  Mi
```

In [ ]:
```
# Beispiel 7 - Mit Tagen rechnen

from datetime import date
heute = date.today()
print(heute)

heute_umf = heute.strftime("%m-%d-%Y. %d.%b.%Y ist ein %A am %d. Tag des %B.")
print(heute_umf)

# mit dem Datum lässt sich rechnen
geburtstag = date(1969,10,5)
heute = date.today()
```

```
alter = heute - geburtstag
print(alter.days, "Tage seit Geburt vergangen")
```

```
2022-05-18
05-18-2022. 18.May.2022 ist ein Wednesday am 18. Tag des May.
19218 Tage seit Geburt vergangen
```

3.2 Zeit

Über das `time` Modul können in Python Zeitberechnungen gemacht werden. Ausserdem können Pausen in den Programmaublauf mit Hilfe von `sleep` eingebaut werden.

In [ ]:
```
# Beispiel 1 - time importieren

import time
```

In [ ]:
```
# Beispiel 2 - die Zeit ausgeben

print(time.localtime())
print(time.gmtime())

jetzt = time.gmtime()
print(jetzt[0])          # gibt das Jahr aus

print(jetzt[6])          # gibt den Wochentag als Zahl aus
```

```
time.struct_time(tm_year=2022, tm_mon=5, tm_mday=18, tm_hour=14, tm_min=52, tm_sec
=38, tm_wday=2, tm_yday=138, tm_isdst=1)
time.struct_time(tm_year=2022, tm_mon=5, tm_mday=18, tm_hour=12, tm_min=52, tm_sec
=38, tm_wday=2, tm_yday=138, tm_isdst=0)
2022
2
```

In [ ]:
```
# Beispiel 3 - mit time Programm pausieren

import time
print("Ich bin müde und gehe schlafen")
time.sleep(5)
print("habe geschlafen")
```

```
Ich bin müde und gehe schlafen
habe geschlafen
```

In [ ]:
```
# Beispiel 4 - Zeit auswerten

import time
print(time.time())
zeitanfang = time.time()
print("Ich bin müde und gehe schlafen")
time.sleep(5)
print("habe geschlafen")
zeitende = time.time()
print("Dauer Programmausführung:",)
print(zeitende-zeitanfang)
```

```
1652878688.9424887
Ich bin müde und gehe schlafen
habe geschlafen
Dauer Programmausführung:
5.0062196254730225
```