2. Datentypen und Verzweigungen

Wie im Kapitel „1. Einstieg in Python“ gezeigt, werden Variablen als Objekte betrachtet und muss daher kein Datentyp festgelegt werden. Es gibt zwei Typen von Objekten:

a. Einzelne Objekte (Zahlen oder Zeichen)
b. Gruppen von Objekten (Zeichenketten, Listen, Tupel, Dictionarys und Sets)

Datentypen

Zunächst werden wir uns mit Zahlen und Operationen mit Zahlen, sowie mit einigen eingebauten Funktionen und mathematischen Funktionen (Modul **math**) beschäftigen.

Zahlen

a.) Ganze Zahlen {-2, -1, 0, 1, 2, 3,...} als ganze Zahlen $\mathbb{Z}$

Als Objekttyp für die ganzen Zahlen dient **int** (engl. Integer). Zahlen dieses Typs sind unendlich genau. Standardmäßig wird das dezimale Zahlensystem mit der Basis 10 benutzt. Es stehen in Python auch weitere Zahlensysteme zur Verfügung:

Zahlensysteme	Funktion	Präfix
Duale Zahlensystem (mit der Basis 2, auch Binärsystem genannt)	bin()	0b
Oktale Zahlensystem (mit der Basis 8)	oct()	0o
Hexadezimal Zahlensystem (mit der Basis 16)	hex()	0x

Die Funktionen bin(), oct() und hex() dienen zur Umrechnung und Ausgabe der Zahl in das entsprechende System. Bei der Eingabe oder Zuweisung muss das Präfix 0b, 0o, bzw. 0x vor der eigentlichen Ziffernfolge stehen, damit das zugehörige Zahlensystem erkannt wird.

b.) Kommazahlen {rationale und irrationale Zahlen} als reelle Zahlen $\mathbb{R}$

Der Datentyp für Kommazahlen heißt **float** und werden auch Fließkommazahlen genannt. Diese werden mit Hilfe des Dezimalpunkts oder der Exponentialschreibweise angegeben. Unterstriche lassen sich bei Zahlen mit vielen Ziffern für bessere Lesbarkeit einsetzten.

```
a = 7.5
b = 2e2
c = 3.5E3
d = 4.2e-3
e = 1_250_00.500_001

print (a, b, c, d, e)
```

Ausgabe: 7.5 200.0 3500.0 0.0042 1250000.500001

Mit der Funktion **type()** kann eine Zahl als ganze Zahl (Integer-Zahl) oder als Fließkommazahl (float-Zahl) ermittelt werden. Der Typ eines Objekts wird ausgegeben.

Bisher haben wir die mathematischen Operatoren für Addition +, Subtraktion -, Multiplikation * und Division / verwendet. Darüber hinaus gibt es in Python zwei weitere Operatoren:

%	Modulo	gibt den Rest einer Ganzzahldivision
**	Potenz	damit wird die Potenzzahl angegeben

Die ebenfalls eingebaute Funktion **round()** dient zur Rundung einer Zahl. Eine Übersicht der eingebauten Funktionen finden Sie am Ende des Skripts.

Auf **mathematische Funktionen** und **Konstanten** kann über das **Modul math** zurückgegriffen werden. Am Anfang des Python-Programms wird mit

```
1 import math
```

Das Modul aufgerufen und es können folgende mathematische Funktionen und Konstanten im Programm genutzt werden:

math.sqrt()	- berechnet die Quadratwurzel von eine Zahl
math.log()	- berechnet den natürlichen Logarithmus einer Zahl
math.exp()	- berechnet e^a von einer Zahl a
math.log10()	- berechnet den 10er-Logarithmus einer Zahl
math.factorial()	- berechnet den Wert der Fakultät einer positiven ganzen Zahl
math.sin()	- Winkel-Fkt. Sinus
math.cos()	- Winkel-Fkt. Cosinus
math.tan()	- Winkel-Fkt. Tangens

Alle drei o.g. trigonometrische Funktionen beziehen sich auf eine Angabe des Winkels im Bogenmaß. Daher wird der Winkel zuvor mit Hilfe der Funktion *radians()* von Grad in Bogenmaß umgewandelt.

math.radians()	- wandelt eine Zahl von Grad in Bogenmaß um
math.degree()	- wandelt von Bogenmaß in Grad um
math.pi	- gibt die π-Zahl
math.e	- gibt die Eulersche Zahl
math.isclose(a, b)	- gibt an, ob eine Zahl nahe dran an einer gegeben Zahl ist oder nicht
math.prod(a, b)	- berechnet das Produkt der Elemente
math.gcd(a,b)	- berechnet den größten gemeinsamen Teiler (=GGT)
math.remainder(a, b)	- berechnet den Rest einer Division

Erstelle ein Python-Programm zur Berechnung des Bedarfs an Wandfarbe „Alpina Weiß“ für einen rechteckigen Raum (BxLxH in m).

Der Benutzer soll die Maße des Raums in Metern mit einer Kommastelle eingeben.

Eine 10 l „Alpina weiß“-Eimer kostet 29,19 € inkl. MwSt. und wird pro Anstrich 160ml/m² verbraucht. Als Vereinfachung werden Fenster und Türen des Raums vernachlässigt bzw. dient die überschüssige Farbe als Reserve für Nachbesserungen.

Als Ausgabe soll die Anzahl der „Alpina-Weiß“-Eimern (ganze Zahl) und die Kosten angezeigt werden.

21_Wandfarbe_v0.py - C:/Users/PAZ20/AppData/Local/Programs/Python/Python39/21_Wand...

File Edit Format Run Options Window Help

```
print("Geben Sie zur Berechnung des Bedarfs an Wandfarbe die Raumgröße an.")
print("Breite des Raums in Metern:")
b = float(input())
print("Länge des Raums in Metern:")
l = float(input())
print("Höhe des Raums in Metern:")
h = float(input())

#Berechnung der Fläche
# je 2 mal Wände, Insgesamt 4 Wände plus Decke
# 2 x B x H + 2 x L x H + B x L

FlGesamt = 2 * b * h + 2 * l * h + b * l

print("Die Raumfläche beträgt insgesamt:", FlGesamt, "qm")

#Nun wird der Bedarf an Farbe berechnet
#160ml wird pro qm-Fläche benötigt
# in ml ausgerechnet, Umwandeln in Liter (/1000)

BedarfFarbe = FlGesamt * 160 / 1000

print("Es werden:", BedarfFarbe, "Liter Farbe für den Anstrich benötigt")

#Wie viele Alpina-weiß-Eimer werden benötigt?
#Den berechneten BedarfFrabe wird gerundet und
#durch 10 l-Eimer dividiert

AnzahlEimer = round(BedarfFarbe/10,0)

print(AnzahlEimer, "Eimer (Alpina weiß)")

#Berechnung der Kosten
#29,19 EUR * AnzahlEimer

Kosten = round(AnzahlEimer * 29.19,2)

print("Die Kosten für die Wand-Farbe betragen:", Kosten, "EUR.")
```

Abb. 9: Python-Programm: Bedarf Wandfarbe (ohne if-then-else!)

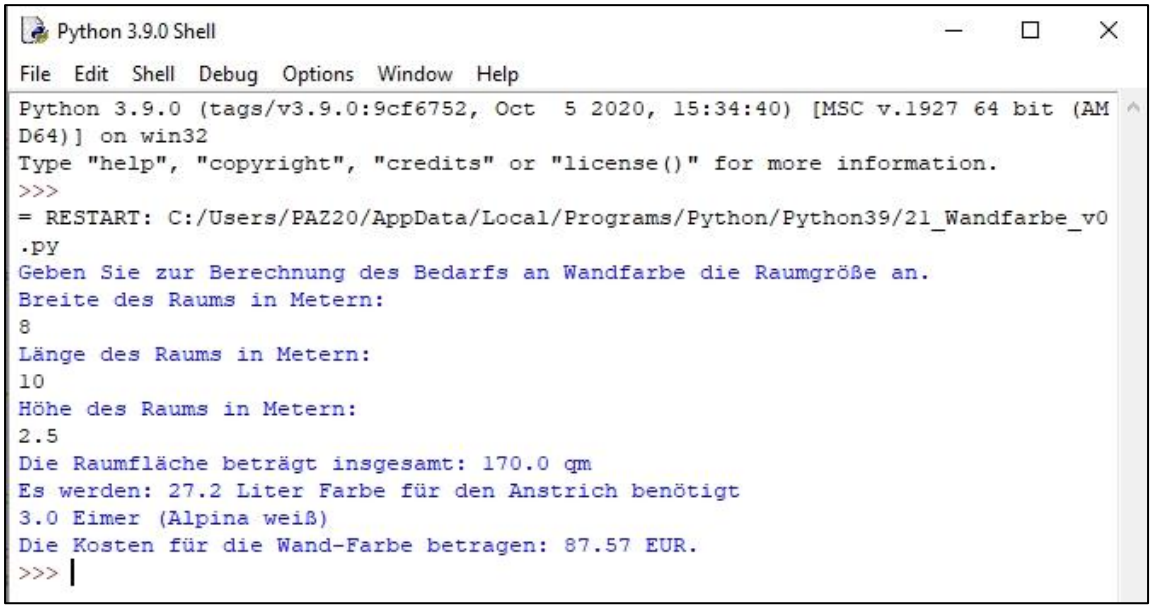


Abb. 10: Python-Programm: Bedarf Wandfarbe (ohne if-then-else!)

Kommentare (kein Datentyp)

Bei umfangreichen Programmen ist es sehr nützlich **Kommentare** zur Erläuterung in den Programmcode mit anzugeben. Kommentare werden durch das Rautezeichen **#** eingeleitet und reichen bis zum Zeilenende. Mehrzeilige Kommentare beginnen und enden jeweils mit drei doppelten Anführungszeichen **"""**. Kommentare werden nicht ausgeführt.

```
1  # Das ist ein einzeiliger Kommentar
2  """Ein Kommentar mit
3  Zwei oder mehreren Zeilen"""
```

Zeichenketten

Zeichenketten sind Sequenzen von einzelnen Zeichen – also Texte. Zeichenketten (engl. Strings) sind Objekte des Datentyps str, die aus mehreren Zeichen oder Wörtern bestehen. Sie werden angegeben mit einfachen, doppelten oder dreimal doppelte Hochkommata.

```
t1 = “Hallo Welt“
t2 = 'Auch das ist eine Zeichenkette'
t3 = ““““Diese Zeichenkette
        steht in
        mehreren Zeilen““““
t4 = 'Hier sind “doppelte Hochkommata“ gespeichert'
```

Die Operatoren + und * dienen zur Verkettung mehrere Sequenzen bzw. zur Vervielfachung einer Sequenz. Mithilfe des Operators in stellt man fest, ob ein bestimmtes Element in einer Sequenz enthalten ist.

Teilbereiche von Sequenzen werden als Slices bezeichnet. Ein Slice wird durch die Angabe eines Bereichs in eckigen Klammern [] hinter der sequenziellen Variablen erzeugt. Er beginnt mit einem Startindex, gefolgt von einem Doppelpunkt und einem Endindex. z.B. [5:8]

Die eingebaute Funktion **len()** ermittelt die Anzahl der Elemente einer Sequenz. Damit kann die Länge einer Zeichenkette berechnet werden. Es gibt für str-Objekte eine Reihe von nützlichen Funktionen zur Bearbeitung und Analyse von Zeichenketten:

count() - ergibt die Anzahl der Vorkommen eines Suchtextes innerhalb des Textes
find() - gibt die Position, an der ein Suchtext innerhalb eines Textes vorkommt
rfind() - gibt die Position des letzten Vorkommens innerhalb der Textes

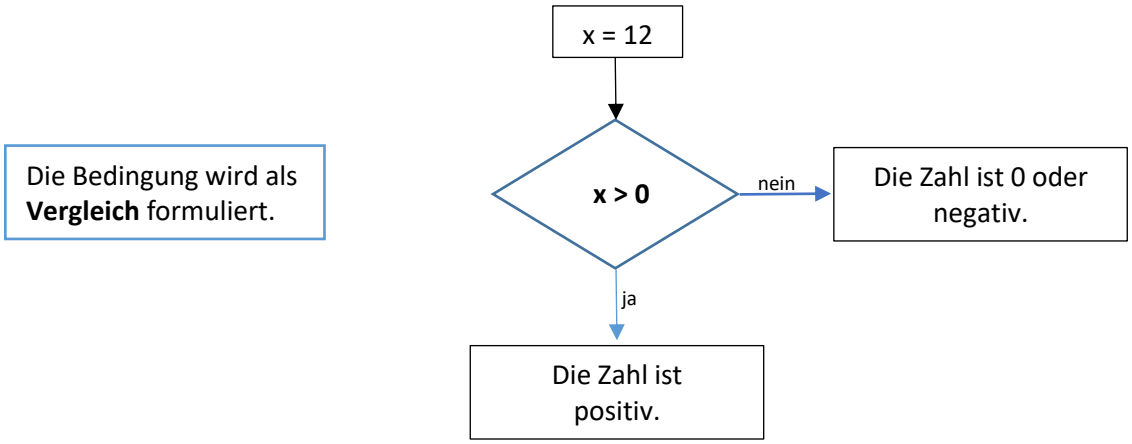
replace() - ersetzt einen gesuchten Teiltext durch einen anderen und liefert den Text

startwith() - untersucht, ob eine Zeichenkette mit einem bestimmten Text beginnt
endwith() - untersucht, ob eine Zeichenkette mit einem bestimmten Text endet

partition() - zerlegt eine Zeichenkette in einzelne Teile anhand eines Textes
split() - zerlegt einen Text in einzelne Teile

Verzweigungen

Ohne den Verzweigungen würden alle Anweisungen im Programm der Reihe nach ausgeführt. Zur Steuerung des Programmablaufs werden Verzweigungen benötigt. Anhand einer Bedingung entscheidet sich, auf welchem Weg das Programm fortgesetzt wird.



Die Bedingung wird als **Vergleich** zwischen **zwei Zahlen** oder **zwei Texten** formuliert. Dabei wird bei der Programmierung oft an erster Stelle eine Variable stehen, die eine der beiden Zahlen/Texten darstellt.

Setzten wir dieses Beispiel in ein Python-Programm um:

1	x = 12	
2	if x > 0:	
3	print(“Die Zahl ist positiv.“)	← if-Zweig
4	else:	
5	print(“Die Zahl ist 0 oder negativ.“)	← else-Zweig

Mit **if** und die darauffolgende **Bedingung** (wahr oder falsch) z.B. hier **x > 0** und mit einem **Doppelpunkt** abschließend wird eine Verzweigung eingeleitet. Die zum if-Zweig gehörenden Anweisungen werden eingerückt, damit Python die Zugehörigkeit zur Verzweigung erkennen kann. Mit **else**: wird der alternative Weg der Verzweigung eingeleitet. Auch die zum else-Zweig gehörenden Anweisungen werden eingerückt. Die nächste darauffolgende Anweisung, die nicht zum else-Zweig gehört, wird nicht eingerückt.

Bei mehr als zwei Alternativen, wird mit einer weiteren Bedingung verzweigt, man spricht von einer **mehrfachen Verzweigung**. Es bietet sich an die Alternativen sortiert aufzuschreiben mit den dazugehörenden Bedingungen. z.B.

Die Zahl ist negativ.	x < 0
Die Zahl ist 0.	x = 0
Die Zahl ist positiv.	x > 0

Wir setzten es im Python-Programm um:

1	x = int(input())	Beliebige Eingabe
2	if x < 0:	
3	print(“Die Zahl ist negativ.“)	← if-Zweig
4	elif x == 0:	
5	print(“Die Zahl ist 0.“)	← elif-Zweig
6	else:	
7	print(“Die Zahl ist positiv.“)	← else-Zweig

Die Verzweigung wird mit **elif** und einer weiteren Bedingung formuliert (siehe Zeile 4). Innerhalb der Verzweigungen können mehrere **elif**-Anweisungen vorkommen.

Mehrere Bedingungen können mit Hilfe der logischen Operatoren **and**, **or** und **not** miteinander verknüpft werden. Auch mehrere Vergleichsoperatoren **(<, <= , ==, >, >=, !=)** können in einer Bedingung enthalten sein.

Die Rangfolge der Operatoren spielt eine Rolle, wenn in der Bedingung mehrere Operatoren auftreten.

Rechenoperatoren → Vergleichsoperatoren → logische Operatoren
(*, /, %, //, +, -) (<, <= , ==, >, >=, !=) (and, or, not)

Übersicht der eingebauten Funktionen

Funktionsname	Beschreibung
abs()	Liefert den Betrag einer Zahl
bin()	Liefert eine binäre bzw. duale Zahl
bytes()	Liefert ein Objekt des Datentyps bytes
chr()	Liefert ein Zeichen zu einer Uni-Code-Zahl
eval()	Liefert einen ausgeführten Python-Ausdruck
exec()	Führt eine Anweisung aus
filter()	Liefert die Elemente eines iterierbaren Objekts, für die eine Funktion True ergibt
float()	Liefert eine Zahl mit Nachkommastellen
format()	Formatiert Zahlen und Zeichenketten
frozenset()	Liefert ein unveränderliches Set
hex()	Liefert eine hexadezimale Zahl
input()	Wartet auf eine Eingabe des Benutzers
int()	Liefert eine ganze Zahl
len()	Liefert die Anzahl der Elemente (einer Zeichenkette)
map()	Liefert Funktionsergebnisse zu einer Reihe von Aufrufen
max()	Liefert das größte Element
min()	Liefert das kleinste Element
oct()	Liefert eine oktale Zahl
open()	Öffnet eine Datei zum Lesen oder Schreiben
ord()	Liefert die Unicode-Zahl zu einem Zeichen
print()	Erzeugt eine Ausgabe
range()	Liefert ein iterierbares Objekt über einen Bereich
repr()	Liefert Informationen über ein Objekt
reversed()	Liefert ein iterierbares Objekt in umgekehrter Reihenfolge
round()	Liefert eine gerundete Zahl
set()	Liefert ein Set
sorted()	Liefert eine sortierte Liste
str()	Liefert eine Zeichenkette
sum()	Liefert die Summe der Elemente
type()	Liefert den Typ eines Objekts
zip()	Verbindet Elemente aus iterierbaren Objekten