Programmzeilen mit der IDLE Shell testen

Gib die Programmzeilen nach dem Prompt (>>>) ein. Ist das Ergebnis OK oder bekommst du eine Fehlermeldung?

Tipp: Mit ALT + p kannst du die letzte Zeile wiederholen.

Zeichenketten (strings) testen

Programmzeile	OK	Fehler
`Hallo Welt`		
`"Hallo Welt"`		
`"Hallo Welt'`		
`'Hallo Welt'`		
`print("Hallo Welt")`		
`pirnt("Hallo Welt")`		

Zeichenketten in Variablen schreiben

Programmzeile	OK	Fehler
`nachricht = Hallo Welt`		
`nachricht = "Hallo Welt"`		
`nachricht`		
`print(nachricht)`		
`len(nachricht)`		
`type(nachricht)`		
`nachricht.upper()`		

Zahlen (integer und float) testen

Programmzeile	OK	Fehler
`3`		
`3.14`		
`3,14`		
`print(3)`		
`print(3.14)`		
`print(3,14)`		

Zahlen in Variablen schreiben

Programmzeile	OK	Fehler
`zahl = 3`		
`zahl = 3.14`		
`zahl`		
`print(zahl)`		
`len(zahl)`		
`type(zahl)`		
`zahl.upper()`		

Die Farbe kennzeichnet einen Text mit besonderer Bedeutung

Farbe	Bedeutung
rot	
grün	
violett	

Mit der IDLE Shell rechnen

Gib die Programmzeilen nach dem Prompt (>>>) ein. Ist das Ergebnis OK oder bekommst du eine Fehlermeldung?

Rechnen mit Zeichenketten

Programmzeile	OK	Fehler
`"Hallo" + "Welt"`		
`"Hallo" + " " + "Welt"`		
`"Hallo " + "Welt"`		
`"Hallo " - "Welt"`		
`print("Hallo " + "Welt")`		
`3 * "Hallo Welt"`		
`3 * "Hallo Welt "`		
`3 / "Hallo Welt "`		
`print(3 * "Hallo Welt ")`		
`nachricht = "Hallo Welt "`		
`print(3 * nachricht)`		

Rechnen mit Zahlen

Programmzeile	OK	Fehler
`3 + 4`		
`3 - 4`		
`3 * 4`		
`3/4`		
`30//4`		
`30%4`		
`ergebnis = 3 * 4`		
`ergebnis/5`		
`ergebnis//5`		
`print(ergebnis/5)`		

Punktrechnung vor Strichrechnung

Programmzeile	OK	Fehler
`3 + 4 * 2`		
`(3 + 4) * 2`		
`12 - 3 / 2`		
`(12 - 3) / 2`		
`12 - 3 // 2`		
`(12 - 3) // 2`		

Zahl in Zeichenkette umwandeln – Zeichenkette in Zahl umwandeln

Programmzeile	OK	Fehler
`zahl = 3`		
`str(zahl)`		
`zahl = 3.14`		
`str(zahl)`		
`zeichenkette = "3.14"`		
`int(zeichenkette)`		
`float(zeichenkette)`		

Mit dem IDLE Editor ein Programm schreiben

Mit dem Menüpunkt File/ New File erzeugst du ein leeres Editor-Fenster mit dem Titel "untitled".

Gib die Programmzeilen im Editor-Fenster ein. Die Farbe dort kennzeichnet einen Text mit besonderer Bedeutung.

Speichere den Inhalt mit dem Menüpunkt File/ Save unter dem Dateinamen "mein_erstes_programm" im Ordner Python/03_Rechnen_mit_Zeichenketten_und_Zahlen

Starte das Programm mit dem Menüpunkt Run/ Run Module.

Notiere die print-Ausgaben in der Tabelle.

Rechnen

Programmzeile	**print-Ausgabe**
`# Das ist ein Kommentar`	
`# Rechnen`	
`nachricht = "Hallo Welt "`	
`print(3 * nachricht)`	
`ergebnis = 3 * 4`	
`print(ergebnis/5)`	

Umwandeln

Programmzeile	**print-Ausgabe**
`# Umwandeln`	
`zahl = 3.14`	
`print(zahl)`	
`print(str(zahl))`	
`zeichenkette = "3.14 "`	
`print(zeichenkette)`	
`print(3 * zeichenkette)`	
`print(3 * float(zeichenkette))`	

Variablen ausgeben

Programmzeile	**print-Ausgabe**
`# Variablen ausgeben`	
`print("nachricht =", nachricht)`	
`print("ergebnis =", ergebnis)`	
`print("zahl =", zahl)`	
`print("zeichenkette =", zeichenkette)`	

Aufgabe: Ziffern mit Punkten darstellen

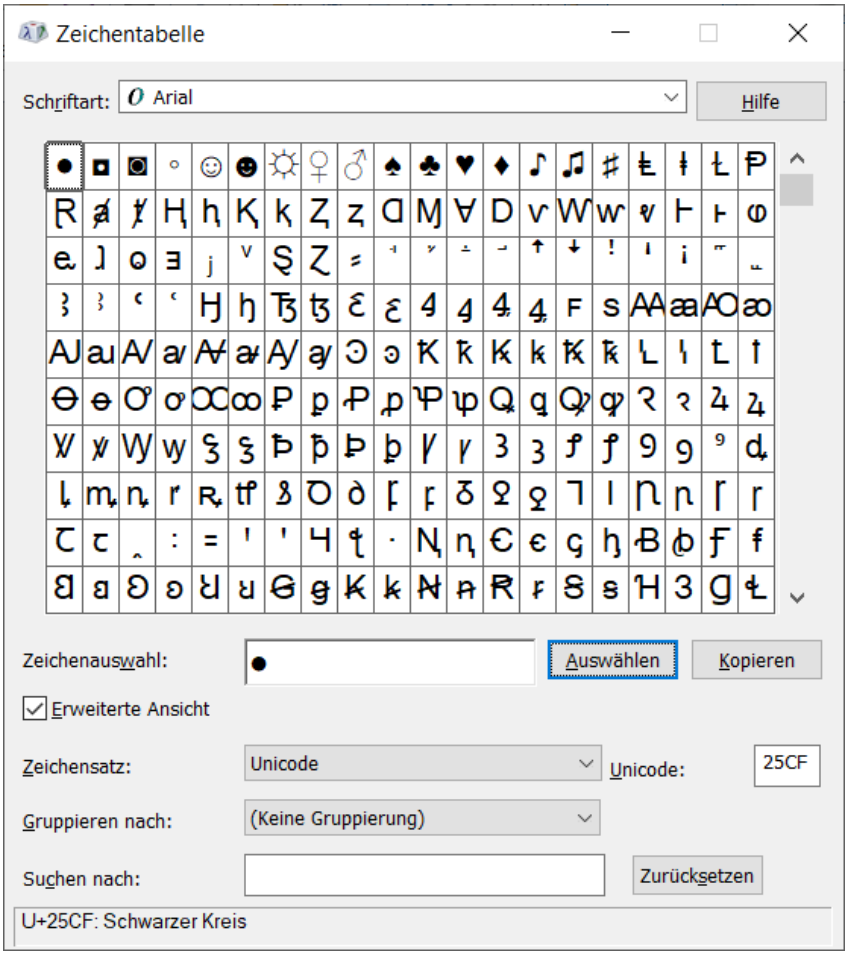
Die Ziffern auf einem Kassenzettel sind aus Punkten zusammengesetzt. Mit 5x7 Punkten können die Ziffern 0 bis 9 gut lesbar dargestellt werden.

	●	●	●	
●				●
●				●
●				●
●				●
●				●
	●	●	●	

Schreibe ein Programm, das mit print-Befehlen und dem Zeichen * oder ● eine Ziffer in ein 5x7 Raster druckt:

a) Drucke eine Null
b) Drucke eine Acht
c) Drucke eine Eins
d) Drucke eine Ziffer deiner Wahl

Tipp: Das Zeichen ● hat den Unicode 25CF. Es kann über die Windows-Zeichentabelle eingegeben werden.



Eine Liste speichert viele änderbare Elemente

Gib die Programmzeilen nach dem Prompt (>>>) ein.

Merke: Der Index steht in eckigen Klammern. Der Index beginnt mit Null!

Notiere die print-Ausgaben in der Tabelle.

Eine Liste anlegen – mit eckigen Klammern!

Programmzeile	print-Ausgabe
`vornamen = ["Axel", "Elke", "Martin]`	
`print(vornamen)`	
`print(vornamen[0])`	
`print(vornamen[0:2])`	
`print(vornamen[-1])`	
`vornamen[2] = "Fritz"`	
`print(vornamen)`	

Eine Liste erweitern

Programmzeile	print-Ausgabe
`vornamen = vornamen + ["Heike", "Sabine"]`	
`print(vornamen)`	
`vornamen += ["Markus"]`	
`print(vornamen)`	

Eine leere Liste anlegen und füllen

Programmzeile	print-Ausgabe
`buchstaben = []`	
`buchstaben.append("a")`	
`print(buchstaben)`	
`buchstaben.append("b")`	
`print(buchstaben)`	

Elemente einer Liste löschen

Programmzeile	print-Ausgabe
`print(vornamen)`	
`vornamen.remove("Heike")`	
`print(vornamen)`	
`del vornamen[0]`	
`print(vornamen)`	
`del vornamen`	
`print(vornamen)`	

Ein zufälliges Element aus einer Liste auswählen

Programmzeile	print-Ausgabe
`import random`	
`handzeichen = ["Schere", "Stein", "Papier"]`	
`print(random.choice(handzeichen))`	

Ein Tupel speichert viele nicht änderbare Elemente

Gib die Programmzeilen nach dem Prompt (>>>) ein.

Merke: Der Index steht in eckigen Klammern. Der Index beginnt mit Null!

Notiere die print-Ausgaben in der Tabelle.

Ein Tupel anlegen – mit runden Klammern!

Programmzeile	**print-Ausgabe**
`punkt = (-10, 5, 7)`	
`print(punkt)`	
`print(punkt[0])`	
`print(punkt[0:2])`	
`print(punkt[-1])`	
`punkt[2] = 15`	
`punkt = (-10, 5, 15)`	
`print(punkt)`	

Ein Element im Tupel suchen

Programmzeile	**print-Ausgabe**
`print(punkt)`	
`print(punkt.count(7))`	
`print(punkt.index(7))`	

Ein Dictionary speichert Paare von Schlüssel und Wert

Mit dem Menüpunkt File/ New File erzeugst du ein leeres Editor-Fenster mit dem Titel "untitled".

Gib die Programmzeilen im Editor-Fenster ein.

Speichere den Inhalt mit dem Menüpunkt File/ Save unter dem Dateinamen "woerterbuch" im Ordner Python/04_Listen_und_Woerterbuecher

Starte das Programm mit dem Menüpunkt Run/ Run Module.

Notiere die print-Ausgaben in der Tabelle.

Ein leeres dictionary anlegen – mit geschweiften Klammern – und füllen

Programmzeile	**print-Ausgabe**
`# Wörterbuch Englisch - Deutsch`	
`# Leeres dictionary anlegen`	
`englisch_deutsch = {}`	
`# dictionary füllen`	
`englisch_deutsch["cat"] = "Katze"`	
`englisch_deutsch["dog"] = "Hund"`	
`englisch_deutsch["cow"] = "Kuh"`	
`print(englisch_deutsch)`	
`print(englisch_deutsch["dog"])`	
`englisch_deutsch["sheep"] = "Schaf"`	
`print(englisch_deutsch)`	

Schlüssel und Werte des dictionary ausgeben

Programmzeile	**print-Ausgabe**
`# Schlüssel ausgeben`	
`print(englisch_deutsch.keys())`	
`print(englisch_deutsch.values())`	

Neues dictionary mit Vertauschung von Schlüssel und Wert erstellen

Wenn jeder Wert nur einmal im dictionary vorkommt, geht das mit folgender Programmzeile (Erläuterung später).

Programmzeile	**print-Ausgabe**
`# neues dictionary erstellen`	
`deutsch_englisch = dict((v,k) for k,v in englisch_deutsch.items())`	
`print(deutsch_englisch)`	
`# Schlüssel ausgeben`	
`print(deutsch_englisch.keys())`	
`print(deutsch_englisch.values())`	

Der Computer fragt …

Mit dem Menüpunkt File/ New File erzeugst du ein leeres Editor-Fenster mit dem Titel "untitled".

Gib die Programmzeilen im Editor-Fenster ein.

Speichere den Inhalt mit dem Menüpunkt File/ Save unter dem Dateinamen "summe_ausgeben" im Ordner Python/05_Benutzereingaben

Starte das Programm mit dem Menüpunkt Run/ Run Module.

Notiere die print-Ausgaben in der Tabelle.

Der Computer erwartet Zahlen

Programmzeile	**print-Ausgabe**
`# Summe von zwei Zahlen ausgeben`	
`# Benutzereingaben anfordern`	
`zahl1 = input("Gib die erste  Zahl ein ")`	
`zahl2 = input("Gib die zweite Zahl ein ")`	
`# Strings in Dezimalzahlen umwandeln`	
`zahl1 = float(zahl1)`	
`zahl2 = float(zahl2)`	
`# Summe ausgeben`	
`print("Die Summe der Zahlen ist", zahl1 + zahl2)`	

Der Computer erwartet Strings

Programmzeile	**print-Ausgabe**
`# Summe von zwei Strings ausgeben`	
`# Benutzereingaben anfordern`	
`str1 = input("Gib den ersten  String ein ")`	
`str2 = input("Gib den zweiten String ein ")`	
`# Summe ausgeben`	
`print("Die Summe der Strings ist", str1 + str2)`	

Aufgabe: Wörterbuch erweitern

Das Dictionary englisch_deutsch soll erweitert werden. Schreibe das Programm dazu.

a) Lege das Dictionary englisch_deutsch an und fülle es mit 3 Paaren.
b) Fordere ein neues englisches Wort an – den Schlüssel.
c) Fordere das passende deutsche Wort an – den Wert.
d) Erweitere das Dictionary mit Schlüssel und Wert.
e) Drucke das erweiterte Dictionary

Was passiert, wenn der Schlüssel bereits vorhanden ist?

Der Computer unterscheidet zwischen "wahr" und "falsch"

Gib die Programmzeilen nach dem Prompt (>>>) ein.

Notiere die print-Ausgaben in der Tabelle.

Bedingung mit Zahlen

Programmzeile	print-Ausgabe
`print(1 == 2)`	
`print(1 != 2)`	
`print(1 < 2)`	
`print(1 > 2)`	
`print(3 == 3)`	
`print(3 != 3)`	
`print(3 <= 3)`	
`print(3 >= 3)`	

Bedingung mit Strings

Programmzeile	print-Ausgabe
`print("Hallo" == "Welt")`	
`print("Hallo" != "Welt")`	
`print("Montag" == "Montag")`	
`print("Montag" != "Montag")`	

Bedingung mit range(stop)

Programmzeile	print-Ausgabe
`bereich = range(10)`	
`print(list(bereich))`	
`print(1 in bereich)`	
`print(10 in bereich)`	

Bedingung mit range(start, stop)

Programmzeile	print-Ausgabe
`bereich = range(2, 10)`	
`print(list(bereich))`	
`print(1 in bereich)`	
`print(9 in bereich)`	

Bedingung mit range(start, stop, step)

Programmzeile	print-Ausgabe
`bereich = range(0, 10, 2)`	
`print(list(bereich))`	
`print(3 in bereich)`	
`print(8 in bereich)`	

Der Computer unterscheidet Fälle

Wenn die Bedingung "wahr" ist, werden die Programmzeilen darunter ausgeführt

Gib die Programmzeilen nach dem Prompt (>>>) ein.

Notiere die print-Ausgaben in der Tabelle.

Eine Bedingung – zwei Fälle

Programmzeile	**print-Ausgabe**
`wert = 5`	
`if wert < 10:`	
`    print("wert ist kleiner als 10")`	
`    print("Ich gehöre auch zu der Bedingung")`	

Eine Bedingung und die Alternative – zwei Fälle

Programmzeile	**print-Ausgabe**
`if wert < 10:`	
`    print("wert ist kleiner als 10")`	
`else:`	
`    print("wert ist größer oder gleich 10")`	

Mehrere Bedingungen und die Alternative – vier Fälle

Programmzeile	**print-Ausgabe**
`if wert == 10:`	
`    print("wert ist gleich 10")`	
`elif wert == 4:`	
`    print("wert ist gleich 4")`	
`elif wert == 5:`	
`    print("wert ist gleich 5")`	
`else:`	
`    print("keine Bedingung ist erfüllt")`	

Der Computer dreht Schleifen

Solange die Bedingung wahr ist, werden die Programmzeilen darunter ausgeführt

Mit dem Menüpunkt File/ New File erzeugst du ein leeres Editor-Fenster mit dem Titel "untitled".

Gib die Programmzeilen im Editor-Fenster ein.

Speichere den Inhalt mit dem Menüpunkt File/ Save unter dem Dateinamen "schleifen" im Ordner Python/07_Fallunterscheidungen_und_Schleifen

Starte das Programm mit dem Menüpunkt Run/ Run Module.
Notiere die print-Ausgaben in der Tabelle.

while Schleife

Programmzeile	**print-Ausgabe**
`# Schleifen`	
`# while-Schleife`	
`# Variable initialisieren`	
`durchgang = 1`	
`while durchgang < 11:`	
`    print(durchgang)`	
`    durchgang = durchgang + 1`	
`print("nach der Schleife")`	

Eine unendliche while Schleife abbrechen – break

Programmzeile	**print-Ausgabe**
`# Variable initialisieren`	
`durchgang = 1`	
`# unendliche Schleife mit Abbruch`	
`while True:`	
`    print("durchgang =", durchgang)`	
`    durchgang = durchgang + 1`	
`    if durchgang > 10:`	
`        break`	
`print("Nach der Schleife")`	

for Schleife – mit Liste

Programmzeile	**print-Ausgabe**
`# Liste anlegen`	
`vornamen = ["Axel", "Elke", "Martin"]`	
`# Solange es ein Element in der Liste gibt`	
`for element in vornamen:`	
`    print(element)`	
`print("nach der Schleife")`	

for Schleife – mit range(stop)

Programmzeile	**print-Ausgabe**
`# Solange es ein Element in der Liste gibt`	
`for element in range(10):`	
`    print(element)`	
`print("nach der Schleife")`	

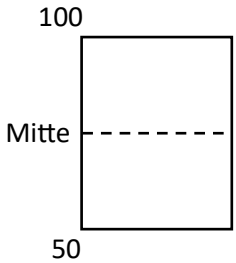
Aufgabe: Dialog mit dem Benutzer

Der Computer reagiert auf eine Benutzereingabe

Der Computer soll den Benutzer nach einem Buchstaben fragen und auf den Buchstaben reagieren. Schreibe das Programm dazu.

a) Der Computer soll solange fragen, bis der Benutzer "e" eingibt.
b) Jede Benutzereingabe soll gedruckt werden.

Die Mitte zwischen zwei Zahlen berechnen



Erweitere das Programm. Berechne die Mitte zwischen der unteren und der oberen Grenze.

a) untere = 50 obere = 100
b) Berechne und drucke die ganzzahlige Mitte zwischen "untere" und "obere"
c) Teste deinen Ausdruck mit neuen Grenzen

Der Computer reagiert auf weitere Benutzereingaben

Erweitere das Programm.

a) Der Computer soll solange fragen, bis der Benutzer "e" eingibt.
b) Wenn der Benutzer "k" eingibt, soll "kleiner" gedruckt werden.
c) Wenn der Benutzer "g" eingibt, soll "größer" gedruckt werden.

Funktionen übernehmen eine Teilaufgabe

Funktionen mit Ganzzahlen (Integer) als Input und Output

Mit dem Menüpunkt File/ New File erzeugst du ein leeres Editor-Fenster mit dem Titel "untitled".

Gib die Programmzeilen im Editor-Fenster ein.

Speichere den Inhalt mit dem Menüpunkt File/ Save unter dem Dateinamen "funktionen_io" im Ordner Python/09_Funktionen_Input_und_Output

Starte das Programm mit dem Menüpunkt Run/ Run Module.

Notiere die print-Ausgaben in der Tabelle.

Funktion ohne Input

Programmzeile	**print-Ausgabe**
`# Funktionen können Input und Output haben`	
`# Funktion ohne Input`	
`def ausgabe():`	
`    print("hier bin ich")`	
`# Aufruf der Funktion`	
`ausgabe()`	

Funktion mit 2 Inputs

Programmzeile	**print-Ausgabe**
`# Funktion mit 2 Inputs`	
`def ausgabe2a(wert1: int, wert2: int):`	
`    print("wert1 =", wert1, "wert2 =", wert2)`	
`# Aufruf der Funktion`	
`ausgabe2a(5, 6)`	

Funktion mit 2 Inputs und Vorgabe

Die Werte mit Vorgabe stehen rechts von den Werten ohne Vorgabe.

Programmzeile	**print-Ausgabe**
`# Funktion mit 2 Inputs und Vorgabe`	
`def ausgabe2b(wert1: int, wert2: int = 15):`	
`    print("wert1 =", wert1, "wert2 =", wert2)`	
`# Aufruf der Funktion`	
`ausgabe2b(5)`	

Funktion mit 1 Input und 1 Output

Programmzeile	**print-Ausgabe**
`# Funktion mit 1 Input`	
`def verdoppeln(wert: int) -> int:`	
`    return wert * 2`	
`# Aufruf der Funktion`	
`ergebnis = verdoppeln(5)`	
`print("ergebnis =", ergebnis)`	

Funktionen übernehmen eine Teilaufgabe – Fortsetzung

Funktion mit Tupel als Output

Programmzeile	print-Ausgabe
`# Funktion mit Tupel als Output`	
`def wo_bin_ich() -> tuple[int, int]:`	
`    x = 2`	
`    y = 4`	
`    return x, y`	
`# Aufruf der Funktion`	
`x, y = wo_bin_ich()`	
`print("x =", x, "y =", y)`	

Funktion mit einer Liste als Input

Achtung: Eine Liste ist am Ort veränderbar (mutable object).

Input der Funktion ist eine Kopie der Liste, damit das Original unverändert bleibt.

Programmzeile	print-Ausgabe
`# Funktion mit einer Liste als Input`	
`def verteuerung(liste: list[float], p:[float]):`	
`    for i in range(len(liste)):`	
`        liste[i] *= 1 + p`	
`# Liste anlegen`	
`original = [9, 12, 12.5, 24.5]`	
`# Liste kopieren`	
`kopie = original.copy()`	
`# Prozentsatz festlegen`	
`p = 0.05`	
`# Aufruf der Funktion`	
`verteuerung(kopie, p)`	
`# Original und Verteuerung`	
`print("original =", original)`	
`print("kopie =", kopie)`	

Aufgaben mit Funktionen lösen

Die Konstruktionsanleitung hilft dabei:

1. Kurzbeschreibung
2. Datenanalyse
3. Funktion definieren: Name – Input: Datentyp – Output: Datentyp
4. Funktions-Rumpf
5. Ergebnisse prüfen
6. Unittest

Aufgabe: Sätze bauen

Gegeben sind drei Listen:

subjekt = ["Der Hund", "Die Journalistin", "Der Maler"]
prädikat = ["vergräbt", "interviewt", "malt"]
objekt = ["den Knochen", "den Bürgermeister", "ein Bild"]

Schreibe ein Programm, das ein zufälliges Subjekt und ein zufälliges Prädikat und ein zufälliges Objekt hintereinanderstellt und den zufälligen Satz ausgibt.

Starte das Programm und beurteile die print-Ausgabe.

Nr	Programmzeile
1	`# Das Programm soll Subjekt, Prädikat, Objekt aus Listen`
	`# zufällig auswählen und einen Satz bauen`
	`# Bibliothek importieren`
	`import random`
	`# Beispielsätze`
	`subjekt = ["Der Hund", "Die Journalistin", "Der Maler"]`
	`prädikat = ["vergräbt", "interviewt", "malt"]`
	`objekt = ["den Knochen", "den Bürgermeister", "ein Bild"]`
2	`# Input der Funktion sind die Listen Subjekt, Prädikat und Objekt`
	`# Output der Funktion ist der Satz`
	`# Funktion mit Datentyp`
3	`def bau_den_satz(subjekt: list[str], prädikat: list[str], \`
	`objekt: list[str]) -> str:`
4	`    mein_subjekt = random.choice(subjekt)`
	`    mein_prädikat = random.choice(prädikat)`
	`    mein_objekt = random.choice(objekt)`
	`    mein_satz = mein_subjekt + " " + mein_prädikat + " " + mein_objekt`
	`    return mein_satz`
	`# Ergebnisse prüfen`
5	`for i in range(3):`
	`    # Funktion aufrufen`
	`    mein_satz = bau_den_satz(subjekt, prädikat, objekt)`
	`    # Ausgabe`
	`    print(mein_satz)`

Aufgaben mit Funktionen lösen – Fortsetzung

Aufgabe: Tabelle drucken

Wir wollen in Großbritannien einkaufen. Die Preise sind dort in britischen Pfund (GBP) angegeben. Wir müssen also umrechnen.

Schreibe ein Programm, das eine Umrechnungstabelle GBP in EUR druckt. Der Kurs ist: 1 GBP = 1,21 EUR

Die Tabelle soll von 0 GBP bis 10 GBP in Schritten von 0.50 GBP gehen.

Vervollständige das Programm.

Starte das Programm und beurteile die print-Ausgabe.

Nr	Programmzeile
1	`# Das Programm soll GPB in EUR umrechnen und eine Tabelle ausgeben`
2	`# Input der Funktion ist eine Dezimalzahl in der Einheit GBP`
	`# Output der Funktion ist eine Dezimalzahl in der Einheit EUR`
	`# Umrechnungsfaktor 1 GBP = 1.21 EUR`
	`# Funktion mit Datentyp`
3	Definiere die Funktion
4	Schreibe den Funktions-Rumpf
	`# Input Liste anlegen`
	`gbp_liste = []`
	`for i in range(21):`
	`    gbp_liste.append(i * 0.5)`
	`# Output Liste initialisieren`
	`eur_liste = []`
	`# Funktion aufrufen`
5	`for x in gbp_liste:`
	`    eur_liste.append(`rufe die Funktion auf`)`
	`# Ergebnisse drucken`
	`print("gbp    eur")`
	`for x, y in zip(gbp_liste, eur_liste):`
	`    print(x, y)`

In der for-Schleife liefert zip(gbp_liste, eur_liste) ein Tupel mit einem Element aus jeder Liste.