# **Programmieren lernen**

# **mit Python IDLE**



# Arbeitsblätter zum Kurs

### Inhalt

[Programmzeilen mit der IDLE Shell testen 2](#_Toc200568168)

[Mit der IDLE Shell rechnen 3](#_Toc200568169)

[Mit dem IDLE Editor ein Programm schreiben 4](#_Toc200568170)

[Aufgabe: Ziffern mit Punkten darstellen 5](#_Toc200568171)

[Eine Liste speichert viele änderbare Elemente 6](#_Toc200568172)

[Ein Tupel speichert viele nicht änderbare Elemente 7](#_Toc200568173)

[Ein Dictionary speichert Paare von Schlüssel und Wert 8](#_Toc200568174)

[Der Computer fragt … 9](#_Toc200568175)

[Der Computer unterscheidet zwischen "wahr" und "falsch" 10](#_Toc200568176)

[Der Computer unterscheidet Fälle 11](#_Toc200568177)

[Der Computer dreht Schleifen 12](#_Toc200568178)

[Aufgabe: Dialog mit dem Benutzer 13](#_Toc200568179)

[Funktionen haben Input und Output: Ganzzahlen 14](#_Toc200568180)

[Funktionen haben Input und Output: Tupel, Liste 15](#_Toc200568181)

[Aufgaben mit Funktionen lösen: Konstruktionsanleitung 16](#_Toc200568182)

[Aufgaben mit Funktionen lösen: Tabelle drucken 17](#_Toc200568183)

[Aufgaben mit Funktionen lösen: Diagramm plotten 18](#_Toc200568184)

[Klassen haben Eigenschaften und Methoden 19](#_Toc200568185)

[Die Eltern-Klasse vererbt – die Kind-Klasse erbt 20](#_Toc200568186)

[Roboter im Irrgarten: Wände bauen 21](#_Toc200568187)

[Roboter im Irrgarten: Wände ablegen 22](#_Toc200568188)

[Roboter im Irrgarten: Wände aus Datei lesen 23](#_Toc200568189)

[Roboter im Irrgarten: Schildkröte bewegen 24](#_Toc200568190)

[Roboter im Irrgarten: Schauen, gehen, drehen 25](#_Toc200568191)

[Roboter im Irrgarten: Rechte-Hand-Methode 27](#_Toc200568192)

[Roboter im Irrgarten: Pledge-Algorithmus 28](#_Toc200568193)

[Auto im Gegenverkehr: Auto steuern 29](#_Toc200568194)

[Auto im Gegenverkehr: Gegenverkehr kommt 31](#_Toc200568195)

[Auto im Gegenverkehr: Zusammenstoß melden 32](#_Toc200568196)

[Auto im Gegenverkehr: Gegenverkehr wird schneller 34](#_Toc200568197)

[Auto im Gegenverkehr: Straße und Sound 35](#_Toc200568198)

[Quellen 36](#_Toc200568199)

# 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:

1. Drucke eine Null
2. Drucke eine Acht
3. Drucke eine Eins
4. 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.

1. Lege das Dictionary englisch\_deutsch an und fülle es mit 3 Paaren.
2. Fordere ein neues englisches Wort an – den Schlüssel.
3. Fordere das passende deutsche Wort an – den Wert.
4. Erweitere das Dictionary mit Schlüssel und Wert.
5. 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.

1. Der Computer soll solange fragen, bis der Benutzer "e" eingibt.
2. Jede Benutzereingabe soll gedruckt werden.

### Die Mitte zwischen zwei Zahlen berechnen

100

Mitte

50

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

1. untere = 50 obere = 100
2. Berechne und drucke die ganzzahlige Mitte zwischen "untere" und "obere"
3. Teste deinen Ausdruck mit neuen Grenzen

### Der Computer reagiert auf weitere Benutzereingaben

Erweitere das Programm.

1. Der Computer soll solange fragen, bis der Benutzer "e" eingibt.
2. Wenn der Benutzer "k" eingibt, soll "kleiner" gedruckt werden.
3. Wenn der Benutzer "g" eingibt, soll "größer" gedruckt werden.

# Funktionen haben Input und Output: Ganzzahlen

### 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 haben Input und Output: Tupel, Liste

### 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: Konstruktionsanleitung

### 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 | |
|  | # Funktion aufrufen | |
| 5 | for i in range(3): | |
| mein\_satz = bau\_den\_satz(subjekt, prädikat, objekt) | |
| # Ergebnis drucken | |
| print(mein\_satz) | |

# Aufgaben mit Funktionen lösen: Tabelle drucken

### 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 | Programmiere den Funktions-Rumpf |
|  |
|  | # Input Liste anlegen |
|  | gbp\_liste = [] |
|  | for i in range(21): |
|  | gbp\_liste.append(i \* 0.5) |
|  | # Output Liste (leer) anlegen |
|  | 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.

# Aufgaben mit Funktionen lösen: Diagramm plotten

### Aufgabe: Tabelle drucken und Diagramm plotten

Unsere Stromkosten sind hoch. Deshalb denken wir über einen Wechsel des Stromanbieters nach.

|  |  |  |
| --- | --- | --- |
| Angebot | Grundgebühr pro Monat | Verbrauchspreis pro kWh |
| Stromtarif "Watt für wenig" | 15,60 € | 0,32 € |
| Stromtarif "Billig Strom" | 12,80 € | 0,36 € |

Welches Angebot ist günstiger? Das hängt von unserem monatlichen Stromverbrauch ab.

Schreibe ein Programm, das eine Vergleichstabelle druckt. Die Überschrift ist:   
Verbrauch Watt für wenig Billig Strom

Darunter stehen der monatliche Verbrauch und die berechneten monatlichen Kosten der beiden Angebote.   
Der Verbrauch geht von 0 kWh bis 150 kWh in Schritten von 10 kWh. Zeige den Vergleich auch in einem Diagramm.

Vervollständige das Programm. Starte das Programm und beurteile die print-Ausgabe.

|  |  |
| --- | --- |
| **Nr** | **Programmzeile** |
| 1 | # Programm vergleicht die Kosten von zwei Stromtarifen |
| 2 | # 1. Angebot: Input der Funktion ist eine Dezimalzahl in der Einheit kWh |
| # Output der Funktion ist eine Dezimalzahl in der Einheit EUR |
|  | # Funktion mit Datentyp |
| 3 | Definiere die Funktion |
| 4 | Programmiere den Funktions-Rumpf |
|  |
| 2 | # 2. Angebot: Input der Funktion ist eine Dezimalzahl in der Einheit kWh |
| # Output der Funktion ist eine Dezimalzahl in der Einheit EUR |
|  | # Funktion mit Datentyp |
| 3 | Definiere die Funktion |
| 4 | Programmiere den Funktions-Rumpf |
|  |
|  | # Input Liste anlegen |
|  | Lege die Liste an |
|  | # 1. Angebot: Output Liste (leer) anlegen |
|  | Lege die Liste an |
|  | # 2. Angebot: Output Liste (leer) anlegen |
|  | Lege die Liste an |
|  | # Funktionen aufrufen |
| 5 | Programmiere eine Schleife |
| Fülle die Liste |
| Fülle die Liste |
| # Ergebnisse drucken |
| Drucke die Überschrift |
| Programmiere eine Schleife |
| Drucke das Ergebnis |
|  | # Modul für das Plotten von Graphen importieren |
|  | import matplotlib.pyplot as plt |
|  | # Ergebnisse plotten |
| 5 | plt.plot(verbrauch, kosten1) |
| plt.plot(verbrauch, kosten2) |
| plt.xlabel("Verbrauch") |
| plt.ylabel("monatliche Kosten") |
| plt.show() |

# Klassen haben Eigenschaften und Methoden

### Eine Instanz der Klasse beschreibt ein konkretes Objekt mit Eigenschaften und Methoden

### Aufgabe: Eine Klasse und eine Instanz programmieren

Programmiere die Klasse "Fahrrad" mit den Eigenschaften:   
Besitzer, Farbe, Typ (Touring, Renn, Mountain), Anzahl Gänge

Programmiere die Methoden:   
klingeln, fahren

Erstelle zwei Instanzen der Klasse "Fahrrad" und gib die Eigenschaften aus.   
Rufe dann die Methoden der Instanzen auf.

Das Programm Katzen\_Klasse.py löst eine ganz ähnliche Aufgabe.   
Schreibe das Programm Fahrrad\_Klasse.py nach dem Vorbild Katzen\_Klasse.py.

Die Vergleichstabelle hilft dabei.

Starte das Programm und beurteile die print-Ausgabe.

|  |  |
| --- | --- |
| **Katzen\_Klasse.py** | **Fahrrad\_Klasse.py** |
| BauplanKatzenKlasse | BauplanFahrradKlasse |
| rufname | besitzer |
| farbe | farbe |
|  | typ |
| alter | gaenge |
| schlafdauer | kmstand |
| tut\_miauen | klingeln |
| tut\_schlafen | fahren |
| katze\_sammy | mein\_fahrrad |
| katze\_sammy.rufname | mein\_fahrrad.besitzer |
| katze\_sammy.farbe | mein\_fahrrad.farbe |
|  | mein\_fahrrad.typ |
| katze\_sammy.alter | mein\_fahrrad.gaenge |
| katze\_soni | leih\_fahrrad |
| katze\_sammy.tut\_miauen | mein\_fahrrad.klingeln |
| katze\_sammy.tut\_schlafen | mein\_fahrrad.fahren |
| katze\_soni.tut\_schlafen | leih\_fahrrad.fahren |

# Die Eltern-Klasse vererbt – die Kind-Klasse erbt

## Die Kind-Klasse erbt alle Eigenschaften und Methoden der Eltern-Klasse.

### Aufgabe: Eine Eltern-Klasse und zwei Kind-Klassen programmieren

Erstelle die Eltern-Klasse "Zweirad" mit den Eigenschaften: Besitzer, Farbe, Typ, Anzahl Gänge   
und den Methoden: fahren, klingeln.

Erstelle die Kind-Klassen "Fahrrad" und "Pedelec", die alle Eigenschaften und Methoden   
der Eltern-Klasse "Zweirad" erben.

Die Klasse "Pedelec" hat zusätzlich die Eigenschaft "Kapazität" (Wattstunden) und die Methode "aufladen".

Erstelle eine Instanz der Klasse "Fahrrad" und eine Instanz der Klasse "Pedelec" und rufe alle Methoden auf.   
Gib die Eigenschaft "Kapazität" der Instanz des "Pedelec" aus.

Das Programm Tier\_Klasse.py löst eine ganz ähnliche Aufgabe.   
Schreibe das Programm Zweirad\_Klasse.py nach dem Vorbild Tier\_Klasse.py.

Fülle zuerst die rechte Spalte der Vergleichstabelle aus.

Starte das Programm und beurteile die print-Ausgabe.

|  |  |
| --- | --- |
| **Tier\_Klasse.py** | **Zweirad\_Klasse.py** |
| class Tier() |  |
| rufname |  |
| farbe |  |
| rechts steht 1 Eigenschaft mehr |  |
| alter |  |
| schlafdauer |  |
| tut\_reden |  |
| tut\_schlafen |  |
| class BauplanKatzenKlasse(Tier) |  |
| rufname |  |
| farbe |  |
| rechts steht 1 Eigenschaft mehr |  |
| alter |  |
| class Hund(Tier) |  |
| rufname |  |
| farbe |  |
| rechts steht 1 Eigenschaft mehr |  |
| alter |  |
| rechts steht die zusätzliche Eigenschaft |  |
| rechts steht die zusätzliche Methode |  |
| katze\_sammy |  |
| katze\_sammy.farbe |  |
| hund\_bello |  |
| hund\_bello.farbe |  |
| hund\_bello.tut\_schlafen |  |
| katze\_sammy.tut\_schlafen |  |
| katze\_sammy.tut\_reden |  |
| hund\_bello.tut\_reden |  |
| rechts steht der Aufruf der zusätzlichen Methode |  |
| rechts steht die Ausgabe der zusätzlichen Eigenschaft |  |

# Roboter im Irrgarten: Wände bauen

## Wir arbeiten mit der grafischen Benutzeroberfläche "Turtle"

### Aufgabe: Einen Irrgarten bauen – Irrgarten\_Klasse.py

Das Programm bringt ein Fenster mit 3 Blöcken unterschiedlicher Farbe auf den Bildschirm. Starte das Programm. Ändere das Programm und starte es erneut.

|  |
| --- |
| **Aufgabe** |
| Setze Blöcke nebeneinander und untereinander, um Wände zu bauen. Experimentiere mit den Farben. |





# Roboter im Irrgarten: Wände ablegen

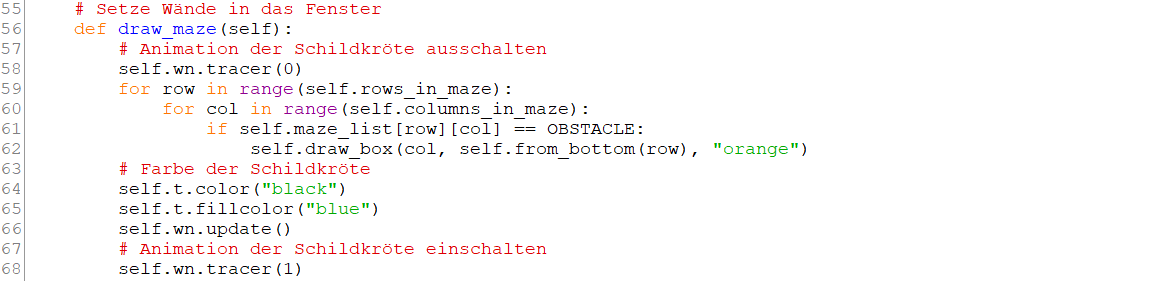
### Aufgabe: Wände des Irrgartens in einer Liste ablegen - Irrgarten\_Klasse\_maze\_list.py

Überall wo ein '+' in der Liste steht, zeichnet das Programm einen Block in das Fenster. In der Mitte des Fensters formen die Wände ein Wort. Starte das Programm. Ändere das Programm und starte es erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Ersetze in Zeile 62 self.from\_bottom(row) durch row |  |
| Kommentiere die Zeile 58 aus |  |







# Roboter im Irrgarten: Wände aus Datei lesen

### Aufgabe: Wände des Irrgartens aus einer Datei lesen - Irrgarten\_Klasse\_maze\_filename.py

Das Programm öffnet die Datei maze3.txt und kopiert alle Zeilen in der Datei in die Variable lines.   
Dann geht das Programm durch jede Zeile von lines und erzeugt eine Liste mit allen Zeichen der Zeile.   
Diese Liste wird an maze\_list angehängt. Starte das Programm.   
Ändere die Datei maze3.txt und starte das Programm erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Vergleiche die '+' in maze3.txt mit den Blöcken im Fenster |  |
| Ergänze und entferne '+' in maze3.txt und vergleiche erneut |  |

Inhalt der Datei maze3.txt

++++++++++++++++++++++

+ ++ ++ + +

+++++ ++ + +++ +

+ + + + +

+ +++++ + + +++ +

+ + + +++ +

+ +++++++ + +

+ + +++++ +

+ ++++++++++ + + +

+ S + +

++++++++++++++++++++++





# Roboter im Irrgarten: Schildkröte bewegen

### Aufgabe: Schildkröte durch den Irrgarten bewegen - Irrgarten\_Klasse\_move\_turtle.py

Nachdem das Programm die Liste maze\_list erstellt hat, sucht es in maze\_list nach dem Startpunkt "S".   
Die Funktion search\_from() setzt die Schildkröte auf den Startpunkt. Starte das Programm. Ändere das Programm und starte es erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Vergleiche das 'S' in maze3.txt mit der Position der Schildkröte |  |
| Schreibe hinter die Zeile 89:  maze.update\_position(start\_row - 2, start\_col) |  |





# Roboter im Irrgarten: Schauen, gehen, drehen

### Aufgabe: Roboter schaut, geht und dreht - Irrgarten\_Klasse\_robot\_methods.py

Das Programm kann die Schildkröte bewegen. Die Schildkröte erkennt aber keine Wände. Wir benötigen also einen Roboter, der schaut, geht und dreht. Die Position des Roboters wird durch seine row und col dargestellt.

|  |  |
| --- | --- |
| left | up  right |
|  | down |

Der Roboter soll Folgendes können:

* in alle 4 Richtungen schauen
* in alle 4 Richtungen gehen
* nach rechts drehen
* nach links drehen

Rechts und links sind abhängig von der aktuellen Richtung. Ein Dictionary gibt uns die neue Richtung für rechts und für links. Beispiel: Die aktuelle Richtung ist "down". Rechts ist "left".

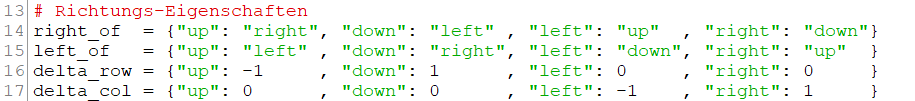
In eine Richtung schauen oder gehen bedeutet:

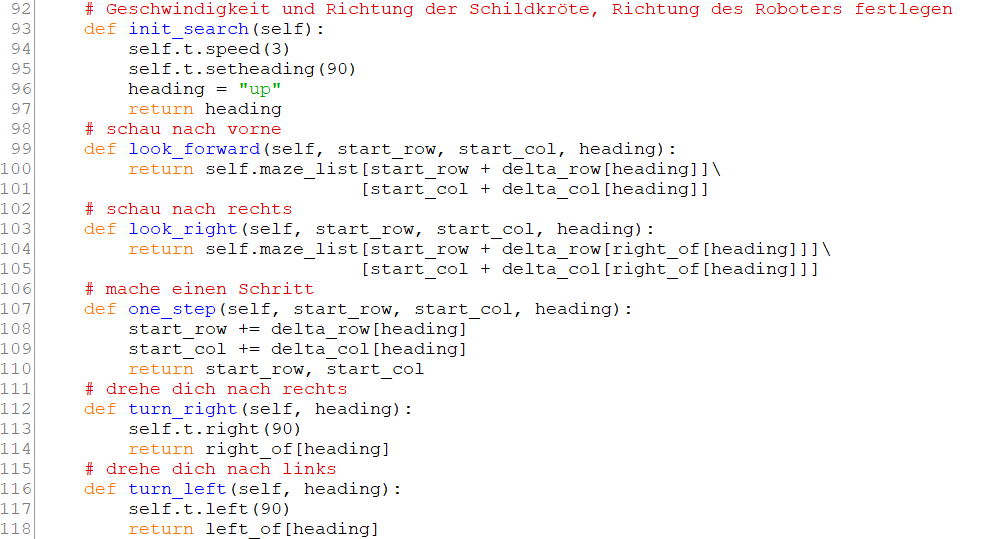
* row unverändert oder row + 1 oder row – 1 *und*
* col unverändert oder col + 1 oder col - 1

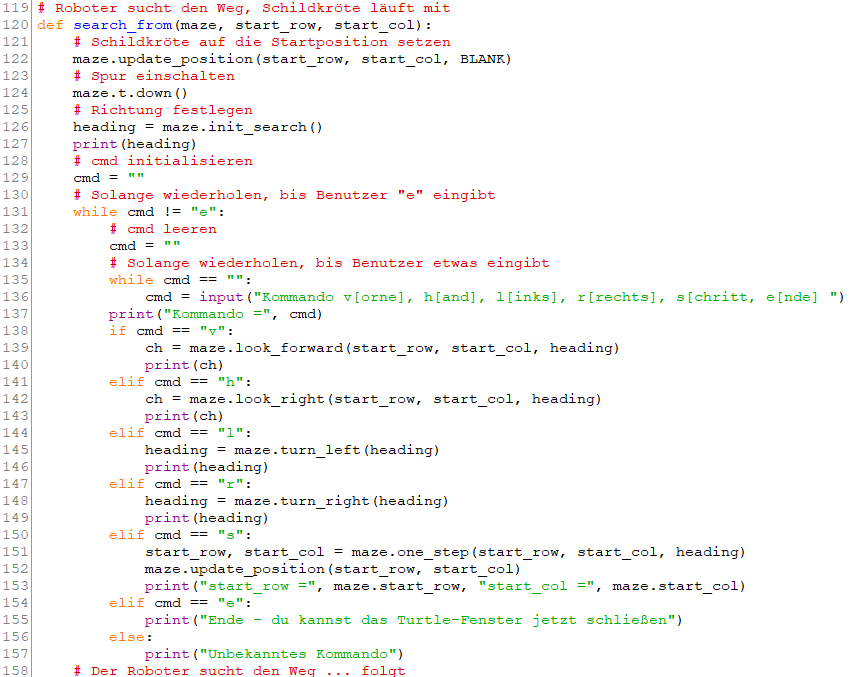
Ein Dictionary gibt uns das für row und col vor. Beispiel: "left" bedeutet row unverändert *und* col – 1.

Die Klasse Maze bekommt zusätzlich 6 Roboter-Methoden. Wir testen die Methoden mit Hilfe der Funktion search\_from(). Die Funktion enthält eine Benutzerabfrage. Dort gebt ihr ein Kommando ein, und der Roboter schaut, geht und dreht. Starte das Programm. Gib Kommandos ein und beobachte die Reaktion von Roboter und Schildkröte. Tipp: Das Kommando h[and] steht für: Schaue zur rechten Hand.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Gib die Kommandos v[orne], h[and], l[inks], r[rechts], s[chritt, e[nde] ein und bewege den Roboter eine kurze Strecke durch den Irrgarten. Kann der Roboter durch Wände gehen? |  |



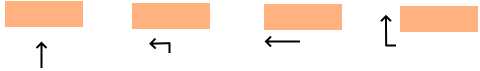




# Roboter im Irrgarten: Rechte-Hand-Methode

### Aufgabe: Mit der Rechte-Hand-Methode findet der Roboter aus dem Irrgarten - turtle\_in\_maze\_right\_hand\_rule\_2x1.py

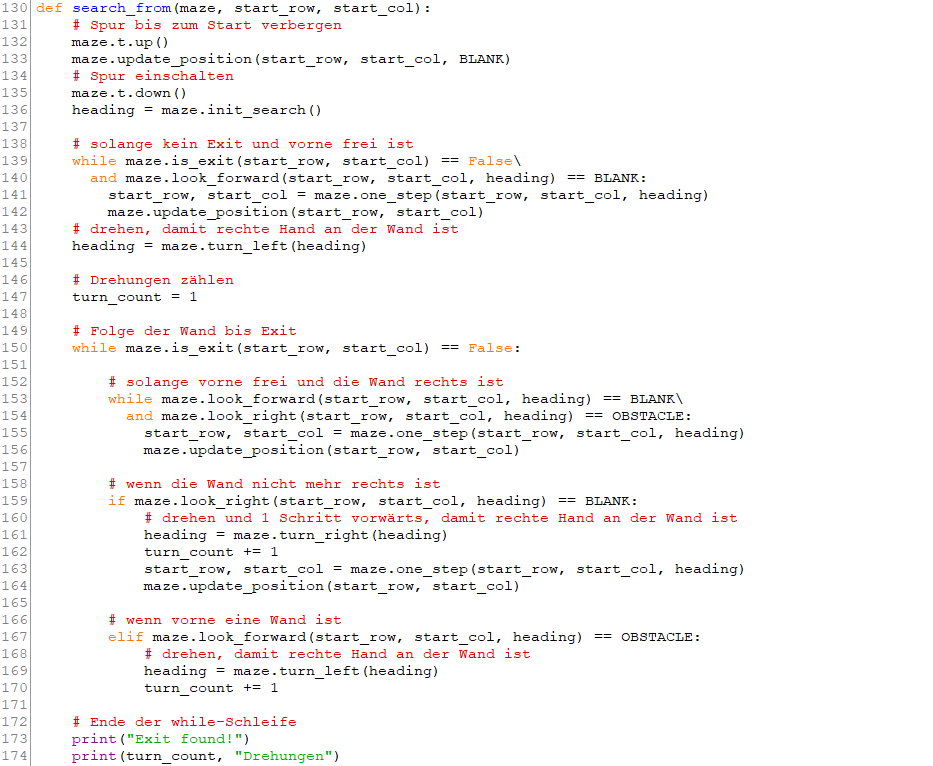
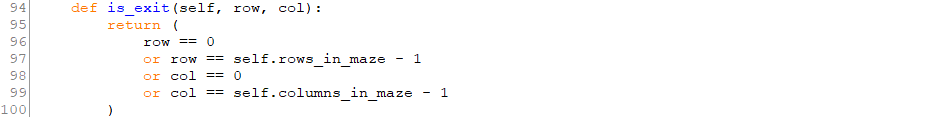
Die Methoden look\_forward() und look\_right()melden eine der folgenden Situationen.   
vorne frei vorne Wand Wand rechts Wand nicht rechts



Der Pfeil zeigt, wie der Roboter reagieren soll, damit seine rechte Hand an der Wand bleibt.

Der Roboter hat den Ausgang erreicht, wenn die Methode is\_exit() True meldet. Starte das Programm.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Ersetze nun in maze3.txt Zeile 8 Spalte 17 das + durch ein Leerzeichen. Findet der Roboter den Weg aus dem Irrgarten?  Wie muss der Irrgarten beschaffen sein, damit der Roboter hinausfindet? |  |

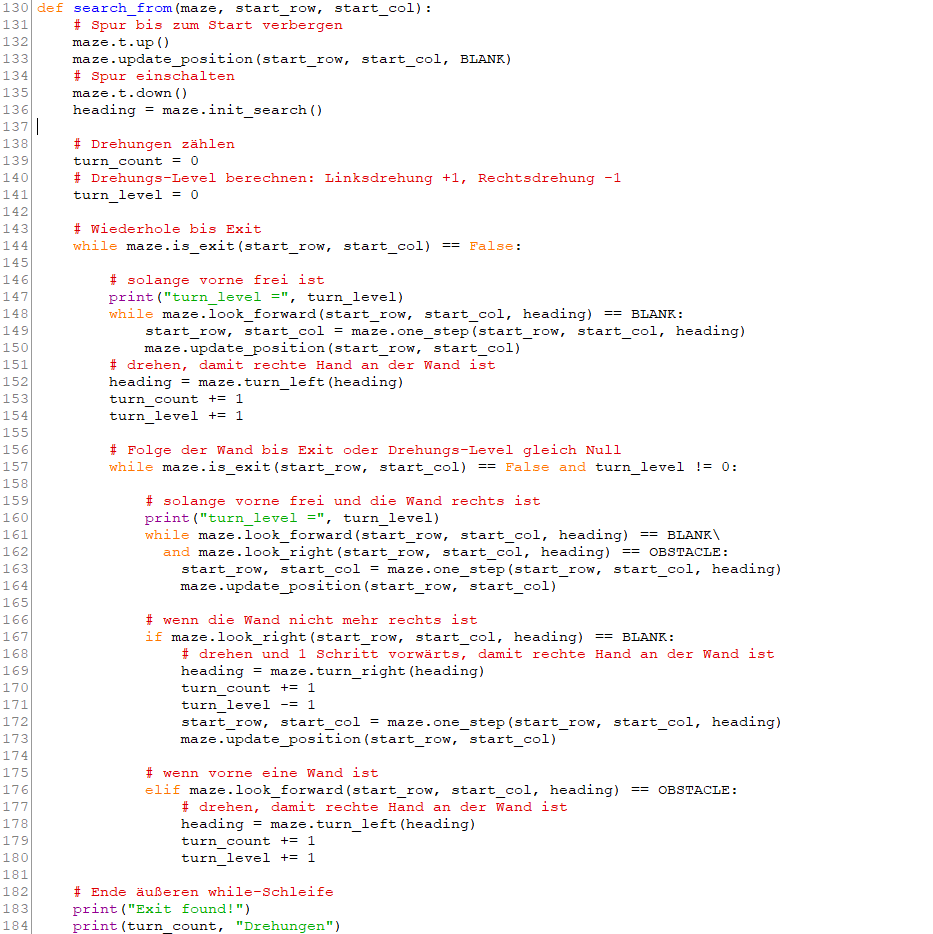


# Roboter im Irrgarten: Pledge-Algorithmus

### Aufgabe: Mit dem Pledge-Algorithmus findet der Roboter aus jedem Irrgarten - turtle\_in\_maze\_pledge\_algorithm\_2x1.py

Der Pledge-Algorithmus arbeitet mit dem Drehungs-Level turn\_level. Zu Beginn ist der Drehungs-Level Null.   
Bei einer Linksdrehung wird er um 1 erhöht, bei einer Rechtsdrehung um 1 erniedrigt. Der Roboter läuft geradeaus bis zur Wand und macht eine Linksdrehung, damit die rechte Hand an die Wand ist. Danach folgt er der Wand bis zum Ausgang oder bis der Drehungs-Level Null ist. Wenn der Drehungs-Level Null ist, läuft der Roboter wieder geradeaus bis zur Wand, ohne auf die rechte Hand zu achten. Starte das Programm.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Ersetze nun in maze3.txt Zeile 8 Spalte 17 das + durch ein Leerzeichen. Findet der Roboter den Weg aus dem geänderten Irrgarten maze3.txt? |  |
| Teste das Programm mit dem Irrgarten my\_maze.txt |  |



# Auto im Gegenverkehr: Auto steuern

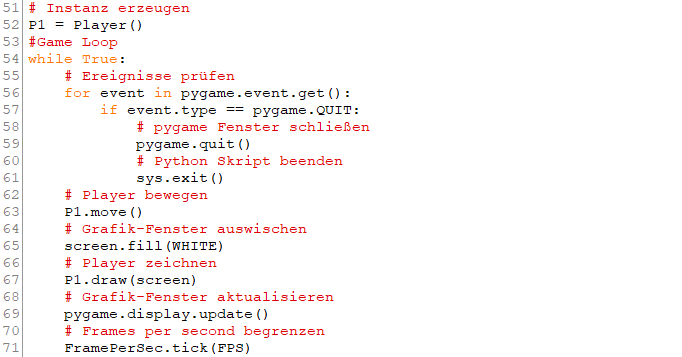
## Wir arbeiten mit der Bibliothek "Pygame" für Computerspiele

### Aufgabe: Wir steuern das Auto nach links – nach rechts – vorwärts – rückwärts - Spieler\_Klasse\_Auto\_steuern.py

Das Programm bringt ein Fenster mit einem blauen Auto auf den Bildschirm. Starte das Programm. Ändere das Programm und starte es erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Steuere das Auto mit den Pfeiltasten der Tastatur. Warum verlässt das Auto das Fenster nicht? |  |
| Ersetze das blaue Auto durch das größere gelbe Auto. Verlässt das gelbe Auto das Fenster? |  |



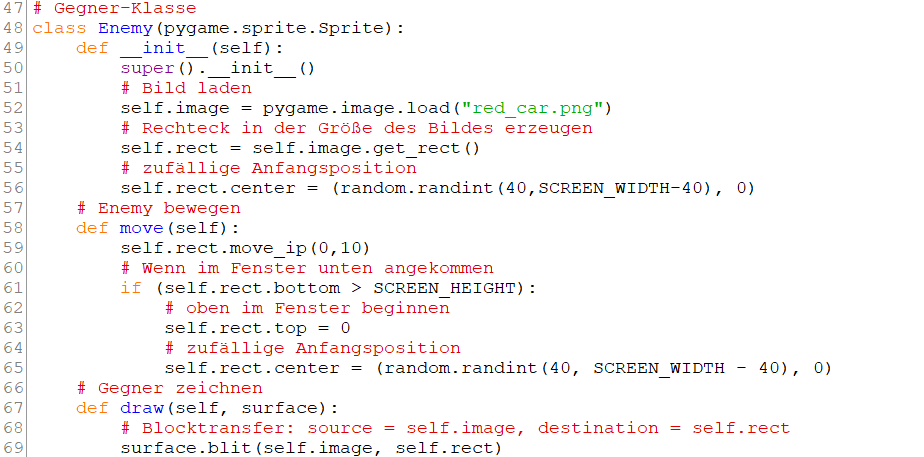


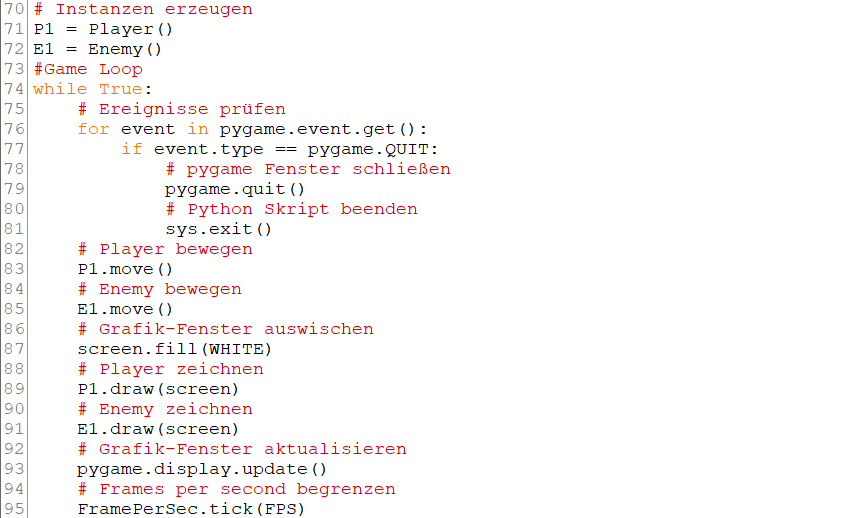
# Auto im Gegenverkehr: Gegenverkehr kommt

### Aufgabe: Ein Auto kommt entgegen – Spieler\_Klasse\_Gegner\_Klasse\_Gegenverkehr.py

Das Programm bringt ein Fenster mit einem blauen Auto *und einem entgegenkommenden roten Auto* auf den Bildschirm. Starte das Programm. Ändere das Programm und starte es erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Steuere das Auto mit den Links- und Rechts-Pfeiltasten der Tastatur. Was passiert bei einem Zusammenstoß der beiden Autos? |  |
| Verdopple die Geschwindigkeit des roten Autos. Die Geschwindigkeit des blauen Autos soll dabei unverändert bleiben. |  |



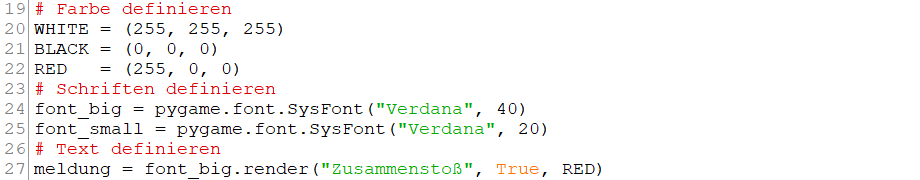


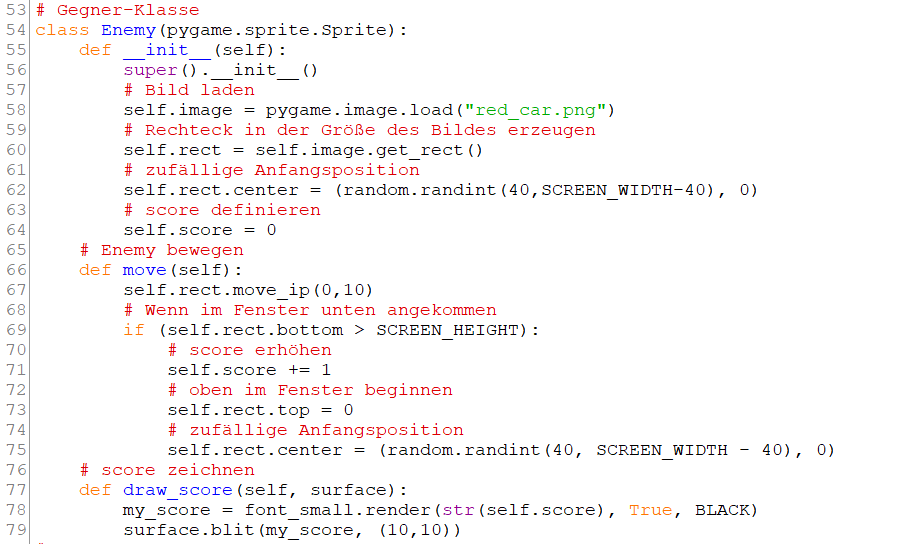
# Auto im Gegenverkehr: Zusammenstoß melden

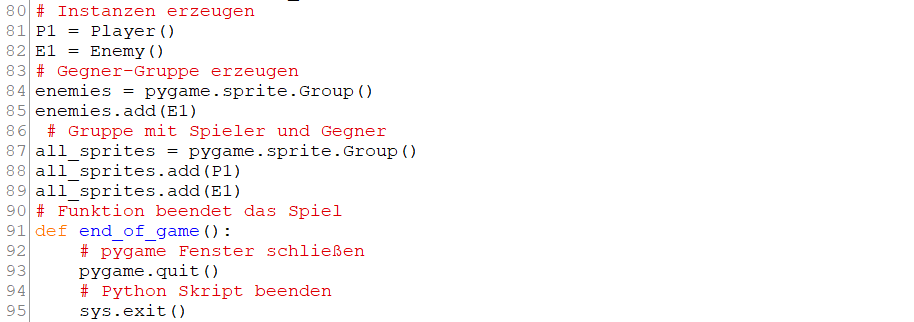
### Aufgabe: Zusammenstoß melden - Spieler\_Klasse\_Gegner\_Klasse\_Zusammenstoss.py

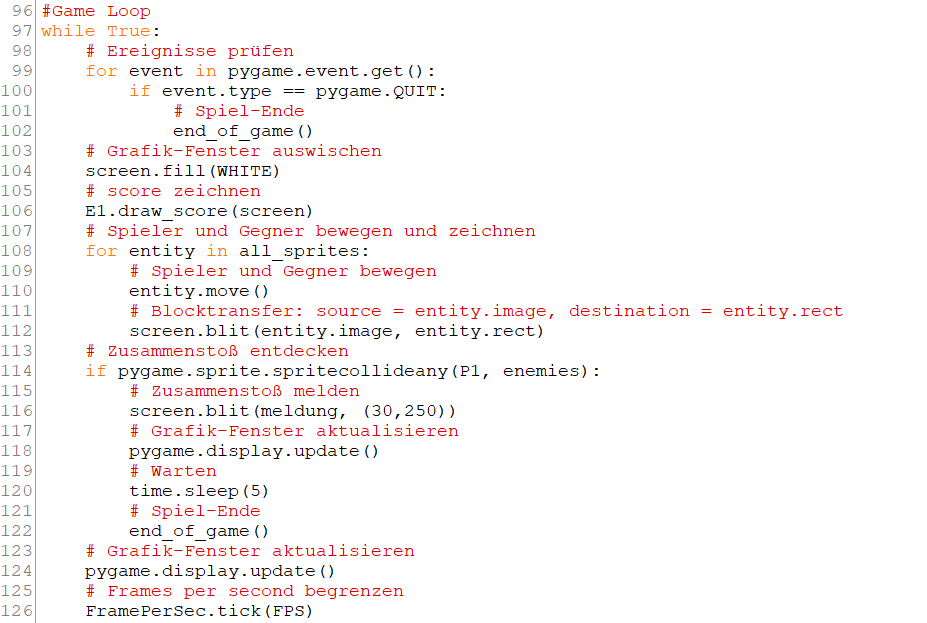
Das Programm bringt ein Fenster mit einem blauen Auto und einem entgegenkommenden roten Auto auf den Bildschirm. *Jedes erfolgreiche Ausweichen des Spielers wird gezählt. Bei einem Zusammenstoß gibt es eine Meldung und das Programm endet.* Starte das Programm. Ändere das Programm und starte es erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Steuere das Auto mit den Links- und Rechts-Pfeiltasten der Tastatur. Was passiert bei einem Zusammenstoß der beiden Autos? |  |
| Erzeuge eine weitere Instanz der Gegner-Klasse. Erweitere die Gruppen enemies und all\_sprites, sodass zwei rote Autos entgegenkommen. |  |



****



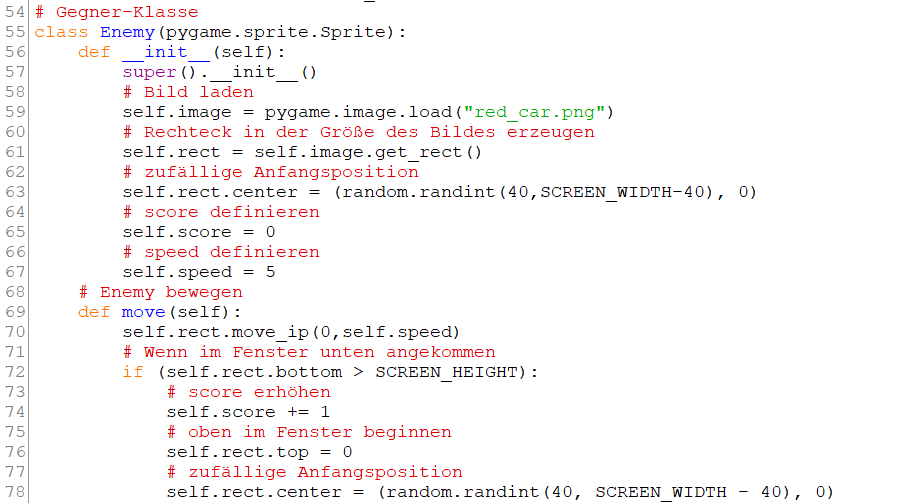


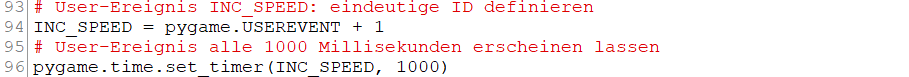
# Auto im Gegenverkehr: Gegenverkehr wird schneller

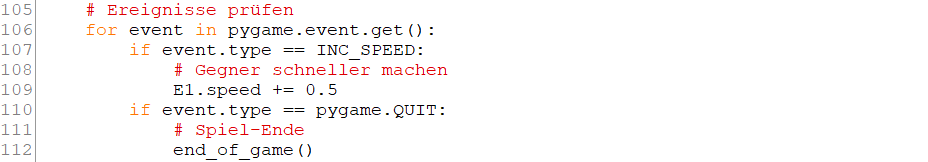
### Aufgabe: Gegenverkehr wird schneller werden - Spieler\_Klasse\_Gegner\_Klasse\_schneller.py

Das Programm bringt ein Fenster mit einem blauen Auto und einem entgegenkommenden roten Auto auf den Bildschirm. Jedes erfolgreiche Ausweichen des Spielers wird gezählt. Bei einem Zusammenstoß gibt es eine Meldung und das Programm endet. *Das rote Auto wird mit jeder Sekunde schneller.* Starte das Programm. Ändere das Programm und starte es erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Steuere das Auto mit den Links- und Rechts-Pfeiltasten der Tastatur. Mache 4 Läufe und notiere die Scores. |  |
| Nun soll das Auto jede Sekunde die Geschwindigkeit wechseln.  Die Geschwindigkeit ist ein Zufallswert zwischen 5 und 20.  Mache 4 Läufe und notiere die Scores. |  |







# Auto im Gegenverkehr: Straße und Sound

### Aufgabe: Straße im Hintergrund, Sound beim Zusammenstoß – Traffic\_Game.py

Das Programm bringt ein Fenster mit einem blauen Auto und einem entgegenkommenden roten Auto auf einer Straße auf den Bildschirm. Jedes erfolgreiche Ausweichen des Spielers wird gezählt. Bei einem Zusammenstoß gibt es eine Meldung und einen Sound und das Programm endet. Das rote Auto wird mit jeder Sekunde schneller. Starte das Programm. Ändere das Programm und starte es erneut.

|  |  |
| --- | --- |
| **Aufgabe** | **Ergebnis** |
| Steuere das Auto mit den Links- und Rechts-Pfeiltasten der Tastatur. Mache 4 Läufe und notiere die Scores. |  |
| Ändere die Spiel-Idee: Ein Dinosaurier überquert die Straße.  Das rote Auto bremst rechtzeitig.   * Ersetze die Meldung "Zusammenstoß" durch "Achtung" * Ersetze "blue\_car.png" durch dinosaur\_LR.png * Ersetze den "crash.wav" durch "car\_screech.wav" |  |
| Der Dinosaurier läuft rückwärts. Der Dinosaurier soll sich drehen, wenn die Pfeiltasten gewechselt werden.   * Lade zwei Dinosaurier-Bilder:  self.image\_LR = pygame.image.load("dinosaur\_LR.png") self.image\_RL = pygame.image.load("dinosaur\_RL.png") * Definiere das Anfangsbild: self.image = self.image\_LR * Wechsle die Bilder abhängig von der gedrückten Pfeiltaste: self.image = self.image\_RL – und – self.image = self.image\_LR |  |
| Die Straße hat keinen Gehweg, auf dem der Dinosaurier sicher ist.   * Setze SCREEN\_WIDTH = 600 * Ersetze "animated\_street.png" durch "animated\_street\_w\_sidewalk.png" * Zu Beginn steht der Dinosaurier auf dem Gehweg. Ersetze self.rect.center = (160, 520) durch self.rect.center = (60, 520) * Das rote Auto muss den Gehweg meiden.  self.rect.center = (random.randint(40,SCREEN\_WIDTH-40), 0) wird 2x im Programm zu: self.rect.center = (random.randint(150,SCREEN\_WIDTH-150), 0) |  |
| Der Score wird nicht richtig gezählt. Der Score soll erhöht werden, wenn der Dinosaurier den Gehweg erreicht hat.   * In der Gegner-Klasse gelöscht und in der Spieler-Klasse hinzugefügt: self.score = 0 * In der Gegner-Klasse gelöscht und in der Spieler-Klasse hinzugefügt: Die Funktion draw\_score() * Neue Bedingung in der Funktion move() der Spieler-Klasse: if (self.direction == "right") and (self.rect.left > SCREEN\_WIDTH-120):  self.score += 1  self.direction = "left" if (self.direction == "left") and (self.rect.right < 120):  self.score += 1 self.direction = "right" |  |

****





# Quellen

|  |  |
| --- | --- |
| **Quelle** | **Thema** |
| <https://docs.python.org/3/> | Python Grundlagen |
| <https://www.python-lernen.de/> | Python Grundlagen |
| Felleisen et al. (2013), Realm of Racket,  No Starch Press, San Francisco | Spiel: Computer errät die Zahl |
| <https://www.deinprogramm.de/> | Konstruktionsanleitung für Funktionen |
| <https://matplotlib.org/stable/tutorials/pyplot.html> | Bibliothek matplotlib |
| <https://docs.python.org/3/library/turtle.html> | Bibliothek turtle |
| <https://runestone.academy/ns/books/published/pythonds3/Recursion/ExploringaMaze.html> | Irrgarten und Turtle |
| <https://gist.github.com/maxrothman/92eb8470408a047b1f6815a4a444d727> | Irrgarten lösen |
| <https://algo.rwth-aachen.de/~algorithmus/algo6.php> | Pledge Algorithmus |
| <https://www.pygame.org/docs/index.html> | Bibliothek pygame |
| <https://coderslegacy.com/python/python-pygame-tutorial/> | Spiel: Traffic Game |