Programmieren lernen

mit Python IDLE

Arbeitsblätter zum Kurs

Simons CC BY-NC-SA 4.0

Inhalt

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

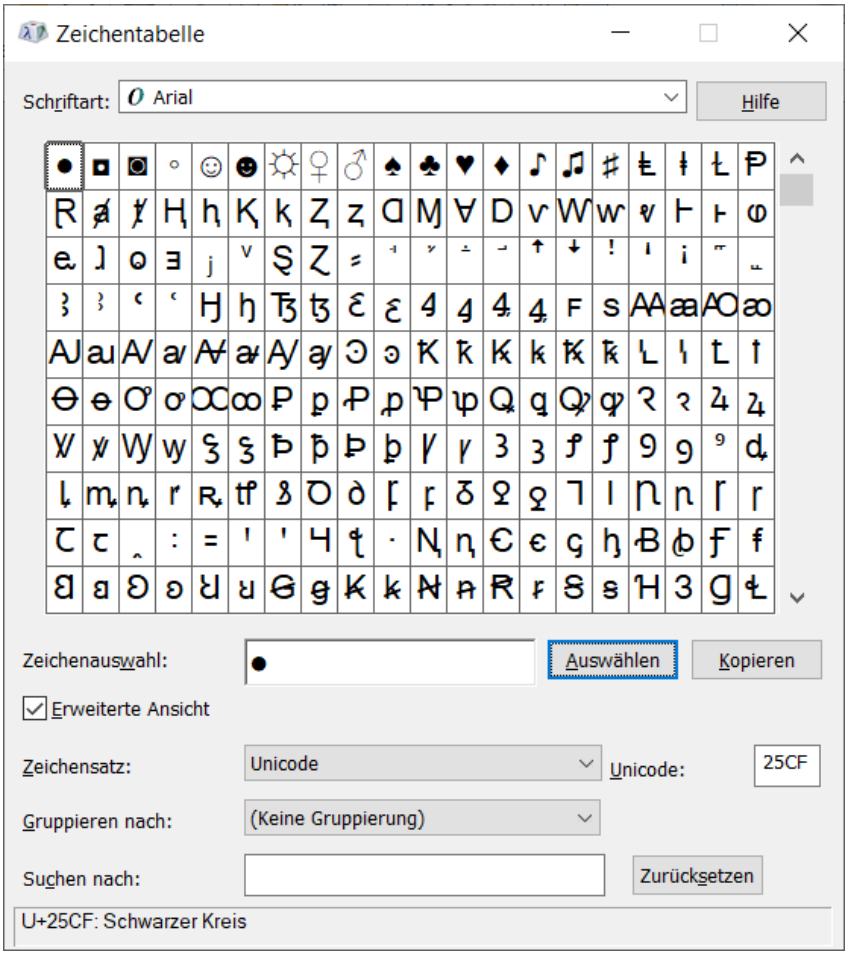
Die Ziffern auf einem Kassenzettel sind aus Punkten zusammengesetzt. Mit 5x7 Punkten können die Ziffern 0 bis 9 gut lesbar dargestellt werden.

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

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")`	

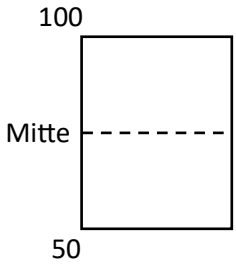
Aufgabe: Dialog mit dem Benutzer

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 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.

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

```
# Programm erzeugt ein Fenster und setzt die Wände eines Irrgartens hinein
# - Eine Funktion setzt die Wände in das Fenster
# Modul für die Turtle-Grafik importieren
import turtle
# Irrgarten-Klasse
class Maze:
    """Klasse für den Bau eines Irrgartens"""
    # Methoden der Klasse
    def __init__(self):
        self.rows_in_maze = 11
        self.columns_in_maze = 22
        # turtle Objekt erzeugen und Aussehen festlegen
        self.t = turtle.Turtle()
        self.t.shape("turtle")
        # Breite, Höhe und Koordinaten des Fensters festlegen
        self.wn = turtle.Screen()
        self.wn.setup(800, 400)
        self.wn.setworldcoordinates(0, 0, self.columns_in_maze, self.rows_in_maze)
    # Zeichne ein ausgefülltes Rechteck
    def draw_box(self, x, y, color):
        self.t.up() # Stift hoch
        self.t.goto(x, y)
        self.t.color(color)
        self.t.fillcolor(color)
        self.t.setheading(90)
        self.t.down() # Stift runter
        self.t.begin_fill()
        # Rechteck zeichnen und füllen
        for i in range(4):
            self.t.forward(1)
            self.t.right(90)
        self.t.end_fill()
    # Setze Wände in das Fenster
    def draw_maze(self):
        # Farben: 'white','black','red','green','blue','cyan','yellow','magenta'
        self.draw_box(0, 0, "orange")
        self.draw_box(self.columns_in_maze - 1, self.rows_in_maze - 1, "magenta")
        self.draw_box(self.columns_in_maze//2, self.rows_in_maze//2, "cyan")
        # Farbe der Schildkröte
        self.t.color("black")
        self.t.fillcolor("blue")
        self.wn.update()
# Instanz erzeugen
my_maze = Maze()
# Irrgarten zeichnen
my_maze.draw_maze()
my_maze.t.up() # Stift hoch
# Schildkröte in Home-Position
my_maze.t.home()
# Turtle event loop
my_maze.wn.mainloop()
```

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.

Aufgabe	**Ergebnis**
Ersetze in Zeile 62 `self.from_bottom(row)` durch `row`	
Kommentiere die Zeile 58 aus	

```
14          self.maze_list =\
15          [['+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+'],
16           ['+',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','+'],
17           ['+',' ',' ',' ',' ',' ','+',' ',' ',' ','+',' ','+','+','+','+',' ',' ',' ',' ',' ','+'],
18           ['+',' ',' ',' ',' ',' ','+',' ',' ',' ','+',' ','+',' ',' ',' ','+',' ',' ',' ',' ','+'],
19           ['+',' ',' ',' ',' ',' ','+',' ',' ',' ','+',' ','+',' ',' ',' ','+',' ',' ',' ',' ','+'],
20           ['+',' ',' ',' ',' ',' ','+',' ',' ',' ','+',' ','+','+','+','+',' ',' ',' ',' ',' ','+'],
21           ['+',' ',' ',' ',' ',' ','+',' ',' ',' ','+',' ','+',' ',' ',' ',' ',' ',' ',' ',' ','+'],
22           ['+',' ',' ',' ',' ',' ','+',' ',' ',' ','+',' ','+',' ',' ',' ',' ',' ',' ',' ',' ','+'],
23           ['+',' ',' ',' ',' ',' ',' ','+','+','+',' ',' ','+',' ',' ',' ',' ',' ',' ',' ',' ','+'],
24           ['+',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','+'],
25           ['+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+','+']]
26          self.rows_in_maze = len(self.maze_list)
27          self.columns_in_maze = len(self.maze_list[0])
28          print(self.rows_in_maze, self.columns_in_maze)
```

```
36      # Lies die Zeilennummer von unten
37      def from_bottom(self, row):
38          # Zeilen der Liste laufen von oben nach unten
39          # y Koordinaten des Fensters laufen von unten nach oben
40          return (self.rows_in_maze - 1) - row
```

```
55      # Setze Wände in das Fenster
56      def draw_maze(self):
57          # Animation der Schildkröte ausschalten
58          self.wn.tracer(0)
59          for row in range(self.rows_in_maze):
60              for col in range(self.columns_in_maze):
61                  if self.maze_list[row][col] == OBSTACLE:
62                      self.draw_box(col, self.from_bottom(row), "orange")
63          # Farbe der Schildkröte
64          self.t.color("black")
65          self.t.fillcolor("blue")
66          self.wn.update()
67          # Animation der Schildkröte einschalten
68          self.wn.tracer(1)
```

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   + +
++++++++++++++++++++++
```

```
10 # Irrgarten-Klasse
11 class Maze:
12     """Klasse für den Bau eines Irrgartens"""
13     # Methoden der Klasse
14     def __init__(self, maze_filename):
15         # maze_list aus Datei lesen
16         self.maze_list = []
17         with open(maze_filename, "r") as maze_file:
18             self.lines = maze_file.readlines()
19             for line in self.lines:
20                 self.maze_list.append([ch for ch in line.rstrip("\n")])
21         self.rows_in_maze = len(self.maze_list)
22         self.columns_in_maze = len(self.maze_list[0])
23         print("rows in maze =", self.rows in maze, "columns in maze =", self.columns in maze)
64 # Instanz erzeugen
65 my_maze = Maze("maze3.txt")
66 # Irrgarten zeichnen
67 my_maze.draw_maze()
68 my_maze.t.up() # Stift hoch
69 # Schildkröte in Home-Position
70 my_maze.t.home()
71 # Turtle event loop
72 my maze.wn.mainloop()
```

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.

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)`	

```
12 # Irrgarten-Klasse
13 class Maze:
14     """Klasse für den Bau eines Irrgartens"""
15     # Methoden der Klasse
16     def __init__(self, maze_filename):
17         # maze_list aus Datei lesen
18         self.maze_list = []
19         with open(maze_filename, "r") as maze_file:
20             self.lines = maze_file.readlines()
21             for line in self.lines:
22                 self.maze_list.append([ch for ch in line.rstrip("\n")])
23         self.rows_in_maze = len(self.maze_list)
24         self.columns_in_maze = len(self.maze_list[0])
25         print("rows_in_maze =", self.rows_in_maze, "columns_in_maze =", self.columns_in_maze)
26         # Start in maze_list suchen
27         for row in range(self.rows_in_maze):
28             for col in range(self.columns_in_maze):
29                 if self.maze_list[row][col] == START:
30                     self.start_row = row
31                     self.start_col = col
32                     break
33         print("start_row =", self.start_row, "start_col =", self.start_col)
 74     # Bewege die Schildkröte
 75     def move_turtle(self, x, y):
 76         x += 0.5
 77         y += 0.5
 78         self.t.setheading(self.t.towards(x, y))
 79         self.t.goto(x, y)
 80     # Aktualisiere die Position der Schildkröte
 81     def update_position(self, row, col, val=None):
 82         # Wenn val angegeben: Element der Liste überschreiben
 83         if val:
 84             self.maze_list[row][col] = val
 85         self.move_turtle(col, self.from_bottom(row))
 86 # Roboter sucht den Weg, Schildkröte läuft mit
 87 def search_from(maze, start_row, start_col):
 88     # Schildkröte auf die Startposition setzen
 89     maze.update_position(start_row, start_col, BLANK)
 90     # Der Roboter sucht den Weg ... folgt
 91 # Instanz erzeugen
 92 my_maze = Maze("maze3.txt")
 93 # Irrgarten zeichnen
 94 my_maze.draw_maze()
 95 my_maze.t.up() # Stift hoch
 96 # Schildkröte in Home-Position
 97 my_maze.t.home()
 98 # Suche den Weg vom Start zum Ausgang
 99 search_from(my_maze, my_maze.start_row, my_maze.start_col)
100 # Turtle event loop
101 my maze.wn.mainloop()
```

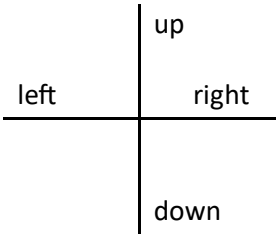
Roboter im Irrgarten: Schauen, gehen, drehen

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.

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?	

```
13 # Richtungs-Eigenschaften
14 right_of  = {"up": "right", "down": "left" , "left": "up"   , "right": "down"}
15 left_of   = {"up": "left" , "down": "right", "left": "down", "right": "up"   }
16 delta_row = {"up": -1     , "down": 1      , "left": 0     , "right": 0     }
17 delta_col = {"up": 0      , "down": 0      , "left": -1    , "right": 1     }
```

```
 92     # Geschwindigkeit und Richtung der Schildkröte, Richtung des Roboters festlegen
 93     def init_search(self):
 94         self.t.speed(3)
 95         self.t.setheading(90)
 96         heading = "up"
 97         return heading
 98     # schau nach vorne
 99     def look_forward(self, start_row, start_col, heading):
100         return self.maze_list[start_row + delta_row[heading]]\
101                              [start_col + delta_col[heading]]
102     # schau nach rechts
103     def look_right(self, start_row, start_col, heading):
104         return self.maze_list[start_row + delta_row[right_of[heading]]]\
105                              [start_col + delta_col[right_of[heading]]]
106     # mache einen Schritt
107     def one_step(self, start_row, start_col, heading):
108         start_row += delta_row[heading]
109         start_col += delta_col[heading]
110         return start_row, start_col
111     # drehe dich nach rechts
112     def turn_right(self, heading):
113         self.t.right(90)
114         return right_of[heading]
115     # drehe dich nach links
116     def turn_left(self, heading):
117         self.t.left(90)
118         return left_of[heading]
```

```
# Roboter sucht den Weg, Schildkröte läuft mit
def search_from(maze, start_row, start_col):
    # Schildkröte auf die Startposition setzen
    maze.update_position(start_row, start_col, BLANK)
    # Spur einschalten
    maze.t.down()
    # Richtung festlegen
    heading = maze.init_search()
    print(heading)
    # cmd initialisieren
    cmd = ""
    # Solange wiederholen, bis Benutzer "e" eingibt
    while cmd != "e":
        # cmd leeren
        cmd = ""
        # Solange wiederholen, bis Benutzer etwas eingibt
        while cmd == "":
            cmd = input("Kommando v[orne], h[and], l[inks], r[rechts], s[chritt, e[nde] ")
        print("Kommando =", cmd)
        if cmd == "v":
            ch = maze.look_forward(start_row, start_col, heading)
            print(ch)
        elif cmd == "h":
            ch = maze.look_right(start_row, start_col, heading)
            print(ch)
        elif cmd == "l":
            heading = maze.turn_left(heading)
            print(heading)
        elif cmd == "r":
            heading = maze.turn_right(heading)
            print(heading)
        elif cmd == "s":
            start_row, start_col = maze.one_step(start_row, start_col, heading)
            maze.update_position(start_row, start_col)
            print("start_row =", maze.start_row, "start_col =", maze.start_col)
        elif cmd == "e":
            print("Ende - du kannst das Turtle-Fenster jetzt schließen")
        else:
            print("Unbekanntes Kommando")
    # Der Roboter sucht den Weg ... folgt
```

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?	

```
    def is_exit(self, row, col):
        return (
            row == 0
            or row == self.rows_in_maze - 1
            or col == 0
            or col == self.columns_in_maze - 1
        )
def search_from(maze, start_row, start_col):
    # Spur bis zum Start verbergen
    maze.t.up()
    maze.update_position(start_row, start_col, BLANK)
    # Spur einschalten
    maze.t.down()
    heading = maze.init_search()

    # solange kein Exit und vorne frei ist
    while maze.is_exit(start_row, start_col) == False\
      and maze.look_forward(start_row, start_col, heading) == BLANK:
        start_row, start_col = maze.one_step(start_row, start_col, heading)
        maze.update_position(start_row, start_col)
    # drehen, damit rechte Hand an der Wand ist
    heading = maze.turn_left(heading)

    # Drehungen zählen
    turn_count = 1

    # Folge der Wand bis Exit
    while maze.is_exit(start_row, start_col) == False:

        # solange vorne frei und die Wand rechts ist
        while maze.look_forward(start_row, start_col, heading) == BLANK\
          and maze.look_right(start_row, start_col, heading) == OBSTACLE:
            start_row, start_col = maze.one_step(start_row, start_col, heading)
            maze.update_position(start_row, start_col)

        # wenn die Wand nicht mehr rechts ist
        if maze.look_right(start_row, start_col, heading) == BLANK:
            # drehen und 1 Schritt vorwärts, damit rechte Hand an der Wand ist
            heading = maze.turn_right(heading)
            turn_count += 1
            start_row, start_col = maze.one_step(start_row, start_col, heading)
            maze.update_position(start_row, start_col)

        # wenn vorne eine Wand ist
        elif maze.look_forward(start_row, start_col, heading) == OBSTACLE:
            # drehen, damit rechte Hand an der Wand ist
            heading = maze.turn_left(heading)
            turn_count += 1

    # Ende der while-Schleife
    print("Exit found!")
    print(turn_count, "Drehungen")
```

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`	

```
130 def search_from(maze, start_row, start_col):
131     # Spur bis zum Start verbergen
132     maze.t.up()
133     maze.update_position(start_row, start_col, BLANK)
134     # Spur einschalten
135     maze.t.down()
136     heading = maze.init_search()
137 
138     # Drehungen zählen
139     turn_count = 0
140     # Drehungs-Level berechnen: Linksdrehung +1, Rechtsdrehung -1
141     turn_level = 0
142 
143     # Wiederhole bis Exit
144     while maze.is_exit(start_row, start_col) == False:
145 
146         # solange vorne frei ist
147         print("turn_level =", turn_level)
148         while maze.look_forward(start_row, start_col, heading) == BLANK:
149             start_row, start_col = maze.one_step(start_row, start_col, heading)
150             maze.update_position(start_row, start_col)
151         # drehen, damit rechte Hand an der Wand ist
152         heading = maze.turn_left(heading)
153         turn_count += 1
154         turn_level += 1
155 
156         # Folge der Wand bis Exit oder Drehungs-Level gleich Null
157         while maze.is_exit(start_row, start_col) == False and turn_level != 0:
158 
159             # solange vorne frei und die Wand rechts ist
160             print("turn_level =", turn_level)
161             while maze.look_forward(start_row, start_col, heading) == BLANK\
162               and maze.look_right(start_row, start_col, heading) == OBSTACLE:
163                 start_row, start_col = maze.one_step(start_row, start_col, heading)
164                 maze.update_position(start_row, start_col)
165 
166             # wenn die Wand nicht mehr rechts ist
167             if maze.look_right(start_row, start_col, heading) == BLANK:
168                 # drehen und 1 Schritt vorwärts, damit rechte Hand an der Wand ist
169                 heading = maze.turn_right(heading)
170                 turn_count += 1
171                 turn_level -= 1
172                 start_row, start_col = maze.one_step(start_row, start_col, heading)
173                 maze.update_position(start_row, start_col)
174 
175             # wenn vorne eine Wand ist
176             elif maze.look_forward(start_row, start_col, heading) == OBSTACLE:
177                 # drehen, damit rechte Hand an der Wand ist
178                 heading = maze.turn_left(heading)
179                 turn_count += 1
180                 turn_level += 1
181 
182     # Ende äußeren while-Schleife
183     print("Exit found!")
184     print(turn_count, "Drehungen")
```

Computerspiel: Auto bewegen

Wir arbeiten mit der Bibliothek "Pygame" für Computerspiele

Aufgabe:

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