PICOBOY COLOR PLUS

Arbeitsblätter

Programmieren mit Python

ab Klasse 8 · 24 Arbeitsblätter · pbcplus.de

Das Lehrerhandbuch gibt es kostenlos auf pbcplus.de/schule.
Musterlösungen und Programmdateien: E-Mail an info@picoboy.de.

Hinweis: Diese Arbeitsblätter wurden mit KI generiert.
Texte und Beispielprogramme stammen von einer künstlichen Intelligenz.
Die Programme wurden getestet – Fehler sind trotzdem möglich.
Rückmeldungen gern an info@picoboy.de.

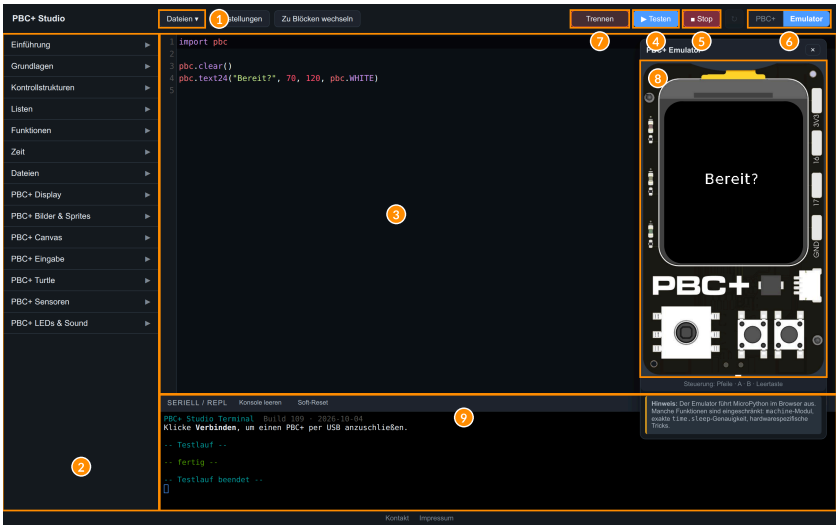
Name: _____ Klasse: _____ Datum: _____

PBC+ Studio kennenlernen

Ziel: Du lernst PBC+ Studio im Textmodus kennen, testest ein erstes Python-Programm im Emulator, änderst es, liest eine Fehlermeldung und speicherst dein Programm.

Vorbereitung: Öffne **pbcstudio.de** in Chrome oder Edge. Siehst du Blöcke, klicke oben auf **Zu Text wechseln**, dann rechts auf **Bearbeitung aktivieren** und bestätige mit **OK**.

Aufgabe 1 ⭐ Wo ist was?



Schreibe zu jeder Nummer den passenden Namen. Diese Wörter helfen dir: *Dateien* · *Editor* · *Emulator* · *Hilfe-Leiste* · *Stop* · *Terminal* · *Testen* · *Verbinden* · *Ziel (Emulator oder PBC+)*

1 _____	4 _____	7 _____
2 _____	5 _____	8 _____
3 _____	6 _____	9 _____

Aufgabe 2 ⭐ Dein erstes Programm

Lösche alles im Editor (**Strg + A**, **Entf**) und tippe ab. Achte auf jedes Zeichen!

```
1 import pbc
2
3 pbc.clear()
4 pbc.text24("Bereit?", 70, 120, pbc.WHITE)
```

Wähle oben rechts **Emulator** und klicke auf ▶ **Testen**. Was steht danach im Terminal?

_____

Aufgabe 3 ⭐ Ändern

1. Ersetze `"Bereit?"` durch `"Hallo,"` und deinen Namen, z. B. `"Hallo, Ben!"`. Ersetze `pbc.WHITE` durch `pbc.YELLOW`.
2. Schreibe in die Klammer von `pbc.clear()` die Farbe `pbc.BLUE`.
3. Teste. Ist der Text zu lang, verkleinere die Zahl `70`.



Was bewirkt die Zahl `70`? _____

Aufgabe 4 ⭐ Eine Fehlermeldung lesen

Ändere in deinem Programm `pbc.YELLOW` in `pbc.yellow` (klein) und teste.

1. Im Terminal steht eine Fehlermeldung. Schreibe ihre letzte Zeile ab:

   _____

2. In welcher Zeile ist der Fehler? _____ Was ist falsch? _____
3. Korrigiere den Fehler und teste erneut.

Aufgabe 5 ⭐ Speichern und öffnen

1. Speichere: **Dateien → Auf den Computer…** Wie heißt die Datei, und wo liegt sie?

   _____

2. Benenne sie in `hallo.py` um. Klicke auf **Dateien → Neu**. Dein Programm ist weg!
3. Hol es zurück: **Dateien → Vom Computer…**

Aufgabe 6 ⭐⭐ Die Hilfe

Öffne in der Hilfe-Leiste links **PBC+ Display** und klicke auf den Eintrag mit `text12 / 18 / 24`. Setze den Cursor unter dein Programm und klicke auf **In den Editor einfügen**. Teste. Welche Zahl gehört zur größten Schrift? _____

Das habe ich gelernt:

- ☐ Ich kenne die Teile von PBC+ Studio.
- ☐ Ich kann ein Python-Programm abtippen, im Emulator testen und anhalten.
- ☐ Ich kann in einer Fehlermeldung die Zeile und die Fehlerart finden.
- ☐ Ich kann ein Programm als `.py`-Datei speichern und wieder öffnen.

Mehr dazu im Buch: Kapitel 1 – Der PicoBoy Color Plus und PBC+ Studio, Kapitel 2 – Das erste Programm

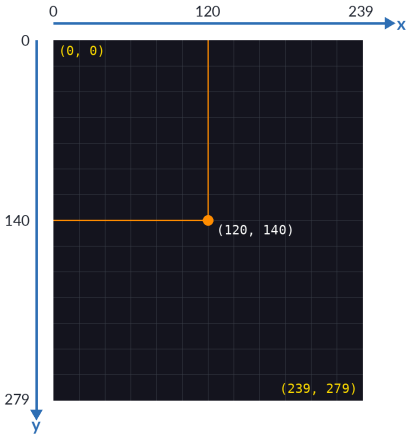
Name: _____ Klasse: _____ Datum: _____

Hallo PBC+!

Ziel: Du schreibst Texte an genau die Stelle des Bildschirms, an der du sie haben willst, und benutzt verschiedene Schriftgrößen und Farben.

So funktioniert's:

```
1 import pbc
2
3 pbc.clear()
4 pbc.text24("Hallo PBC+!", 39, 120, pbc.WHITE)
```



`pbc.text24(text, x, y, farbe)` schreibt einen Text. **x** zählt von links (0 bis 239), **y** von **oben** (0 bis 279). (x, y) ist die linke obere Ecke des Textes.

Schriften: `pbc.text` (klein, 8 Pixel), `pbc.text12`, `pbc.text18`, `pbc.text24` (groß). Farben: `pbc.WHITE`, `pbc.RED`, `pbc.GREEN`, `pbc.BLUE`, `pbc.YELLOW`, `pbc.CYAN`, `pbc.MAGENTA`, `pbc.ORANGE`, `pbc.PURPLE`, `pbc.GRAY`, `pbc.BLACK`.

Aufgabe 1 ⭐ Abtippen und verschieben

Tippe das Programm oben ab und teste es. Ändere dann nur die Zahlen für x und y. Was passiert …

- mit `x = 0`? _____
- mit `y = 260`? _____
- mit `x = 200`? _____

Aufgabe 2 ⭐ Erst denken, dann testen

Zeichne in das Koordinatensystem oben ein, wo diese Texte ungefähr erscheinen. Teste danach.

```
pbc.text12("A", 200, 20, pbc.WHITE)
pbc.text12("B", 20, 250, pbc.WHITE)
pbc.text12("C", 120, 60, pbc.WHITE)
```

Aufgabe 3 ⭐ Vier Ecken und die Mitte

Schreibe ein Programm, das „links oben", „rechts oben", „links unten", „rechts unten" und „Mitte" an die passenden Stellen schreibt (`text12`). Die Ecken des Bildschirms sind abgerundet: Halte etwa 15 Pixel Abstand zum Rand.

Welche x-Zahl hast du für „rechts oben" gebraucht? _____



Aufgabe 4 ⭐⭐ Wetterbericht

Baue diesen Wetterbericht nach. Benutze alle vier Schriften: `text` für „Wetter heute“, `text24` für das Wetter, `text18` für die Temperatur und `text12` für den Wind.



Wie viele Pixel Abstand brauchen die Zeilen mindestens, damit sie sich nicht berühren? _____

Aufgabe 5 ⭐⭐⭐ Ein Plakat

Gestalte ein Plakat für eine Veranstaltung deiner Schule (Konzert, Sportfest, Theater …): farbiger Hintergrund mit `pbc.clear(farbe)`, Überschrift groß, darunter Tag, Uhrzeit und Ort. Teile dein Programm mit Kommentaren (`#`) in Abschnitte. Achtung: Keine Umlaute! „Aula“ geht, „Flür“ nicht.

Das habe ich gelernt:

- ☐ Ich weiß, wo (0, 0) ist und in welche Richtung x und y wachsen.
- ☐ Ich kann einen Text an eine bestimmte Stelle schreiben.
- ☐ Ich kenne vier Schriftgrößen und die fertigen Farben.
- ☐ Ich kann ein Programm mit Kommentaren gliedern.

Mehr dazu im Buch: Kapitel 2 – Das erste Programm

Name: _____ Klasse: _____ Datum: _____

Frage und Antwort

> **Ziel:** Dein Programm „redet“ mit dir in der Konsole: Es stellt Fragen, merkt sich die Antworten und rechnet mit eingegebenen Zahlen.

So funktioniert’s:

Befehl	Bedeutung
`print("Hallo", name)`	schreibt eine Zeile ins Terminal; mehrere Werte durch Kommas trennen
`name = input("Frage? ")`	stellt die Frage, wartet auf eine Eingabe + **Enter** und merkt sich die Antwort als `name`
`int(text)`	macht aus einem Text wie `"14"` die Zahl `14`
`str(zahl)`	macht aus der Zahl `14` den Text `"14"`, z. B. für das Display

Die Antwort tippst du **ins Terminal** unten. `input` liefert immer **Text**, auch wenn du Ziffern eintippst. Im Terminal gehen Umlaute, auf dem Display nicht.

Aufgabe 1 ⭐ Abtippen und testen

```
1 print("Hallo! Ich bin der PicoBoy.")
2 fach = input("Was ist dein Lieblingsfach? ")
3 print("Oh,", fach, "mag ich auch!")
```

Teste das Programm. Was passiert, wenn du **nur Enter** drückst, ohne etwas einzutippen?

_____

Was passiert, wenn du in Zeile 3 `fach` in Anführungszeichen setzt? _____

Aufgabe 2 ⭐ Erst denken, dann testen

Schreibe auf, was im Terminal erscheint. Prüfe danach mit dem Emulator.

```
1 print("3" + "4")
2 print(3 + 4)
3 print("3", "4")
4 print("Ha" + "llo", "Welt")
```

1 _____

2 _____

3 _____

4 _____

Aufgabe 3 ⭐ Ein Interview

Schreibe ein Programm, das nacheinander nach **Name**, **Hobby** und **Lieblingsessen** fragt. Für jede Antwort brauchst du einen eigenen Namen, z. B. `name`, `hobby`, `essen`. Danach bedankt sich das Programm im Terminal, und die Antworten erscheinen auf dem Display, etwa mit `pbc.text18(hobby, 20, 135, pbc.CYAN)`.

Denk an `import pbc` in der ersten Zeile!



Aufgabe 4 ⭐⭐ Wie alt im Jahr 2030?

```
1 jahr = input("In welchem Jahr bist du geboren? ")
2 print("Im Jahr 2030 bist du", 2030 - jahr, "Jahre alt.")
```

1. Tippe ab und teste. Schreibe die letzte Zeile der Fehlermeldung ab:

   _____

2. Warum klappt die Rechnung nicht? _____
3. Korrigiere das Programm mit `int(…)`. Wo genau muss `int` hin?

   _____

Aufgabe 5 ⭐⭐⭐ Die Lieblingszahl

Frag nach der Lieblingszahl. Zeige auf dem Display groß die Zahl und darunter die nächste Zahl (Zahl + 1). Tipp: Die Display-Befehle brauchen einen Text, also `str(zahl + 1)`.

Das habe ich gelernt:

- ☐ Ich kann mit `print` Text und Werte ins Terminal schreiben.
- ☐ Ich kann mit `input` eine Frage stellen und die Antwort in einer Variable merken.
- ☐ Ich weiß, dass `input` Text liefert, und wandle mit `int` in eine Zahl um.
- ☐ Ich kann eine Zahl mit `str` auf dem Display zeigen.

Mehr dazu im Buch: Kapitel 3 – Die Konsole: print und input

Name: _____ Klasse: _____ Datum: _____

Rechnen mit Variablen

Ziel: Du rechnest mit Variablen, teilst mit Rest, würfelst mit Zufallszahlen und gibst Ergebnisse mit f-Strings ordentlich aus.

So funktioniert's:

Zeichen	Bedeutung
`+ - * /`	plus, minus, mal, geteilt
`//`	ganzzahlig geteilt: `17 // 5` → `3`
`%`	Rest beim Teilen: `17 % 5` → `2`
`**`	hoch: `2 ** 3` → `8`
`x += 5`	kurz für `x = x + 5`

```python
import random
wurf = random.randint(1, 6)
preis = 2.5
print(f"Wurf: {wurf}")
print(f"{preis * 3:.2f} Euro")
```

`random.randint(1, 6)`: Zufallszahl von 1 bis 6. f-String: `f"…"` – Werte in `{…}` werden eingesetzt, `:.2f` heißt zwei Nachkommastellen.

Aufgabe 1 ⭐ Was kommt heraus?

Trage zuerst ein, was du erwartest. Tippe dann ab und prüfe.

```python
a = 20
b = 6
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
print(a ** 2)
```

3 _____

4 _____

5 _____

6 _____

7 _____

8 _____

9 _____

Aufgabe 2 ⭐ Die Klassenfahrt

```python
kosten = float(input("Gesamtkosten in Euro: "))
personen = int(input("Wie viele fahren mit? "))

pro_person = kosten / personen
print(f"Jede Person zahlt {pro_person:.2f} Euro.")
```

Tippe ab und teste mit 4350 Euro und 27 Personen. Ergebnis: _____ Was passiert, wenn du `:.2f` weglässt? _____

Aufgabe 3 ⭐ Gummibärchen gerecht verteilen

Schreibe ein Programm, das fragt, wie viele Gummibärchen es gibt und wie viele Kinder sie bekommen. Es gibt aus, wie viele **jedes Kind** bekommt und wie viele **übrig** bleiben. Benutze `//` und `%`.

Bei 50 Gummibärchen und 6 Kindern: jedes _____, übrig _____

Aufgabe 4 ⭐⭐ Ein Würfel auf dem Display

Würfle mit `random.randint(1, 6)` und zeige die Zahl groß auf dem Display. Darunter steht, was das Doppelte des Wurfs ist. Benutze f-Strings auch für das Display: `pbc.text24(f"{wurf}", 112, 120, pbc.RED)`. Denk an `import pbc` **und** `import random`.



Aufgabe 5 ⭐⭐ Punkte im Spiel

Was gibt das Programm aus? Fülle die Lücken aus, dann teste.

```
punkte = 0
punkte += 100
print(f"Level geschafft: {punkte}")
punkte -= 30
print(f"Falle erwischt: {punkte}")
punkte *= 2
print(f"Doppelte Punkte: {punkte}")
```

Level geschafft: _____ Falle erwischt: _____ Doppelte Punkte: _____

Aufgabe 6 ⭐⭐⭐ Kopfrechentrainer

Das Programm denkt sich zwei Zufallszahlen von 2 bis 9 aus und fragt: „Was ist 6 mal 5?“ Du tippst eine Antwort ein, danach zeigt es dir deine Antwort und das richtige Ergebnis. Tipp: Auch die Frage in `input(…)` kann ein f-String sein.

Das habe ich gelernt:

- ☐ Ich kann mit Variablen und `+ - * / // % **` rechnen.
- ☐ Ich kann mit `//` und `%` etwas gerecht verteilen.
- ☐ Ich kann mit `random.randint` eine Zufallszahl erzeugen.
- ☐ Ich kann Ergebnisse mit f-Strings ausgeben, auch mit zwei Nachkommastellen.

Mehr dazu im Buch: Kapitel 4 – Variablen und Rechnen

Name: _____ Klasse: _____ Datum: _____

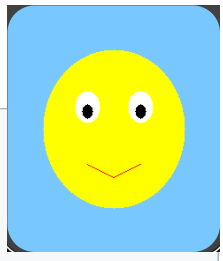
Ein Bild aus Formen

Ziel: Du zeichnest ein Bild aus Linien, Rechtecken und Ellipsen, mischst eigene Farben und weißt, in welcher Reihenfolge man zeichnen muss.

Befehl	zeichnet …
`pbc.line(x1, y1, x2, y2, farbe)`	eine Linie von (x1, y1) nach (x2, y2)
`pbc.rect(x, y, b, h, farbe)`, `pbc.fill_rect(…)`	ein Rechteck als Umriss / gefüllt; (x, y) = **linke obere Ecke**, b = Breite, h = Höhe
`pbc.ellipse(mx, my, rx, ry, farbe)`	eine Ellipse um den **Mittelpunkt** (mx, my); rx, ry = Radien; mit `fill=True` gefüllt
`pbc.rgb(r, g, b)`	eine eigene Farbe aus Rot, Grün, Blau (je 0 bis 255)

Aufgabe 1 ⭐ Ein Gesicht

Öffne `ab05_start.py` (**Dateien → Vom Computer…**) und teste es. So beginnt das Programm:



```
1 import pbc
2
3 pbc.clear(pbc.rgb(120, 200, 255))
4 # Kopf
5 pbc.ellipse(120, 140, 80, 90, pbc.YELLOW, fill=True)
```

1. Lösche in Zeile 5 `, fill=True`. Was passiert?

   _____

2. Welche Zahl in Zeile 5 machst du größer, damit der Kopf **breiter** wird? _____

Aufgabe 2 ⭐ Das Gesicht verschönern

Ergänze eine Nase, rosa Wangen (`pbc.rgb(255, 150, 150)`) und braune Haare (`pbc.rgb(120, 70, 20)`): eine Ellipse, die oben hinter dem Kopf hervorschaut. Wohin gehört diese Zeile?

_____

Aufgabe 3 ⭐ Wo sind die Augen?

Hier sieht man nur einen grünen Kreis, keine Augen. Warum? Korrigiere.

```
pbc.ellipse(95, 120, 10, 10, pbc.BLACK, fill=True)
pbc.ellipse(145, 120, 10, 10, pbc.BLACK, fill=True)
pbc.ellipse(120, 140, 70, 70, pbc.GREEN, fill=True)
```

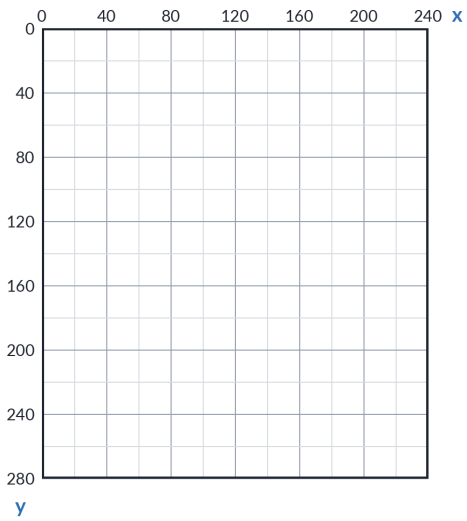
Warum? _____

Aufgabe 4 ⭐ Ecke oder Mitte?

Zeichne in das Raster ein, wo die vier Formen erscheinen. Teste danach.

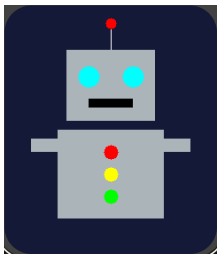
```
pbc.rect(40, 40, 80, 60, pbc.WHITE)
pbc.ellipse(40, 40, 30, 30, pbc.RED)
pbc.fill_rect(140, 180, 60, 60, pbc.GREEN)
pbc.ellipse(170, 210, 30, 30, pbc.BLUE)
```

Warum liegt der rote Kreis nicht **im** weißen Rechteck, obwohl beide mit `40, 40` beginnen?



Aufgabe 5 ⭐⭐ Ein Roboter

Baue den Roboter nach. Kopf und Körper sind gefüllte Rechtecke in der Farbe `pbc.rgb(170, 180, 190)`. Speichere sie in einer Variablen `metall`, damit du sie nur einmal eintippen musst. Der Kopf beginnt bei (70, 50) und ist 100 × 80 Pixel groß, der Körper beginnt bei (60, 140) und ist 120 × 100 Pixel groß. Die anderen Zahlen findest du durch Ausprobieren.



Plane, in welcher Reihenfolge du zeichnest:

Aufgabe 6 ⭐⭐⭐ Dein eigenes Bild

Zeichne ein eigenes Bild (Auto, Tier, Landschaft …) mit mindestens einer eigenen Farbe. Nimm `pbc.WIDTH` für etwas, das über die ganze Breite geht, z. B. eine Straße. Gliedere dein Programm mit Kommentaren.

Das habe ich gelernt:

- ☐ Ich kann Linien, Rechtecke und Ellipsen zeichnen, gefüllt und als Umriss.
- ☐ Ich weiß: Bei Rechtecken ist (x, y) die linke obere Ecke, bei Ellipsen der Mittelpunkt.
- ☐ Ich zeichne von hinten nach vorn, weil Späteres das Frühere verdeckt.
- ☐ Ich kann mit `pbc.rgb` eigene Farben mischen.

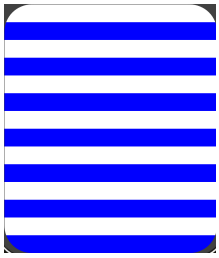
Mehr dazu im Buch: Kapitel 5 – Zeichnen und Farben

Name: _____ Klasse: _____ Datum: _____

Muster mit Schleifen

Ziel: Du lässt den PicoBoy Anweisungen mit einer `for`-Schleife wiederholen und zeichnest damit Streifen, Treppen und Schachbretter.

So funktioniert's:



```
1 import pbc
2
3 pbc.clear(pbc.BLUE)
4 for i in range(7):
5     pbc.fill_rect(0, i * 40, 240, 20, pbc.WHITE)
```

`range(7)` liefert 0, 1, 2 … 6. Bei jedem Durchlauf steht die nächste Zahl in `i`, und die **eingerückte** Zeile 5 läuft einmal. Doppelpunkt am Ende von Zeile 4 nicht vergessen! `range(a, b)` zählt von a bis **vor** b, `range(a, b, s)` in Schritten von s.

Aufgabe 1 ⭐ Erst denken, dann testen

Was gibt das Programm im Terminal aus? Schreibe es auf und teste dann.

```
1 for i in range(4):
2     print(i, i * 10)
3 print("fertig")
```

Wie oft erscheint „fertig", und warum?

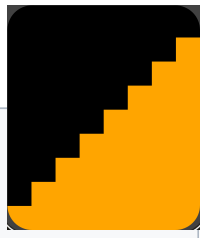
_____

Aufgabe 2 ⭐ Streifen

Tippe das Programm von oben ab (oder öffne `ab06_start.py`) und teste es.

1. Bei welchen y-Werten beginnen die weißen Streifen? _____
2. Mache die Streifen nur 10 Pixel hoch. Welche Zahl änderst du? _____
3. Ändere das Programm so, dass die Streifen **senkrecht** stehen. Tipp: `for x in range(0, 240, 40):`

Aufgabe 3 ⭐ Eine Treppe



Jede Stufe ist 30 Pixel breit und reicht bis zum unteren Rand (y = 280). Ergänze die Lücke und teste.

```
for stufe in range(8):
    x = stufe * 30
    y = 250 - stufe * 30
    pbc.fill_rect(x, y, 30, ______, pbc.ORANGE)
```

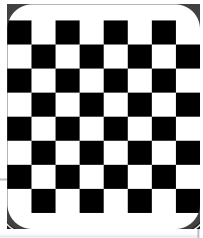
Lücke: _____ Tipp: Stufe 0 ist 30 hoch, Stufe 1 ist 60 hoch …

Aufgabe 4 ⭐⭐ Schleife in der Schleife

```
for zeile in range(5):
    for spalte in range(6):
        x = 30 + spalte * 36
        y = 50 + zeile * 45
        farbe = pbc.rgb(255, zeile * 60, 0)
        pbc.ellipse(x, y, 12, 12, farbe, fill=True)
```

Wie viele Kreise entstehen? _____ Welche Farbe hat die oberste Reihe (`zeile` = 0)? _____ Die unterste? _____ Teste danach.

Aufgabe 5 ⭐⭐ Ein Schachbrett

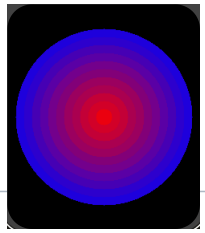


Zeichne ein Schachbrett aus 8 × 8 Feldern, jedes 30 × 30 Pixel. Fülle den Bildschirm zuerst weiß und zeichne nur die schwarzen Felder, beginnend bei y = 20. Tipp: In Zeile 0 sind die Spalten 0, 2, 4, 6 schwarz, in Zeile 1 die Spalten 1, 3, 5, 7. Das liefert `range(zeile % 2, 8, 2)`.

```

```

Aufgabe 6 ⭐⭐⭐ Der Tunnel



Zeichne mit einer einzigen Schleife gefüllte Kreise um die Bildschirmmitte mit den Radien 110, 100, 90 … 10. Berechne die Farbe aus dem Radius `r`, z. B. `pbc.rgb(255 - r * 2, 0, r * 2)`. Warum musst du **rückwärts** zählen?

_____

Das habe ich gelernt:

- ☐ Ich kann mit `for` und `range` Anweisungen wiederholen.
- ☐ Ich weiß, was eingerückt zur Schleife gehört und was danach kommt.
- ☐ Ich kann mit der Schleifenvariablen Positionen und Farben ausrechnen.
- ☐ Ich kann mit zwei Schleifen ineinander Zeilen und Spalten zeichnen.

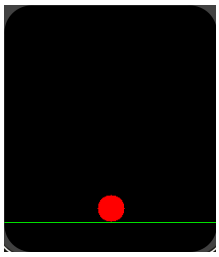
Mehr dazu im Buch: Kapitel 6 – Schleifen und Zeit

Name: _____ Klasse: _____ Datum: _____

Der springende Ball

Ziel: Du bringst einen Ball zum Springen: Er fällt, prallt auf und steigt wieder, endlos. Dafür brauchst du Schleifen, Pausen und das Übermalen der alten Stelle.

So funktioniert's: Öffne `ab07_start.py` (**Dateien → Vom Computer…**).



```
1 import pbc
2 import time
3
4 pbc.clear()
5 pbc.hline(0, 246, 240, pbc.GREEN)
6 # Der Ball faellt von oben bis auf den Boden
7 for y in range(30, 231, 5):
8     pbc.ellipse(120, y, 15, 15, pbc.RED, fill=True)
9     time.sleep_ms(20)
10    pbc.ellipse(120, y, 15, 15, pbc.BLACK, fill=True)
11 pbc.ellipse(120, 230, 15, 15, pbc.RED, fill=True)
```

Für jedes `y` gilt: **zeichnen – warten – übermalen**. `time.sleep_ms(20)` wartet 20 Millisekunden. Ohne `import time` gibt es einen `NameError`.

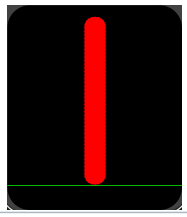
Aufgabe 1 ⭐ Schneller, langsamer

Teste das Programm. Ändere dann jeweils nur eine Zahl und teste wieder.

- `time.sleep_ms(60)` statt `20`: _____
- Schrittweite `10` statt `5` in Zeile 7: _____
- Wie viele Bilder zeigt die Schleife mit Schrittweite 5? _____

Aufgabe 2 ⭐ Ohne Übermalen

Was passiert, wenn du Zeile 10 löschst? Vermute zuerst, teste dann.



_____

Warum malt Zeile 10 den Ball **schwarz**?

_____

Aufgabe 3 ⭐ Hoch und runter, immer wieder

Der Ball soll nach dem Aufprall wieder nach oben steigen und dann endlos weiterspringen. Baue das Programm so um (Zeile 11 brauchst du nicht mehr; achte auf die Einrückung):

```
1 while True:
2     # Fallen
3     for y in range(30, 231, 5):
4         ...    # die drei Zeilen von oben
5     # Steigen
6     for y in range(230, 29, -5):
7         ...    # dieselben drei Zeilen
```

Warum steht im zweiten `range` die 29 und nicht 30? _____ Wie hältst du das Programm an? _____

Aufgabe 4 ⭐⭐ Platsch!

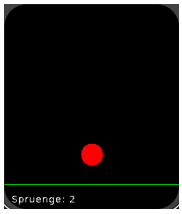
Ein echter Ball wird beim Aufprall kurz flach gedrückt. Zeichne zwischen den beiden `for`-Schleifen eine breite, flache rote Ellipse auf den Boden, z. B. `pbc.ellipse(120, 237, 22, 8, pbc.RED, fill=True)`, warte 80 ms und übermale sie wieder schwarz.

Aufgabe 5 ⭐⭐ Sprünge zählen

Lege vor `while True` eine Variable `spruenge = 0` an. Erhöhe sie bei jedem Aufprall um 1 und zeige sie unten an: erst ein schwarzes Rechteck über die alte Zahl, dann `pbc.text12(f"Spruenge: {spruenge}", 10, 258, pbc.WHITE)`

In welche Zeile deines Programms gehört `spruenge += 1`?

_____



Aufgabe 6 ⭐⭐⭐ Der Ball hüpft weiter

Lass den Ball beim Springen nach rechts wandern. Ersetze `120` durch eine Variable `x` und erhöhe sie in **beiden** Schleifen nach jedem Bild: `x = (x + 2) % pbc.WIDTH`. Was bewirkt `% pbc.WIDTH`?

_____

> Der Ball flackert ein wenig, weil jedes Zeichnen sofort auf dem Display erscheint. Ganz ohne Flackern geht es mit einem Canvas (Buch, Kapitel 17). Einen Ball, der an allen Rändern abprallt, baust du mit `if` (Kapitel 7).

Das habe ich gelernt:

- ☐ Ich kann ein Programm mit `time.sleep_ms` warten lassen.
- ☐ Ich kann eine Figur bewegen: zeichnen – warten – übermalen.
- ☐ Ich kann mit `range` vorwärts und rückwärts zählen.
- ☐ Ich kann mit `while True` etwas endlos wiederholen und es mit Stop anhalten.

Mehr dazu im Buch: Kapitel 6 – Schleifen und Zeit

Name: _____ Klasse: _____ Datum: _____

Zahlen raten

Ziel: Dein Programm trifft Entscheidungen mit `if`, `elif` und `else` und wiederholt mit `while`, bis eine Bedingung erfüllt ist. Am Ende knackst du einen Tresor-Code im Terminal.

Aufgabe 1 ⭐ Der Tresor

Tippe ab (oder öffne `ab08_start.py`) und teste. Die Eingabe machst du unten im Terminal, danach **Enter**.

```
1 code = 42
2 tipp = int(input("Tresor-Code (0 bis 99): "))
3 if tipp == code:
4     print("Klick! Der Tresor ist offen.")
5 else:
6     print("Falscher Code.")
```

Teste mit `17` und `42`. Welche Zeilen laufen? Bei 17: _____ Bei 42: _____ Warum sind die Zeilen 4 und 6 eingerückt? _____

Aufgabe 2 ⭐ Wahr oder falsch?

Es ist `code = 42` und `tipp = 17`. Kreuze an.

Ausdruck	True	False	Ausdruck	True	False
`tipp < code`	☐	☐	`tipp != code`	☐	☐
`tipp >= 17`	☐	☐	`tipp + 25 == code`	☐	☐
`tipp > 0 and tipp > 50`	☐	☐	`tipp > 0 or tipp > 50`	☐	☐

Was ist der Unterschied zwischen `code = 42` und `code == 42`?

Aufgabe 3 ⭐ Ein Hinweis hilft

Bei falschem Code soll das Programm verraten, ob der richtige Code größer oder kleiner ist. Fülle die Lücken, ändere dein Programm, teste mit 17, 42 und 80.

```
if tipp == code:
    print("Klick! Der Tresor ist offen.")
elif tipp ______ code:
    print("Falsch. Der Code ist größer.")
______
    print("Falsch. Der Code ist kleiner.")
```

Aufgabe 4 ⭐ Bis der Tresor aufgeht

Jetzt soll das Programm so lange fragen, bis der Code stimmt, und die Versuche zählen. Baue dein Programm nach diesem Plan um:

```
code = 42
versuche = 0
while True:
    tipp = int(input("Tresor-Code (0 bis 99): "))
    versuche += 1
    # hier dein if / elif / else aus Aufgabe 3,
    # im if-Block nach dem print zusätzlich: break
print(f"Du hast {versuche} Versuche gebraucht.")
```

Was bewirkt `break`? _____

Wie viele Versuche brauchst du höchstens, wenn du geschickt rätst? _____

Aufgabe 5 ⭐⭐ Alarm!

1. Der Code soll zufällig sein: `import random` als erste Zeile, `code = random.randint(0, 99)`.
2. Nach 6 falschen Versuchen geht der Alarm los. Ersetze `while True:` durch `while versuche < 6:`. Nach der Schleife steht:

```
if tipp != code:
    print(f"ALARM! Der Code war {code}.")
```

Warum braucht man dieses `if` nach der Schleife?

_____

Aufgabe 6 ⭐⭐⭐ Der PicoBoy rät

Jetzt umgekehrt: Du denkst dir eine Zahl von 1 bis 100, der PicoBoy rät. Du antwortest mit `k` (zu klein), `g` (zu groß) oder `r` (richtig). Der PicoBoy merkt sich in `unten` und `oben`, welche Zahlen noch möglich sind, und rät immer die Mitte: `tipp = (unten + oben) // 2`. Bei `k` wird `unten = tipp + 1`, bei `g` wird `oben = tipp - 1`. Schreibe das Programm. Wie viele Versuche braucht er höchstens?

_____

Das habe ich gelernt:

- ☐ Ich kann mit `if`, `elif` und `else` zwischen mehreren Fällen entscheiden.
- ☐ Ich kenne die Vergleiche `==`, `!=`, `<`, `<=`, `>`, `>=` und weiß, dass `=` etwas anderes ist.
- ☐ Ich weiß, dass alles, was zu einem `if` oder einer Schleife gehört, eingerückt sein muss.
- ☐ Ich kann mit `while` wiederholen, bis etwas eintritt, und mit `break` aussteigen.

Mehr dazu im Buch: Kapitel 7 – Entscheidungen

Name: _____ Klasse: _____ Datum: _____

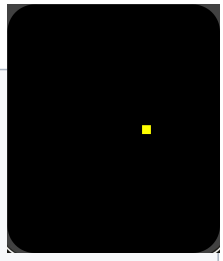
Steuern mit dem Joystick

Ziel: Du steuerst eine Spielfigur mit dem Joystick über den Bildschirm, hältst sie am Rand an und verknüpfst Bedingungen mit `and`, `or` und `not`.

So funktioniert's: `pbc.pressed_right()` ist `True`, **solange** der Joystick nach rechts gedrückt ist. Ebenso `pressed_left()`, `pressed_up()`, `pressed_down()`, `pressed_center()` (Joystick drücken), `pressed_a()`, `pressed_b()`. **Im Emulator:** Pfeiltasten, Leertaste = MITTE, Tasten A und B. Reagiert er nicht, klicke einmal auf das Bild des PicoBoy.

Aufgabe 1 ⭐ Die Figur bewegt sich

Öffne `ab09_start.py` (oder tippe ab) und teste mit den Pfeiltasten.



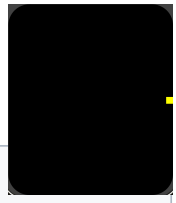
```
1 import pbc
2 import time
3 x = 116
4 y = 136
5 pbc.clear()
6 while True:
7     # Figur an der alten Stelle loeschen
8     pbc.fill_rect(x, y, 10, 10, pbc.BLACK)
9     if pbc.pressed_right():
10        x += 2
11    if pbc.pressed_left():
12        x -= 2
13    # Figur an der neuen Stelle zeichnen
14    pbc.fill_rect(x, y, 10, 10, pbc.YELLOW)
15    time.sleep_ms(20)
```

1. Ergänze zwei `if`-Abfragen, damit die Figur auch nach oben und unten fährt. Achtung: y wächst nach **unten**.
2. Mach Zeile 8 zum Kommentar (`#` davor). Was passiert jetzt?

_____

Aufgabe 2 ⭐ Halt am Rand!

Fahre lange nach rechts: Die Figur verschwindet. Sie soll am Rand stehen bleiben. Die Figur ist 10 × 10 Pixel groß, (x, y) ist ihre linke obere Ecke, der Bildschirm ist 240 × 280 Pixel groß. Ergänze und teste alle vier Ränder.



```
if pbc.pressed_right() and x < ____:
if pbc.pressed_left() and x > ____:
if pbc.pressed_up() and y > ____:
if pbc.pressed_down() and y < ____:
```

Warum ist die rechte Grenze nicht 240? _____

Aufgabe 3 ⭐ `and`, `or`, `not`

Wann läuft der Block? Kreuze an, welche Tasten dafür gedrückt sein können.

Bedingung	nur A	nur B	A und B	nichts
`if pbc.pressed_a() and pbc.pressed_b():`	☐	☐	☐	☐
`if pbc.pressed_a() or pbc.pressed_b():`	☐	☐	☐	☐
`if not pbc.pressed_a():`	☐	☐	☐	☐

Aufgabe 4 ⭐⭐ Turbo

Solange B gedrückt ist, fährt die Figur schneller. Setze dafür am Anfang der Schleife `schritt = 2` und, wenn B gedrückt ist, `schritt = 6`. Ersetze dann überall `2` durch `schritt`. Am Rand musst du jetzt aufpassen: Schreibe zum Beispiel `x + schritt <= 230` statt `x < 230`. Warum?

Aufgabe 5 ⭐⭐ Alarmfarbe

Solange A **oder** MITTE gedrückt ist, soll die Figur rot sein, sonst gelb. Speichere die Farbe in einer Variablen `farbe` und benutze sie in der Zeile, die die Figur zeichnet.

Aufgabe 6 ⭐⭐⭐ Ins Ziel

Zeichne vor der Schleife ein grünes Ziel: `pbc.fill_rect(200, 125, 30, 30, pbc.GREEN)`. Die Figur startet links (`x = 20`). Liegt sie ganz im Ziel, erscheint „Ziel erreicht!“, und das Programm endet mit `break`. Für „ganz im Ziel“ brauchst du vier Vergleiche, die mit `and` verbunden sind. Schreibe die Bedingung auf:



Das habe ich gelernt:

- ☐ Ich kann mit `pbc.pressed_…()` abfragen, ob eine Taste gerade gedrückt ist.
- ☐ Ich kann eine Figur bewegen, indem ich sie lösche, ihre Position ändere und neu zeichne.
- ☐ Ich kann Bedingungen mit `and`, `or` und `not` verknüpfen.
- ☐ Ich kann eine Figur am Rand anhalten.

Mehr dazu im Buch: Kapitel 8 – Tasten und Joystick

Name: _____ Klasse: _____ Datum: _____

Der Reaktionstest

Ziel: Du misst mit `time.ticks_ms()` auf die Millisekunde genau, wie schnell du reagierst, erkennst mit `pbc.was_pressed_…()`, wer zu früh drückt, und baust ein Duell für zwei Personen.

So funktioniert's: `pbc.was_pressed_center()` ist `True`, wenn der Joystick (MITTE, im Emulator die **Leertaste**) seit der letzten Abfrage gedrückt wurde, und zwar **einmal pro Druck**. Der PicoBoy merkt sich den Druck, auch wenn das Programm gerade wartet. `time.ticks_ms()` ist ein Zähler, der jede Millisekunde um 1 wächst. `time.ticks_diff(spaeter, frueher)` rechnet aus, wie viele Millisekunden dazwischen liegen.

Aufgabe 1 ⭐ Wie schnell bist du?

Öffne `ab10_start.py` und teste. Sobald der gelbe Kreis erscheint, drückst du die Leertaste. Die Zeit steht im Terminal.




```
1  import pbc
2  import time
3  import random
4  pbc.clear()
5  pbc.text18("Achtung ...", 75, 40, pbc.WHITE)
6  time.sleep_ms(random.randint(1000, 4000))
7  pbc.ellipse(120, 150, 60, 60, pbc.YELLOW, fill=True)
8  start = time.ticks_ms()
9  while not pbc.was_pressed_center():
10     time.sleep_ms(1)
11 ende = time.ticks_ms()
12 zeit = time.ticks_diff(ende, start)
13 print("Reaktionszeit:", zeit, "ms")
```

Versuch	1	2	3	4	5
Zeit in ms					

Aufgabe 2 ⭐ Programm verstehen

1. Warum wartet Zeile 6 eine **zufällige** Zeit? _____
2. Was macht die Schleife in Zeile 9 und 10? _____
3. Teste: Drücke die Leertaste, **bevor** der Kreis erscheint. Was passiert, und warum?

_____

_____

Aufgabe 3 ⭐ Bewertung auf dem Display

Hänge ans Programm an: Bildschirm löschen, die Zeit groß anzeigen (`pbc.text24(f"{zeit} ms", 75, 100, pbc.WHITE)`) und darunter eine Bewertung mit `if`, `elif` und `else`:



Zeit	unter 300 ms	unter 500 ms	sonst
Text (Farbe)	„Blitzschnell!“ (grün)	„Gut!“ (gelb)	„Aufwachen!“ (rot)

Warum muss `zeit < 300` vor `zeit < 500` geprüft werden? _____

Aufgabe 4 ⭐⭐ Zu früh!

Wer drückt, bevor der Kreis erscheint, soll keine Zeit bekommen. Prüfe direkt nach Zeile 6 mit `if pbc.was_pressed_center():`, ob schon gedrückt wurde. Dann wird der Bildschirm rot, und es erscheint „Zu frueh!“. Alles andere (Kreis, Messen, Bewertung) kommt in den `else`-Block. Denk an die Einrückung!



Aufgabe 5 ⭐⭐⭐ Das Duell

Zwei Personen, ein PicoBoy: Person 1 drückt A, Person 2 drückt B. Wer nach dem Erscheinen des Kreises zuerst drückt, gewinnt. Warte in einer Schleife `while True:` auf `was_pressed_a()` oder `was_pressed_b()`, merke dir den Sieger in einer Variablen und steige mit `break` aus. Damit ein zu frühes Drücken nicht zählt, rufe vor dem Kreis `pbc.was_pressed_a()` und `pbc.was_pressed_b()` einmal auf. Warum hilft das?



_____

Das habe ich gelernt:

- ☐ Ich kann mit `time.ticks_ms()` und `time.ticks_diff()` eine Zeitspanne messen.
- ☐ Ich kenne den Unterschied zwischen `pressed_…()` und `was_pressed_…()`.
- ☐ Ich kann mit einer Schleife auf eine bestimmte Taste warten.
- ☐ Ich kann mit `if` / `elif` / `else` ein Ergebnis bewerten.

Mehr dazu im Buch: Kapitel 8 – Tasten und Joystick

Name: _____ Klasse: _____ Datum: _____

Musik und Licht

> **Ziel:** Du schaltest die LEDs des PicoBoy, mischst Farben mit der RGB-LED und spielst Töne und eine Melodie.

So funktioniert's:



Befehl	Wirkung
`pbc.led_red(True)` / `(False)`	rote LED an / aus (ebenso `led_yellow`, `led_green`)
`pbc.leds_off()`	alle drei LEDs aus
`pbc.rgb_led(r, g, b)`	RGB-LED: Rot, Grün, Blau je 0 bis 255
`pbc.rgb_led_off()`	RGB-LED aus
`pbc.tone(440)`	Dauerton mit 440 Hz, bis `pbc.tone_off()`
`pbc.tone(440, 300)`	Ton für 300 ms. **Wartet nicht!** Danach `time.sleep_ms(…)`

Aufgabe 1 ⭐ Die Ampel

Öffne `ab11_start.py` (oder tippe ab) und teste. Die LEDs links neben dem Display zeigen eine Ampel.

```
 1 import pbc
 2 import time
 3
 4 pbc.led_red(True)                # Rot
 5 time.sleep(2)
 6 pbc.led_yellow(True)             # Rot-Gelb
 7 time.sleep(1)
 8 pbc.led_red(False)
 9 pbc.led_yellow(False)
10 pbc.led_green(True)              # Gruen
11 time.sleep(2)
```

1. Eine echte Ampel hat noch eine Phase. Welche? _____ Ergänze sie. Am Ende sollen alle LEDs aus sein.
2. Die Ampel soll dreimal hintereinander laufen. Benutze eine `for`-Schleife.

Aufgabe 2 ⭐ Farben mischen

Rate zuerst, welche Farbe die RGB-LED zeigt. Prüfe dann mit `pbc.rgb_led`.

`255, 0, 0` →	`255, 255, 0` →	`255, 255, 255` →
`0, 0, 255` →	`0, 255, 255` →	`0, 0, 0` →

Aufgabe 3 ⭐ Was hörst du?

Diese Zeilen sollen drei Töne nacheinander spielen (`import pbc` und `import time` stehen darüber):

```
pbc.tone(523, 300)
pbc.tone(659, 300)
pbc.tone(784, 300)
```

Teste. Was hörst du, und warum? _____
Korrigiere das Programm, sodass alle drei Töne zu hören sind.

Aufgabe 4 ⭐⭐ Bruder Jakob

Programmiere den Anfang von „Bruder Jakob“. Frequenzen: C = 262, D = 294, E = 330, F = 349, G = 392 (in Hz). Jede Note dauert 400 ms, nur das G mit Strich dauert doppelt so lang:

C D E C · C D E C · E F G– · E F G–

Tipps: Lege die Frequenzen als Konstanten an (`C = 262` …). Spiele jeden Ton 50 ms kürzer, als du wartest, damit man die Noten unterscheiden kann. Jeder Teil kommt zweimal vor: Dafür passt eine `for`-Schleife.

Aufgabe 5 ⭐⭐ Licht zur Musik

Lass die RGB-LED bei jeder Note deiner Melodie in einer anderen Farbe leuchten, zum Beispiel C rot, D grün, E blau. Schreibe den Namen des Lieds auf das Display. Am Ende soll die LED aus sein.

Welche Zeile fügst du vor jedes `pbc.tone` ein? _____

Aufgabe 6 ⭐⭐⭐ Die Türklingel

Schreibe eine Klingel: Wird Taste A gedrückt (`pbc.was_pressed_a()`), klingt „ding“ (659 Hz) und dann „dong“ (523 Hz). Dabei leuchten die RGB-LED gelb und nacheinander die rote und die grüne LED. Danach ist alles wieder aus und die Klingel wartet auf den nächsten Druck.

Das habe ich gelernt:

- ☐ Ich kann die drei LEDs ein- und ausschalten.
- ☐ Ich kann mit der RGB-LED Farben aus Rot, Grün und Blau mischen.
- ☐ Ich weiß, dass `pbc.tone` nicht wartet, und kann Töne mit Pausen nacheinander spielen.
- ☐ Ich kann eine kurze Melodie programmieren.

Mehr dazu im Buch: Kapitel 9 – LEDs und Töne

Name: _____ Klasse: _____ Datum: _____

Die Geheimschrift

Ziel: Du untersuchst Texte Zeichen für Zeichen, rechnest mit Buchstaben und verschlüsselst eine Nachricht wie Julius Caesar.

So funktioniert's: `len(text)` zählt die Zeichen. `text[i]` holt das Zeichen mit dem **Index** `i`; gezählt wird ab **0**, `text[-1]` ist das letzte Zeichen. `text[a:b]` ist das Stück von `a` bis **vor** `b`. `for zeichen in text:` durchläuft einen Text Zeichen für Zeichen. `ord("A")` liefert den Zeichencode 65, `chr(65)` macht daraus wieder `"A"`.

Aufgabe 1 ⭐ Texte untersuchen

Was gibt jede Zeile aus? Schreibe es zuerst auf, teste dann.

```
1 wort = "Geheimnis"
2 print(len(wort))
3 print(wort[0])
4 print(wort[3])
5 print(wort[-1])
6 print(wort[0:4])
7 print(wort.upper())
```

2 _____

3 _____

4 _____

5 _____

6 _____

7 _____

Aufgabe 2 ⭐ Buchstaben zählen

```
1 satz = input("Satz: ")
2
3 anzahl = 0
4 for zeichen in satz:
5     if zeichen == "e" or zeichen == "E":
6         anzahl += 1
7 print("Der Buchstabe e kommt", anzahl, "Mal vor.")
```

Tippe ab und teste mit einem eigenen Satz. Ändere das Programm dann so, dass es die Leerzeichen zählt. Wie viele Wörter hat dein Satz also? _____

Aufgabe 3 ⭐ Buchstaben sind Zahlen

Die Großbuchstaben A bis Z haben die Codes 65 bis 90. Ergänze, prüfe dann mit `print`.

`ord("C")` = _____ · `chr(66)` = _____ · `chr(ord("A") + 3)` = _____ · `ord("Z")-65` = _____

Aufgabe 4 ⭐ Die Caesar-Verschlüsselung

Caesar verschob jeden Buchstaben um 3 Stellen im Alphabet: A → D, B → E, … Am Ende geht es vorn weiter: X → A, Y → B, Z → C. Öffne `ab12_start.py`:

```
1 text = "TREFFEN UM DREI"
2 schluessel = 3
3
4 geheim = ""
5 for zeichen in text:
6     if "A" <= zeichen <= "Z":
7         nummer = ord(zeichen) - 65      # A=0, B=1 ... Z=25
8         # HIER: nummer um schluessel verschieben
9         geheim += chr(nummer + 65)
10    else:
11        geheim += zeichen
12 print(geheim)
```

Ersetze Zeile 8 durch eine Rechnung, die `schluessel` zu `nummer` addiert. Tipp: `% 26` (Rest bei Division durch 26) macht aus 26 wieder 0, aus 27 die 1 …

Zeile 8: _____

Geheimtext: _____

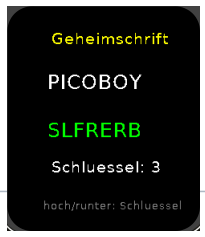
Aufgabe 5 ⭐⭐ Entschlüsseln

Diese Nachricht wurde mit dem Schlüssel 3 verschlüsselt: `PDWKH LVW FRRO`. Wie musst du `schluessel` setzen, um sie zu entschlüsseln? _____ Was steht da?

_____

Aufgabe 6 ⭐⭐⭐ Schlüssel einstellen

Zeige auf dem Display ein Wort und darunter seine Verschlüsselung. Mit dem Joystick (hoch/runter) stellst du den Schlüssel von 0 bis 25 ein; das verschlüsselte Wort ändert sich sofort mit. Tipp: Zeichne nur neu, wenn sich der Schlüssel geändert hat, und übermale vorher den alten Text mit `pbc.fill_rect`.



Das habe ich gelernt:

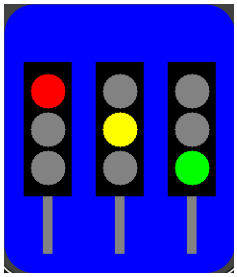
- ☐ Ich kann mit `len`, `text[i]` und `text[a:b]` Texte untersuchen.
- ☐ Ich kann einen Text mit `for` Zeichen für Zeichen durchlaufen.
- ☐ Ich kann mit `ord` und `chr` zwischen Zeichen und Zahlen umrechnen.
- ☐ Ich kann eine Nachricht mit der Caesar-Verschlüsselung ver- und entschlüsseln.

Mehr dazu im Buch: Kapitel 10 – Texte (Strings)

Name: _____ Klasse: _____ Datum: _____

Ordnung mit Funktionen

Ziel: Du räumst ein unübersichtliches Programm auf, indem du wiederkehrende Teile in eine eigene Funktion mit Parametern steckst. **Neu:** `def`, Parameter, Standardwerte



So funktioniert's:

```
1 def gruss(name, mal):
2     for i in range(mal):
3         print("Hallo", name)
4
5
6 gruss("Ada", 2)
7 gruss("Ben", 1)
```

`def` legt eine **Funktion** fest. Die eingerückten Zeilen sind ihr Körper; sie laufen erst beim **Aufruf** (Zeile 6 und 7). `name` und `mal` sind **Parameter**: Beim Aufruf bekommen sie die Werte in den Klammern, der Reihe nach.

Aufgabe 1 ⭐ Das Startprogramm

Öffne `ab13_start.py` (**Dateien → Vom Computer…**) und teste es. Nach `pbc.clear(pbc.BLUE)` kommt Ampel 1:

```
# Ampel 1: rot
pbc.fill_rect(20, 60, 50, 140, pbc.BLACK)
pbc.ellipse(45, 90, 18, 18, pbc.RED, fill=True)
pbc.ellipse(45, 130, 18, 18, pbc.GRAY, fill=True)
pbc.ellipse(45, 170, 18, 18, pbc.GRAY, fill=True)
pbc.fill_rect(40, 200, 10, 60, pbc.GRAY)
```

Danach folgen fast dieselben Zeilen für Ampel 2 und 3. Was ist bei den drei Ampeln **verschieden**?

_____

Aufgabe 2 ⭐ Was gibt das Programm aus?

```
1 def zeige(farbe, zahl):
2     print(zahl, "mal", farbe)
3
4
5 print("Los")
6 zeige("rot", 2)
7 zeige(3, "gelb")
```

Erst denken, dann testen! Ausgabe:

_____

_____

_____

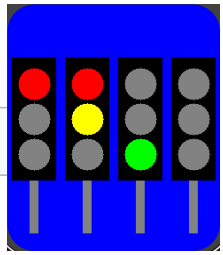
Aufgabe 3 ⭐ Eine Funktion für alle Ampeln

Ergänze die Funktion und ersetze damit die 18 Zeilen für die Ampeln:

```
def ampel(x, oben, mitte, unten):
    pbc.fill_rect(x, 60, 50, 140, pbc.BLACK)
    pbc.ellipse(x + 25, 90, 18, 18, oben, fill=True)
```

Ampel 1 zeichnest du nach `pbc.clear(pbc.BLUE)` mit `ampel(20, pbc.RED, pbc.GRAY, pbc.GRAY)`. Schreibe die Aufrufe für Ampel 2 und 3 dazu und teste. Wie viele Zeilen ist dein Programm jetzt lang? _____

Aufgabe 4 ⭐⭐ Standardwerte



Meistens ist ein Licht aus (grau). Gib den drei Farb-Parametern den Standardwert `pbc.GRAY`:

```
def ampel(x, oben=pbc.GRAY, mitte=pbc.GRAY, unten=pbc.GRAY):
```

Jetzt reicht für eine rote Ampel `ampel(5, oben=pbc.RED)`. Zeichne vier Ampeln: Rot, Rot-Gelb, Grün und eine ausgeschaltete. Nimm für x die Werte 5, 65, 125 und 185.

Wie lautet dein Aufruf für die ausgeschaltete Ampel? _____

Aufgabe 5 ⭐⭐ Die Ampel schaltet

Schreibe eine Funktion `phase(oben, mitte, unten, sekunden)`. Sie ruft `ampel(95, oben, mitte, unten)` auf und wartet dann `sekunden` Sekunden (`import time`!). Lass im Hauptprogramm in einer `while True`-Schleife die vier Phasen Rot (3 s), Rot-Gelb (1 s), Grün (3 s), Gelb (1 s) ablaufen.

Aufgabe 6 ⭐⭐⭐ Echte Lichter

Erweitere `phase` so, dass auch die drei Status-LEDs des PicoBoy mitleuchten (`pbc.led_red(True)` …). Tipp: `oben != pbc.GRAY` ist `True`, wenn das obere Licht an ist.

Das habe ich gelernt:

- ☐ Ich kann mit `def` eine Funktion mit Parametern schreiben und aufrufen.
- ☐ Ich erkenne Wiederholungen in einem Programm und ersetze sie durch Aufrufe.
- ☐ Ich weiß, dass die Argumente der Reihe nach den Parametern zugeordnet werden.
- ☐ Ich kann Parametern Standardwerte geben.

Mehr dazu im Buch: Kapitel 11 – Funktionen

Name: _____ Klasse: _____ Datum: _____

Werkzeuge mit Rückgabewert

Ziel: Du schreibst kleine Rechenfunktionen, die ein Ergebnis zurückgeben, und prüfst mit Tests, ob sie richtig rechnen. **Neu:** `return`, Funktionen mit `True` / `False`, `assert`

So funktioniert's (`ab14_start.py`):

```
1 def ist_gerade(n):
2     return n % 2 == 0
3
4
5 print(ist_gerade(4), "erwartet: True")
6 print(ist_gerade(7), "erwartet: False")
7 print(ist_gerade(0), "erwartet: True")
```

`return` gibt ein Ergebnis an die Stelle zurück, an der die Funktion aufgerufen wurde, und beendet die Funktion. Der Aufruf `ist_gerade(4)` steht dann für den **Rückgabewert** `True`.

Testen mit `assert`: `assert ist_gerade(4)` tut nichts, wenn die Bedingung stimmt. Ist sie falsch, bricht das Programm mit `AssertionError` und der Zeilennummer ab. Teste **typische Fälle** und **Grenzfälle** (0, gleiche Zahlen, negative Zahlen).

Aufgabe 1 ⭐ Gerade und teilbar

Öffne `ab14_start.py` und teste es. Schreibe darunter eine Funktion `ist_teilbar(zahl, teiler)`, die `True` liefert, wenn `zahl` ohne Rest durch `teiler` teilbar ist. Teste sie mit `print` wie oben, z. B. mit 15 und 5 sowie 15 und 4.

Aufgabe 2 ⭐ `return` oder `print`?

```
1  def halb(x):
2      return x / 2
3
4
5  def sag_halb(x):
6      print(x / 2)
7
8
9  a = halb(10)
10 b = sag_halb(10)
11 print(a, b)
12 print(halb(halb(8)))
```

Was erscheint im Terminal? Erst denken, dann testen!

Warum ist `b` nicht `5.0`?

Aufgabe 3 ⭐ Das Maximum

Schreibe eine Funktion `maximum(a, b)`, die die größere Zahl zurückgibt. Diese Tests müssen bestehen:

```
assert maximum(3, 8) == 8
assert maximum(8, 3) == 8
assert maximum(5, 5) == 5
```

Schreibe dann `maximum3(a, b, c)` für drei Zahlen. Tipp: Sie darf `maximum` aufrufen, sogar zweimal. Welche Punktzahl ist die höchste von 17, 23 und 19? _____

Aufgabe 4 ⭐⭐ Der Abstand

Der Abstand zweier Zahlen ist immer positiv: Von 3 bis 10 sind es 7, von 10 bis 3 auch. Schreibe `abstand(a, b)` und teste sie mit `assert` – auch mit gleichen und mit negativen Zahlen. Wie groß ist der Unterschied zwischen 21 Grad am Tag und −2 Grad in der Nacht? _____

Aufgabe 5 ⭐⭐ Die Note

Schreibe `note(prozent)` nach dieser Tabelle und teste besonders die Grenzen (85 und 84, 40 und 39 …). Frage dann mit `input` nach den Prozent und gib die Note aus.

ab Prozent	85	70	55	40	20	darunter
Note	1	2	3	4	5	6

Aufgabe 6 ⭐⭐⭐ Zwei Ergebnisse

Eine Funktion kann mehrere Werte zurückgeben: `return a, b`. Schreibe `kleinste_und_groesste(a, b, c)` und benutze sie so: `klein, gross = kleinste_und_groesste(14, 3, 9)`. Berechne damit die Spannweite (größte minus kleinste Zahl).

Das habe ich gelernt:

- ☐ Ich kann eine Funktion schreiben, die mit `return` ein Ergebnis zurückgibt.
- ☐ Ich kenne den Unterschied zwischen `return` und `print`.
- ☐ Ich kann Funktionen schreiben, die `True` oder `False` liefern.
- ☐ Ich kann meine Funktionen mit `assert` an Grenzfällen testen.

Mehr dazu im Buch: Kapitel 12 – Funktionen mit Rückgabewert

Name: _____ Klasse: _____ Datum: _____

Listen 1: Messwerte und Bestenliste

Ziel: Du speicherst viele Werte in einer Liste, sammelst neue Werte mit `append`, wertest die Liste mit `len`, `sum`, `min`, `max` aus und sortierst sie mit `sorted`.

So funktioniert's: Eine **Liste** steht in eckigen Klammern: `werte = [14, 17, 21]`. `werte[0]` ist das erste Element (der **Index** beginnt bei 0), `werte[-1]` das letzte. `werte.append(9)` hängt hinten an, `werte.pop()` nimmt das letzte heraus. `len`, `sum`, `min`, `max` zählen, addieren und suchen; `sorted(werte)` liefert eine sortierte **neue** Liste.

Aufgabe 1 ⭐ Eine Woche Wetter

```
1 # Hoechsttemperaturen Mo bis So
2 temperaturen = [14, 17, 21, 19, 12, 16, 18]
3
4 print(temperaturen[0])
5 print(temperaturen[-1])
6 print(len(temperaturen))
7 print(max(temperaturen), min(temperaturen))
8 print(sum(temperaturen) / len(temperaturen))
```

Tippe ab und teste. Was bedeuten die Ausgaben?

Zeile 4: _____
Zeile 5: _____
Zeile 8: _____

Wie holst du die Temperatur vom Mittwoch? _____

Aufgabe 2 ⭐ Erst denken, dann testen

Schreibe neben jede `print`-Zeile, was sie ausgibt. Prüfe danach im Emulator.

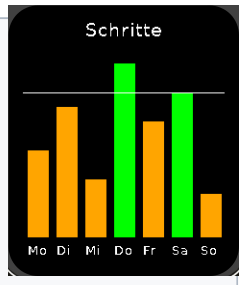
```
1 punkte = [7, 3, 9]
2 punkte.append(5)
3 print(punkte)
4 print(punkte[1] + punkte[3])
5 print(sorted(punkte))
6 letzter = punkte.pop()
7 print(letzter, punkte)
8 print(9 in punkte)
```

Zeile 3: _____
Zeile 4: _____
Zeile 5: _____
Zeile 7: _____
Zeile 8: _____

Aufgabe 3 ⭐ Messwerte sammeln

Miss bei fünf Personen den Puls. Dein Programm fragt die fünf Werte mit `input` ab und sammelt sie mit `append` in der Liste `puls = []`. Danach gibt es die Liste, den höchsten und niedrigsten Wert und den Durchschnitt aus.

Aufgabe 4 ⭐⭐ Schrittzähler als Diagramm




```
import pbc

tage = ["Mo", "Di", "Mi", "Do", "Fr", "Sa", "So"]
schritte = [6, 9, 4, 12, 8, 10, 3]   # in Tausend

pbc.clear()
for i in range(len(schritte)):
    x = 20 + i * 30
    hoehe = schritte[i] * 15
    pbc.fill_rect(x, 240 - hoehe, 22, hoehe, pbc.ORANGE)
```

1. Tippe ab und teste. Schreibe dann mit `pbc.text12` unter jeden Balken den Tag aus der Liste `tage` (y = 245).
2. Tage mit mindestens 10 000 Schritten werden grün.
3. Zeichne eine weiße Ziellinie bei 10 000 Schritten. Bei welchem y liegt sie? _____

Aufgabe 5 ⭐⭐ Bestenliste im 100-m-Lauf

Die Zeiten der Klasse: `zeiten = [13.2, 12.8, 14.1, 12.5, 13.7]` (in Sekunden). Hier ist **kleiner** besser. Gib die Plätze 1 bis 3 so aus: `Platz 1: 12.5 s`. Brauchst du `sorted(zeiten)` oder `sorted(zeiten, reverse=True)`? _____

Aufgabe 6 ⭐⭐⭐ Neue Zeiten eintragen

Frage im Terminal so lange nach neuen Zeiten (`float(input(…))`), bis die Eingabe leer ist. Jede neue Zeit kommt in die Liste; die Liste wird sortiert und behält nur die besten 5 (`pop()`). Melde den Platz (`zeiten.index(neu) + 1`) und zeige die Bestenliste auf dem Display, die neue Zeit grün.

Das habe ich gelernt:

- ☐ Ich kann eine Liste anlegen und mit dem Index auf Elemente zugreifen.
- ☐ Ich kann eine Liste mit `append` Schritt für Schritt füllen.
- ☐ Ich kann eine Liste mit `len`, `sum`, `min`, `max` auswerten.
- ☐ Ich kann eine Liste auf- und absteigend sortieren.

Mehr dazu im Buch: Kapitel 13 – Listen

Name: _____ Klasse: _____ Datum: _____

Listen 2: Merk dir die Folge

> **Ziel:** Du programmierst das Gedächtnisspiel „Merk dir die Folge“: Der PicoBoy zeigt eine immer längere Folge von Feldern, du spielst sie mit dem Joystick nach. Die Folge steht in einer **Liste**. (2 Stunden)



Vorbereitung: Öffne `ab16_start.py` (**Dateien → Vom Computer…**). Das Programm enthält schon fertige Funktionen:

- `zeichne_alle()` zeichnet die vier Felder.
- `leuchte(nr)` lässt Feld `nr` aufleuchten, mit Ton.
- `lies_taste()` wartet auf den Joystick und **liefert** die Feldnummer.
- `zeige_text(text)` schreibt unten einen Text.

Feldnummern: **0** oben · **1** rechts · **2** unten · **3** links. Du arbeitest nur unten im Programm, ab Zeile 45:

ab16_start.py

```
45 # ---------- Ab hier arbeitest du ----------
46 zeichne_alle()
47 folge = [0, 2, 1]
48 zeige_text("Merk dir die Folge!")
49 time.sleep_ms(800)
50 leuchte(folge[0])
51 leuchte(folge[1])
52 leuchte(folge[2])
53 zeige_text("Fertig")
```

Stunde 1: Die Folge zeigen

Aufgabe 1 ⭐ **Testen und verlängern** Teste das Programm. In welcher Reihenfolge leuchten die Farben? _____ Verlängere die Folge auf 5 Felder. Was passiert mit `leuchte(folge[5])`? _____

Aufgabe 2 ⭐ **Eine Schleife statt vieler Zeilen** Ersetze alle `leuchte`-Zeilen durch eine `for`-Schleife über die Liste `folge`. Mach daraus eine Funktion `spiele_folge(folge)` und schreibe sie zu den anderen Funktionen über Zeile 45. Teste mit einer Folge aus 6 Feldern.

Aufgabe 3 ⭐ Eine zufällige Folge Beginne mit der leeren Liste `folge = []` und hänge in einer Schleife viermal `random.randint(0, 3)` an. Gib die Liste mit `print(folge)` im Terminal aus, dann kannst du prüfen, ob die Felder richtig leuchten.

Aufgabe 4 ⭐ Erst denken, dann testen

```
1 folge = [2, 0]
2 folge.append(3)
3 folge.append(1)
4 print(len(folge))
5 print(folge[1], folge[-1])
6 for i in range(len(folge)):
7     print(i, folge[i])
```

Was gibt das Programm aus?

Stunde 2: Nachspielen und das ganze Spiel

Aufgabe 5 ⭐ Hat der Spieler recht? Die Funktion soll `True` liefern, wenn der Spieler die ganze Folge richtig nachspielt, sonst `False`. Ergänze die Bedingung und teste mit `folge = [0, 2, 1]`: Zeige „Richtig!“ oder „Leider falsch“.

```
1 def spieler_ist_richtig(folge):
2     for nr in folge:
3         taste = lies_taste()
4         leuchte(taste)
5         if                    :
6             return False
7     return True
```

Warum steht `return True` **nach** der Schleife und nicht darin?

Aufgabe 6 ⭐⭐ Das ganze Spiel Baue die Spielschleife: Die Liste beginnt leer. In jeder Runde kommt mit `append` ein zufälliges Feld dazu, unten steht „Runde 1“, „Runde 2“ …, die Folge wird gezeigt, und der Spieler spielt nach. Bei einem Fehler endet das Spiel (`break`) und zeigt, wie viele Runden geschafft wurden. Tipp: Die Rundenzahl ist `len(folge)`.

Aufgabe 7 ⭐⭐⭐ Profi-Version Wähle aus: a) Jede Runde wird schneller – gib `leuchte` einen Parameter `dauer`. b) Nach dem Spiel startet Taste A ein neues; die Ergebnisse aller Spiele stehen in einer Liste, angezeigt wird der Rekord mit `max`. c) Ein tiefer Ton bei einem Fehler.

Das habe ich gelernt:

- ☐ Ich kann eine Liste mit `append` Schritt für Schritt verlängern.
- ☐ Ich kann eine Liste mit `for` durchlaufen und mit dem Index auf Elemente zugreifen.
- ☐ Ich kann einer Funktion eine Liste übergeben.
- ☐ Ich kann eine Funktion schreiben, die `True` oder `False` liefert.

Mehr dazu im Buch: Kapitel 13 – Listen

Name: _____ Klasse: _____ Datum: _____

Listen 3: Sternenregen

Ziel: Du lässt viele Sterne gleichzeitig fallen. Ihre Positionen stehen in Listen: erst in **parallelen Listen**, dann als **Listen in Listen**. Du erzeugst Listen mit einer **List Comprehension** und entfernst Sterne sicher aus einer Liste.

Aufgabe 1 ⭐ Drei Sterne, zwei Listen

ab17_start.py

```
 1 import pbc
 2 import time
 3
 4 xs = [40, 120, 200]
 5 ys = [0, 60, 120]
 6
 7 pbc.clear()
 8 while True:
 9     for i in range(len(xs)):
10         pbc.fill_rect(xs[i], ys[i], 4, 4, pbc.BLACK)
11         ys[i] += 3
12         if ys[i] > 275:
13             ys[i] = 0
14         pbc.fill_rect(xs[i], ys[i], 4, 4, pbc.WHITE)
15     time.sleep_ms(30)
```

Tippe ab (oder öffne `ab17_start.py`) und teste.

Stern Nummer `i` steht bei `xs[i]`, `ys[i]`. Listen, deren Elemente mit gleichem Index zusammengehören, heißen **parallele Listen**.

Was tun Zeile 10 und 14?

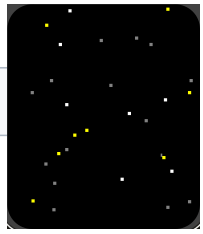
Was tun Zeile 12 und 13?

Füge einen vierten Stern hinzu. Was musst du ändern?

Aufgabe 2 ⭐ 30 Sterne in einer Zeile

Eine **List Comprehension** erzeugt eine ganze Liste nach einer Regel:

```
xs = [random.randint(0, 236) for i in range(30)]
```



„Die Liste aus 30 Zufallszahlen von 0 bis 236." Erzeuge so `xs` und `ys` (0 bis 275) für 30 Sterne. Denk an `import random`.

Gib jedem Stern außerdem ein eigenes Tempo von 1 bis 4 in einer dritten Liste `tempo` und benutze `ys[i] += tempo[i]`. Schnelle Sterne wirken näher: Färbe sie mit `FARBEN = [pbc.BLACK, pbc.GRAY, pbc.GRAY, pbc.WHITE, pbc.YELLOW]` und `FARBEN[tempo[i]]`.

Aufgabe 3 ⭐⭐ Ein Stern – eine Liste

Statt drei Listen bekommt jeder Stern eine kleine Liste `[x, y, tempo]`:

```
sterne = [[random.randint(0, 236), random.randint(0, 275), random.randint(1, 4)]
          for i in range(30)]
```

Schreibe die Spielschleife um: `for stern in sterne:` und `stern[1] += stern[2]`. Was ist `sterne[0][2]`? _____

Aufgabe 4 ⭐⭐ Gelandet ist gelandet

Ein Stern, der unten ankommt, soll jetzt aus der Liste **verschwinden**. Erst ein Test in der Konsole. Was gibt das Programm aus, und was sollte es ausgeben?

_____ statt _____

```
1 sterne_y = [280, 290, 100, 300]
2 for y in sterne_y:
3     if y > 275:
4         sterne_y.remove(y)
5 print(sterne_y)
```

Wird während der Schleife entfernt, rücken die Elemente nach vorn, und eines wird übersprungen. Abhilfe: über eine **Kopie** laufen, `for stern in sterne[:]:`, und mit `sterne.remove(stern)` im Original entfernen. Baue das ein, zähle die gelandeten Sterne und zeige `len(sterne)` an. Mit Taste A kommt oben ein neuer Stern dazu (`append`). Die Schleife läuft `while len(sterne) > 0:`.

Warum wird beim Durchlaufen der Kopie kein Stern übersprungen?

Aufgabe 5 ⭐⭐⭐ Ausweichen!

Unten fliegt ein Raumschiff (ein Rechteck), das du mit dem Joystick nach links und rechts steuerst. Sterne, die unten ankommen, starten wieder oben. Berührt ein Stern das Schiff, ist das Spiel vorbei. Zeige, wie viele Sekunden du überlebt hast (`time.ticks_ms`). Plane zuerst: Wann berührt ein 4 × 4 Pixel großer Stern bei (x, y) ein Schiff bei (`schiff_x`, `SCHIFF_Y`) mit 20 × 12 Pixeln? Schreibe die Bedingung auf:

Das habe ich gelernt:

- ☐ Ich kann viele Objekte mit parallelen Listen oder Listen in Listen bewegen.
- ☐ Ich kann eine Liste mit einer List Comprehension erzeugen.
- ☐ Ich weiß, warum ich beim Entfernen über eine Kopie laufe.

Mehr dazu im Buch: Kapitel 14 – Listen im Spiel

Name: _____ Klasse: _____ Datum: _____

Der Highscore bleibt

Ziel: Du speicherst einen Bestwert in einer Datei und liest ihn beim nächsten Start wieder ein. So vergisst dein Programm nichts mehr.

So funktioniert's:

```
1 import os
2 with open("rekord.txt", "w") as datei:    # "w": schreiben, Altes wird gelöscht
3     datei.write(str(punkte))              # nur Text: Zahl mit str() umwandeln
4
5 if "rekord.txt" in os.listdir():          # gibt es die Datei schon?
6     with open("rekord.txt", "r") as datei:    # "r": lesen
7         rekord = int(datei.read())        # Text wieder in eine Zahl umwandeln
```

Modus `"a"` hängt Text hinten an, `"\n"` beendet eine Zeile. Dateien bleiben auf dem Gerät dauerhaft, im Emulator nur, bis du die Seite neu lädst.

Aufgabe 1 ⭐ Gemerkt ist gemerkt

Tippe Programm A ab und teste es. Lösche dann alles im Editor, tippe Programm B ab und teste es.

```
# Programm A
zahl = input("Lieblingszahl: ")
with open("zahl.txt", "w") as datei:
    datei.write(zahl)
print("Gespeichert!")
```

```
# Programm B
with open("zahl.txt", "r") as datei:
    zahl = int(datei.read())
print("Mal 2:", zahl * 2)
```

Woher kennt Programm B die Zahl? _____

Seite neu laden, nur B starten – welcher Fehler? _____

Aufgabe 2 ⭐ Was steht in der Datei? Erst überlegen, dann testen.

```
1 with open("spiel.txt", "w") as datei:
2     datei.write("Tor\n")
3 with open("spiel.txt", "a") as datei:
4     datei.write("Pfosten\n")
5     datei.write("Tor\n")
6 with open("spiel.txt", "r") as datei:
7     print(datei.read())
```

Ausgabe:

_____

_____

_____

Und mit `"w"` statt `"a"` in Zeile 3? _____

Aufgabe 3 ⭐ Reaktionstest mit Bestzeit

Öffne `ab18_start.py` (**Dateien → Vom Computer…**) und teste: Warte, bis der Bildschirm grün wird, und drücke dann A. Die Bestzeit wird aber noch nicht gespeichert. Ergänze die beiden Funktionen:




```python
def lade_bestzeit():
    # Aufgabe 3: Bestzeit aus der Datei lesen
    return 9999         # noch keine Bestzeit

def speichere_bestzeit(zeit):
    # Aufgabe 3: zeit in die Datei schreiben
    print("Speichern fehlt noch!")
```

- `lade_bestzeit()` liefert die Zahl aus `bestzeit.txt`, wenn es die Datei gibt, sonst `9999`.
- `speichere_bestzeit(zeit)` schreibt `zeit` in `bestzeit.txt`.

Teste mehrmals. Warum liefert `lade_bestzeit()` ohne Datei `9999` und nicht `0`?

Aufgabe 4 ⭐⭐ Wer hält den Rekord?

Bei einer neuen Bestzeit fragt das Programm mit `input` nach dem Namen. Speichere Zeit und Namen in zwei Zeilen der Datei `rekord.txt`. Beim Laden liest du sie mit zwei `datei.readline()` wieder ein (beim Namen mit `.strip()` das `\n` entfernen). Zeige an: „Bestzeit: 287 ms von Lea“. So sieht deine neue Funktion `speichere_bestzeit(zeit, name)` aus:

```
```

Aufgabe 5 ⭐⭐⭐ Alle Versuche

Hänge nach jedem Versuch die Zeit als neue Zeile an die Datei `alle_zeiten.txt` an (Modus `"a"`). Lies danach alle Zeilen in eine Liste und zeige, wie viele Versuche es schon gab und wie schnell du im Durchschnitt bist.

Das habe ich gelernt:

- ☐ Ich kann mit `open` und `with` eine Datei schreiben und lesen.
- ☐ Ich wandle Zahlen beim Speichern mit `str()` und beim Lesen mit `int()` um.
- ☐ Ich prüfe mit `os.listdir()`, ob es eine Datei schon gibt.
- ☐ Ich kenne den Unterschied zwischen den Modi `"w"` und `"a"`.

Mehr dazu im Buch: Kapitel 15 – Dateien: Daten dauerhaft speichern

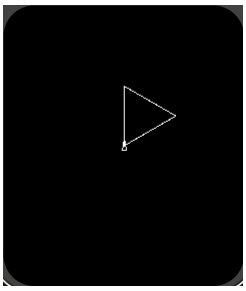
Name: _____ Klasse: _____ Datum: _____

Turtle 1: Vielecke mit Funktionen

Ziel: Du steuerst eine Turtle, die beim Laufen zeichnet, und schreibst eine Funktion, die jedes Vieleck zeichnen kann. Daraus baust du Rosetten und Sterne.

So funktioniert's:

```
1 import turtle
2
3 t = turtle.Turtle()
4 for i in range(3):
5     t.forward(60)
6     t.right(120)
```



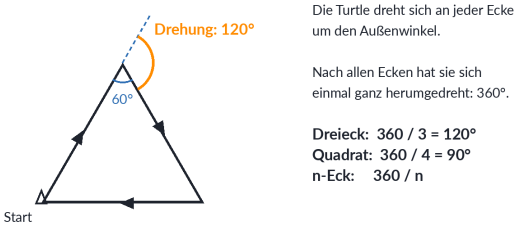
`turtle.Turtle()` legt eine Turtle an. Sie steht in der Mitte und schaut nach **oben**. `t.forward(60)` läuft 60 Pixel vorwärts und zeichnet, `t.right(120)` dreht sie um 120 Grad nach rechts (`t.left` nach links). Weitere Befehle: `t.back(d)`, `t.setPenColor("red")`, `t.speed(0)` (sofort zeichnen), `t.hide()` (Turtle verstecken).

Aufgabe 1 ⭐ Abtippen und verändern

Tippe das Programm ab und teste es. Ändere dann `t.right(120)` in `t.right(60)`. Was zeichnet die Turtle jetzt? _____

Aufgabe 2 ⭐ Wie weit muss sie sich drehen?

Nach allen Ecken hat sich die Turtle einmal ganz herumgedreht, also um 360 Grad. Ergänze die Tabelle und teste mindestens zwei Zeilen.



Ecken	3	4	5	6	8	10
Drehung	120°					

Regel: Drehung = _____

Aufgabe 3 ⭐ Eine Funktion für alle Vielecke

Ergänze die Lücken und teste. Zeichne damit ein Fünfeck (Seite 60), ein Sechseck (Seite 40) und ein Zehneck (Seite 20) in verschiedenen Farben.

```
import turtle

def vieleck(t, ecken, seite):
    winkel = ________
    for i in range(______):
        t.forward(seite)
        t.right(winkel)

t = turtle.Turtle()
vieleck(t, 5, 60)
```

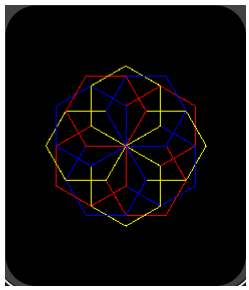
Aufgabe 4 ⭐⭐ Rosette

Lass die Turtle zwölfmal ein Sechseck (Seite 40) zeichnen und sich nach jedem Sechseck um 30 Grad nach rechts drehen. Benutze `t.speed(0)`.

Färbe die Sechsecke reihum mit einer Liste `FARBEN = ["red", "yellow", "blue"]` und `FARBEN[i % 3]`.

Warum sind es genau 12 Sechsecke bei 30 Grad?

_____

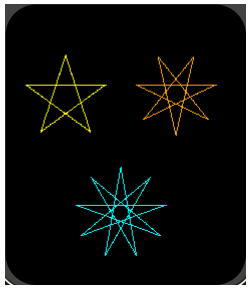


Aufgabe 5 ⭐⭐ Sterne

Ein Stern mit 5 Zacken: fünfmal `t.forward(80)` und `t.right(144)`. Wie oft dreht sich die Turtle dabei ganz herum? (Tipp: 5 · 144 = ?) _____

Schreibe eine Funktion `stern(t, zacken, laenge)` mit dem Drehwinkel `180 - 180 / zacken`. Teste sie mit 5, 7 und 9 Zacken. Was passiert bei 6 Zacken?

_____



Aufgabe 6 ⭐⭐⭐ Sternenhimmel

Zeichne 15 Sterne in zufälligen Größen und Farben an zufällige Stellen. `t.setPosition(x, y)` springt an die Stelle (x, y), ohne zu zeichnen; (0, 0) ist die Mitte, y zählt nach **oben**. Mit `t.speed(0)` geht es schneller. Du brauchst `import random`.

Das habe ich gelernt:

- ☐ Ich kann eine Turtle anlegen und mit `forward`, `right` und `left` steuern.
- ☐ Ich weiß, warum sich die Turtle bei einem n-Eck um 360 / n Grad dreht.
- ☐ Ich kann eine Funktion mit Parametern schreiben, die jedes Vieleck zeichnet.
- ☐ Ich kann aus einer Funktion Muster wie Rosetten und Sterne bauen.

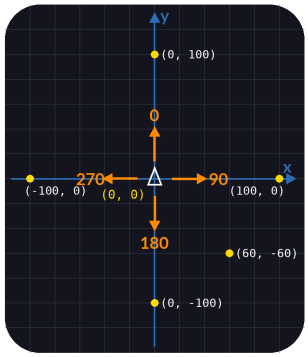
Mehr dazu im Buch: Kapitel 16 – Turtle-Grafik

Name: _____ Klasse: _____ Datum: _____

Turtle 2: Die Turtle sieht Farben

> **Ziel:** Ein Turtle-Roboter fährt durch ein Labyrinth. Mit `on_color` erkennt er rote Wände und die grüne Ziellinie, mit `setDirection` und `setPosition` stellst du Richtung und Start ein.

So funktioniert's: Öffne `ab20_start.py` (**Dateien → Vom Computer…**). Eine versteckte Turtle `maurer` zeichnet das Labyrinth aus roten Linien. Die zweite Turtle `robo` fährt los:



ab20_start.py

```
# Der Roboter startet in der Mitte.
robo.setPenColor("yellow")
robo.speed(0)
robo.setDirection(90)
while not robo.on_color("red"):
    robo.forward(1)
```

`robo.on_color("red")` ist `True`, wenn das Pixel **3 Pixel vor** der Turtle genau rot ist. `robo.setDirection(90)` lässt die Turtle nach rechts schauen.

Aufgabe 1 ⭐ Bis zur Wand

Teste das Programm. Wo bleibt der Roboter stehen? _____

Was bedeutet Zeile 41 in Worten? „Solange vor dem Roboter _____“

Was passiert, wenn du in Zeile 42 `forward(1)` durch `forward(10)` ersetzt? Probiere es aus und erkläre. _____

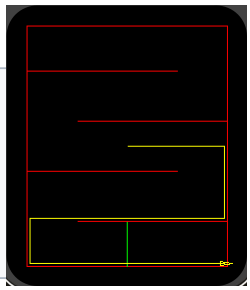
Aufgabe 2 ⭐ Abbiegen

Ergänze unten im Programm: Der Roboter dreht sich an der Wand um 90 Grad nach rechts und fährt bis zur nächsten Wand. Kopiere dazu die Schleife.

In welche Richtung schaut er nach `setDirection(180)`? _____ Und nach `setDirection(270)`? _____

Aufgabe 3 ⭐ Der Weg als Liste

Ersetze Zeile 38 bis 42 durch eine Funktion und eine Liste mit dem Weg:



```
def fahre_bis_wand(t):
    while not t.on_color("red"):
        t.forward(1)

ABBIEGEN = ["rechts", "rechts", "links", "links"]
```

Schreibe eine `for`-Schleife über `ABBIEGEN`: bis zur Wand fahren, dann je nach Eintrag `right(90)` oder `left(90)`. Nach der Schleife fährt der Roboter noch einmal bis zur Wand. Er fährt über die grüne Ziellinie.

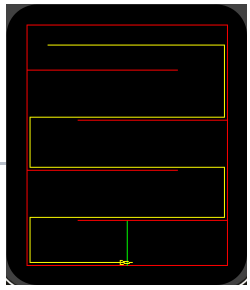
Aufgabe 4 ⭐⭐ Ziel erkannt!

Der Roboter soll auch an der **grünen** Linie anhalten. Ändere die Bedingung der `while`-Schleife in `fahre_bis_wand` mit `and`. Lass die Funktion außerdem die Schritte zählen und mit `return` zurückgeben. Gib am Ende aus: `Ziel erreicht nach … Schritten!`

Wie viele Schritte braucht dein Roboter? _____

Aufgabe 5 ⭐⭐ Neuer Start

Starte den Roboter oben links: `robo.setPosition(-80, 100)`, Blick nach rechts. Wie muss die Liste `ABBIEGEN` jetzt aussehen?



_____

Aufgabe 6 ⭐⭐⭐ Und wieder zurück

Am Ziel dreht sich der Roboter um (`right(180)`) und fährt mit einer neuen Farbe zurück in den Startgang. Schreibe eine Funktion `rueckweg(abbiegen)`, die aus der Liste den Rückweg berechnet. Überlege: In welcher Reihenfolge kommen die Abzweigungen, und was wird aus „rechts“?

Das habe ich gelernt:

- ☐ Ich kann mit `on_color` prüfen, welche Farbe vor der Turtle liegt.
- ☐ Ich kann die Richtung mit `setDirection` einstellen (0 oben, 90 rechts, 180 unten, 270 links).
- ☐ Ich kann eine Turtle mit `setPosition` an eine Stelle springen lassen.
- ☐ Ich kann einen Weg als Liste speichern und mit einer Schleife abfahren.

Mehr dazu im Buch: Kapitel 16 – Turtle-Grafik

Name: _____ Klasse: _____ Datum: _____

Canvas und Sprites

Ziel: Du zeichnest Bilder flimmerfrei im Speicher (Canvas), setzt einen Hintergrund und steuerst eine Figur (Sprite) mit dem Joystick über den Bildschirm.

Vorbereitung: `heissluftballon.png` hochladen: **Dateien → Bilder verwalten… → PNG hochladen…**

Canvas: `c = pbc.Canvas(240, 280)` legt ein Bild im Speicher an. Darauf zeichnest du mit `c.fill(farbe)`, `c.fill_rect(…)`, `c.ellipse(…, farbe, True)`, `c.text18(…)`. Erst `pbc.show(c)` zeigt es. **Sprite:** `s = pbc.sprite("bild.png")` steht unsichtbar bei (0, 0); `s.show()`, `s.set_to(x, y)`, `s.set_by(dx, dy)`; lesen: `s.x`, `s.y`, `s.width`, `s.height`.

Aufgabe 1 ⭐ Ein Bild im Speicher

```
1 import pbc
2
3 nacht = pbc.Canvas(240, 280)
4 nacht.fill(pbc.rgb(0, 0, 60))
5 nacht.ellipse(170, 60, 30, 30, pbc.YELLOW, True)
6 nacht.ellipse(158, 52, 26, 26, pbc.rgb(0, 0, 60), True)
7 nacht.fill_rect(0, 230, 240, 50, pbc.rgb(0, 80, 0))
8 nacht.text18("Gute Nacht!", 20, 240, pbc.WHITE)
9 pbc.show(nacht)
```



Tippe ab und teste. Wie entsteht die Mondsichel? _____

Was passiert, wenn du Zeile 9 löschst? _____

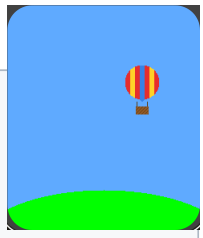
Aufgabe 2 ⭐ Schluss mit Flackern

Die Sonne wandert, aber es flackert (`ab21_flackern.py`). Baue das Programm auf einen Canvas `bild` um: vor der Schleife anlegen, darauf zeichnen, dann `pbc.show(bild)`. Auf dem Canvas: `bild.fill(…)` statt `pbc.clear(…)`, `True` statt `fill=True`. Notiere rechts, welche Zeilen sich ändern und welche dazukommen.

```
1  import pbc
2  import time
3
4  x = 0
5  while x < 240:
6      pbc.clear(pbc.CYAN)
7      pbc.fill_rect(0, 220, 240, 60, pbc.GREEN)
8      pbc.ellipse(x, 80, 25, 25, pbc.YELLOW, fill=True)
9      x = x + 3
10     time.sleep_ms(20)
```

Aufgabe 3 ⭐ Ballonfahrt

Öffne `ab21_start.py`. Der wichtigste Teil:



```
himmel = pbc.Canvas(240, 280)
himmel.fill(pbc.rgb(100, 170, 255))
himmel.ellipse(120, 300, 160, 70, pbc.GREEN, True)
pbc.Sprite._background = himmel
pbc.show(himmel)

ballon = pbc.sprite("heissluftballon.png")
ballon.set_to(96, 100)
ballon.show()
```

Teste. Unten in der Schleife steuern links und rechts den Ballon. Ergänze hoch und runter.

Was ändert sich ohne `ballon.show()`? _____

Aufgabe 4 ⭐⭐ Nicht aus dem Bild fliegen

Der Ballon ist 48 Pixel breit und 64 hoch. Ergänze die Lücken, damit er den Bildschirm nicht verlässt:

```
if pbc.pressed_left() and ballon.x > 1:
if pbc.pressed_right() and ballon.x + ballon._____ < pbc.WIDTH - 1:
if pbc.pressed_up() and ballon._____ > 1:
if pbc.pressed_down() and ballon.y + ballon._____ < pbc._____ - 1:
```

Aufgabe 5 ⭐⭐ Wo ist der Text?

Jemand schreibt in Aufgabe 3 nach `ballon.show()` die Zeile `pbc.text18("Ballonfahrt", 70, 20, pbc.WHITE)`. Der Text erscheint, ist aber weg, sobald sich der Ballon bewegt. Warum? Wie geht es richtig?

_____

_____

_____

Aufgabe 6 ⭐⭐⭐ Wind

Der Wind treibt den Ballon in jedem Durchlauf 1 Pixel nach rechts. Ist er rechts hinaus, kommt er links wieder herein (`set_to(-ballon.width, ballon.y)`). Den Joystick nach rechts braucht man dann nicht mehr.

Das habe ich gelernt:

- ☐ Ich kann auf einem Canvas zeichnen und ihn mit `pbc.show` zeigen.
- ☐ Ich kann eine Animation flimmerfrei machen.
- ☐ Ich kann ein Sprite anlegen, zeigen und mit dem Joystick bewegen.
- ☐ Ich weiß, warum Texte in den Hintergrund-Canvas gehören.

Mehr dazu im Buch: Kapitel 17 – Der Canvas und Kapitel 18 – Sprites

Name: _____ Klasse: _____ Datum: _____

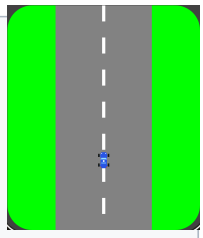
Kollisionen

> **Ziel:** Du baust ein kleines Rennspiel. Dein Auto bleibt im Bildschirm, muss zurück, wenn es ins Gras fährt, sammelt Sterne und weicht Hütchen aus.

Vorbereitung: `rennauto.png`, `stern.png` und `huetchen.png` hochladen (**Dateien → Bilder verwalten… → PNG hochladen…**). Öffne dann `ab22_start.py`.

Befehl	ist `True`, wenn …
`a.touches(b)`	sich die **Rechtecke** der Bilder von Sprite `a` und Sprite `b` überlappen
`s.touches_edge()`	ein Teil des Sprites **außerhalb** des Bildschirms liegt
`s.on_color(farbe)`	ein sichtbares Pixel des Sprites über genau dieser Farbe im **Hintergrund-Canvas** steht

Aufgabe 1 ⭐ Das Startprogramm verstehen



```
while True:
    dx = 0
    dy = 0
    if pbc.pressed_left():
        dx = -3
    if pbc.pressed_right():
        dx = 3
    if pbc.pressed_up():
        dy = -3
    if pbc.pressed_down():
        dy = 3
    auto.set_by(dx, dy)
    # Hier kommen die Kollisionen hin.

    time.sleep_ms(20)
```

Teste. Welchen Wert haben `dx` und `dy`, wenn du keine Taste drückst? _____

Fahre nach oben aus dem Bildschirm. Was passiert? _____

Aufgabe 2 ⭐ Rand

Ersetze den Kommentar in Zeile 13 durch diese Zeilen und teste:

```
    if auto.touches_edge():
        auto.set_by(-dx, -dy)
```

Erkläre: Warum bleibt das Auto jetzt am Rand stehen?

Aufgabe 3 ⭐ Gras

Das Gras ist in `strecke` mit `pbc.GREEN` gemalt. Ergänze: Fährt das Auto ins Gras, erscheint im Terminal „Gras!“ und es muss zurück zum Start (113, 240).

Aufgabe 4 ⭐⭐ Sterne sammeln

Lege vor der Schleife ein Sprite `stern` an einer zufälligen Stelle auf der Straße an (x von 60 bis 162). Berührt das Auto den Stern, gibt es einen Punkt, die Punkte stehen im Terminal, und der Stern springt an eine neue Stelle.

Aufgabe 5 ⭐⭐ Hütchen

Ein Hütchen (`huetchen.png`) kommt in jedem Durchlauf 2 Pixel von oben entgegen. Ist es unten hinaus, startet es oben neu bei y = −18. Berührt das Auto das Hütchen, endet das Spiel mit „Unfall!“ und dem Punktestand.

Manchmal gibt es einen „Unfall“, obwohl sich Auto und Hütchen gar nicht berühren. Warum?

Aufgabe 6 ⭐⭐⭐ Punkte auf dem Bildschirm

Zeige die Punkte links oben im Gras statt im Terminal. Male dafür vorher ein grünes Rechteck über die alte Zahl, schreibe mit `strecke.text18(…)` und zeige den Hintergrund neu mit `pbc.show(strecke)`. Wozu ist das grüne Rechteck nötig? _____

Das habe ich gelernt:

- ☐ Ich kann prüfen, ob ein Sprite am Rand ist, und es dort anhalten.
- ☐ Ich kann mit `on_color` auf Farben im Hintergrund reagieren.
- ☐ Ich kann mit `touches` Dinge einsammeln und Zusammenstöße erkennen.
- ☐ Ich weiß, dass `touches` mit Rechtecken rechnet.

Mehr dazu im Buch: Kapitel 19 – Kollisionen

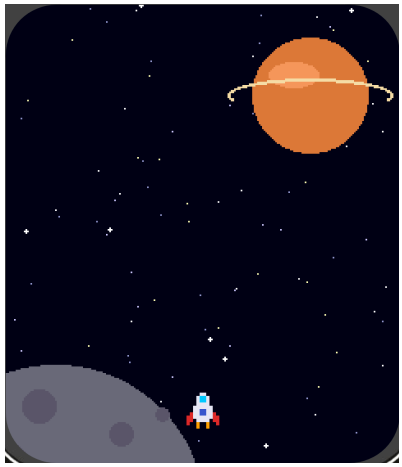
Name: _____ Klasse: _____ Datum: _____

Projekt Weltraumspiel 1: Raumschiff und Sterne

Ziel: In drei Stunden baust du ein Weltraumspiel: Ein Raumschiff weicht Meteoren aus. Heute entstehen der Hintergrund und das Raumschiff, das du mit einer eigenen Funktion steuerst.

Vorbereitung: Lade die Bilder `weltraum.png`, `raumschiff.png`, `meteor.png` und `stern.png` über **Dateien → Bilder verwalten…** hoch. Öffne dann `ab23a_start.py` mit **Dateien → Vom Computer…**

```
1  # Weltraumspiel – Teil 1
2  import pbc
3  import time
4
5  TEMPO = 5
6
7  # Hintergrund
8  himmel = pbc.load_image("weltraum.png")
9  pbc.Sprite._background = himmel
10 pbc.show(himmel)
11
12 # Raumschiff
13 schiff = pbc.sprite("raumschiff.png")
14 schiff.set_to(110, 235)
15 schiff.show()
16
17 # Spielschleife
18 while True:
19     if pbc.pressed_left():
20         schiff.set_by(-TEMPO, 0)
21     if pbc.pressed_right():
22         schiff.set_by(TEMPO, 0)
23     time.sleep_ms(20)
```



Aufgabe 1 ⭐ Starten und ausprobieren

Teste das Programm und steuere mit dem Joystick (Pfeiltasten). Was passiert, wenn du lange nach links drückst? _____

Aufgabe 2 ⭐ Was passiert, wenn …

Überlege erst, dann probiere aus (und mach die Änderung danach rückgängig).

- … du Zeile 15 löschst? _____
- … du Zeile 9 löschst? _____
- … du in Zeile 5 `TEMPO = 15` schreibst? _____

Aufgabe 3 ⭐ Steuerung in eine Funktion

Das Spiel wird später lang. Damit die Spielschleife übersichtlich bleibt, kommt die Steuerung in eine Funktion. Ergänze die Lücken und schreibe die Funktion **über** `# Hintergrund`. Das Raumschiff ist 20 Pixel breit.

```
def steuere(schiff):
    if pbc.pressed_left() and schiff.x > ____:
        schiff.set_by(-TEMPO, 0)
    if pbc.pressed_right() and schiff.x < pbc.WIDTH - schiff.____:
        schiff.set_by(TEMPO, 0)
```

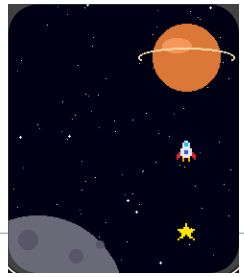
Ersetze dann die vier `if`-Zeilen in der Spielschleife durch einen Aufruf der Funktion: _____ Teste: Bleibt das Schiff jetzt an beiden Rändern stehen?

Aufgabe 4 ⭐⭐ Hoch und runter

Erweitere `steuere`: Mit Joystick hoch und runter fliegt das Schiff nach oben und unten, aber nur in der unteren Hälfte des Bildschirms (y zwischen 140 und 250). Du brauchst `pbc.pressed_up()`, `pbc.pressed_down()` und `schiff.y`.

Aufgabe 5 ⭐⭐⭐ Sterne sammeln

Ein Stern (`stern.png`) erscheint an einer zufälligen Stelle in der unteren Hälfte. Berührt das Raumschiff ihn (`schiff.touches(stern)`), ertönt ein kurzer hoher Ton und der Stern springt an eine neue zufällige Stelle. Schreibe dafür eine Funktion `neuer_platz(stern)`, die `random.randint` benutzt. Denke an `import random`!



Das habe ich gelernt:

- ☐ Ich kann ein Bild als Hintergrund für Sprites festlegen.
- ☐ Ich kann ein Sprite anlegen, zeigen und mit dem Joystick bewegen.
- ☐ Ich kann die Steuerung in eine Funktion auslagern und am Rand stoppen.
- ☐ Ich weiß, dass ein Sprite erst nach `show()` zu sehen ist.

Mehr dazu im Buch: Kapitel 18 – Sprites, Kapitel 21 – Projekt: Ein Spiel Schritt für Schritt

Name: _____ Klasse: _____ Datum: _____

Projekt Weltraumspiel 2: Asteroiden aus der Liste

> **Ziel:** Asteroiden (im Programm: Meteore) fallen auf dein Raumschiff zu. Du verwaltest sie in einer Liste, erkennst Treffer und zählst die Leben.

Vorbereitung: Öffne `ab23b_start.py`. Es enthält die Lösung von AB 23a mit der Funktion `steuere`. Sind die Bilder noch hochgeladen? Wenn nicht: **Dateien → Bilder verwalten…**

Aufgabe 1 ⭐ Ein Meteor

Unter `# Aufgabe 1: Meteor anlegen`:

```
meteor = pbc.sprite("meteor.png")
meteor.set_to(random.randint(0, 220), -20)
meteor.show()
```

In der Spielschleife unter `# Aufgabe 1: Meteor bewegen` (eingerückt):

```
    meteor.set_by(0, TEMPO_METEOR)
    if meteor.y > pbc.HEIGHT:
        meteor.set_to(random.randint(0, 220), -20)
```

Warum startet der Meteor bei y = −20 und nicht bei 0? _____

Aufgabe 2 ⭐ Drei Meteore aus einer Liste

Statt einem Meteor sollen `ANZAHL = 3` Meteore fallen. Lege die Konstante oben an und schreibe die Funktion über `# Hintergrund`:

```
def nach_oben(meteor):
    meteor.set_to(random.randint(0, 220), random.randint(-150, -20))
```

Ersetze deinen Meteor aus Aufgabe 1 durch eine Liste. Ergänze die Lücken:

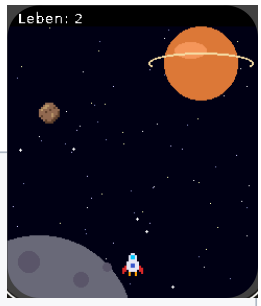
```
meteore = []
for i in range(________):
    meteor = pbc.sprite("meteor.png")
    nach_oben(meteor)
    meteor.show()
    meteore.________(meteor)
```

In der Spielschleife bewegt `for meteor in meteore:` jeden Meteor (rücke die Zeilen aus Aufgabe 1 ein und nimm `nach_oben(meteor)`). Warum fallen die Meteore versetzt und nicht alle gleichzeitig?

_____

Aufgabe 3 ⭐ Treffer!

Die Leben stehen oben auf dem Bildschirm. Weil die Sprites bei jeder Bewegung neu gezeichnet werden, kommt der Text auf das Hintergrundbild `himmel`, nicht direkt auf den Bildschirm. Schreibe diese Funktion über `# Hintergrund`:




```
def zeige_leben(leben):
    himmel.fill_rect(0, 0, 240, 20, pbc.BLACK)
    himmel.text12(f"Leben: {leben}", 10, 4, pbc.WHITE)
    pbc.show(himmel)
```

Lege vor der Spielschleife `leben = 3` an und zeige die Leben an. Prüfe in der `for`-Schleife mit `meteor.touches(schiff)`, ob ein Meteor das Schiff trifft. Dann: ein Leben weniger, ein tiefer Ton (`pbc.tone(200, 300)`), der Meteor startet neu, die Anzeige wird aktualisiert.

`touches` vergleicht die Rechtecke um die Bilder. Was bedeutet das für die Ecken?

Aufgabe 4 ⭐⭐ Game Over

Die Spielschleife soll nur laufen, solange `leben > 0` ist. Danach werden alle Sprites versteckt (`hide()`), und auf `himmel` erscheint in einem schwarzen Kasten groß „GAME OVER“. Vergiss `pbc.show(himmel)` nicht.

```

```

Aufgabe 5 ⭐⭐⭐ Jeder Meteor hat sein Tempo

Lege eine zweite Liste `tempi` an, die für jeden Meteor eine zufällige Geschwindigkeit von 4 bis 12 speichert. Tipp: Durchlaufe die Meteore mit `for i in range(len(meteore)):` und benutze `meteore[i]` und `tempi[i]`.

Das habe ich gelernt:

- ☐ Ich kann viele gleiche Sprites in einer Liste anlegen und mit `for` bewegen.
- ☐ Ich kann mit `touches` einen Treffer erkennen und Leben abziehen.
- ☐ Ich schreibe Text auf den Hintergrund, damit er nicht verschwindet.
- ☐ Ich kann ein Spiel mit `while leben > 0` enden lassen.

Mehr dazu im Buch: Kapitel 19 – Kollisionen, Kapitel 21 – Projekt: Ein Spiel Schritt für Schritt

Name: _____ Klasse: _____ Datum: _____

Projekt Weltraumspiel 3: Punkte und Highscore

Ziel: Dein Weltraumspiel bekommt Punkte, einen Endbildschirm und einen Highscore, der in einer Datei gespeichert wird. Danach baust du eine eigene Erweiterung ein.

Vorbereitung: Öffne `ab23c_start.py` (das Spiel aus AB 23b mit Leben und „GAME OVER“). Speichere jede funktionierende Fassung unter neuem Namen, z. B. `weltraum_1.py`, `weltraum_2.py` …

Aufgabe 1 ⭐ Punkte für jedes Ausweichen

Für jeden Meteor, der unten am Schiff vorbeifliegt, gibt es einen Punkt.



1. Ersetze `zeige_leben` durch diese Funktion und passe alle Aufrufe an:

```
def zeige_stand(punkte, leben):
    himmel.fill_rect(0, 0, 240, 20, pbc.BLACK)
    himmel.text12(f"Punkte: {punkte}", 10, 4, pbc.WHITE)
    himmel.text12(f"Leben: {leben}", 170, 4, pbc.WHITE)
    pbc.show(himmel)
```

2. Lege `punkte = 0` an. Wo zählst du den Punkt? Im Zweig `elif meteor.y > pbc.HEIGHT:` – ergänze dort `punkte += 1` und `zeige_stand(punkte, leben)`.
3. Schreibe die Punkte auch in den GAME-OVER-Kasten (`text18`).

Aufgabe 2 ⭐ Den Rekord speichern

Ergänze die Lücken und schreibe beide Funktionen über `# Hintergrund`. Oben brauchst du `import os` und die Konstante `DATEI = "weltraum_rekord.txt"`.

```
def lade_rekord():
    if DATEI in os.listdir():
        with open(DATEI, "____") as f:
            return ______(f.read())
    return 0


def speichere_rekord(wert):
    with open(DATEI, "____") as f:
        f.write(______(wert))
```

Am Spielende, nach dem Verstecken der Sprites, lädst du den alten Rekord mit `rekord = lade_rekord()`. Ist `punkte` größer, wird `rekord = punkte` und mit `speichere_rekord(rekord)` gespeichert. Zeige `rekord` im Endbildschirm an (`text12`).

Aufgabe 3 ⭐ Testen

Spiele zweimal hintereinander. Welcher Rekord steht beim zweiten Spiel da?

_____

Lade jetzt die Seite PBC+ Studio im Browser neu und spiele noch einmal. Was passiert mit dem Rekord, und warum?

_____

_____

Aufgabe 4 ⭐⭐ Deine Erweiterung

Wähle eine Idee (oder eine eigene), plane sie in zwei, drei Schritten und baue sie ein:

- **Bonusstern** (`stern.png`): fällt langsam, bringt 5 Punkte, wenn du ihn fängst.
- **Immer schneller:** Alle 10 Punkte fallen die Meteore schneller (Tipp: `tempo = TEMPO_METEOR + punkte // 10`).
- **Neustart:** Nach „GAME OVER“ startet Taste A ein neues Spiel.
- **Ufo** (`ufo.png`): fliegt waagrecht über den Bildschirm; eine Berührung kostet ein Leben.
- **Licht:** Die RGB-LED leuchtet bei einem Treffer kurz rot.

Meine Idee: _____

Schritt 1: _____

Schritt 2: _____

Schritt 3: _____

Aufgabe 5 ⭐⭐⭐ Zwei Erweiterungen

Baue eine zweite Idee aus Aufgabe 4 ein. Teste nach jedem Schritt. Was hat nicht sofort geklappt, und wie hast du den Fehler gefunden?

_____

_____

_____

Das habe ich gelernt:

☐ Ich kann Punkte und Leben gemeinsam anzeigen.

☐ Ich kann einen Highscore in einer Datei speichern und wieder laden.

☐ Ich weiß, dass Dateien im Emulator nur bis zum Neuladen der Seite bleiben.

☐ Ich kann ein Spiel planvoll in kleinen Schritten erweitern.

Mehr dazu im Buch: Kapitel 15 – Dateien, Kapitel 21 – Projekt: Ein Spiel Schritt für Schritt

Name: _____ Klasse: _____ Datum: _____

Projekt: Mein eigenes Programm

Ziel: Du planst ein eigenes Programm, baust es in kleinen Schritten, findest Fehler und stellst es am Ende der Klasse vor. Dieser Bogen ist dein Planungs- und Arbeitsbogen für die ganze Projektzeit.

So gehst du vor: Idee finden → Steckbrief → Stufenplan → bauen und nach jeder Stufe testen → Fehler notieren → aufräumen → vorstellen. Als Gerüst kannst du `ab24_vorlage.py` öffnen.

Aufgabe 1 ⭐ Eine Idee finden

Ideen: Würfel · Stoppuhr · Reaktionsduell · Zahlenraten · Ampel · Metronom · Mini-Klavier · Vokabeltrainer · Turtle-Muster · Uhr · Hindernislauf mit Sprites · Snake · Labyrinth · Wasserwaage (nur mit Gerät) · eigene Idee

Was soll dein Programm tun? (höchstens zwei Sätze)

_____

_____

Was ist die **kleinste** Fassung, die schon funktioniert?

_____

Aufgabe 2 ⭐ Der Steckbrief

Name: _____

Datei: _____

Bedienung: Was macht welche Taste?

◀▶ _____

▲▼ _____

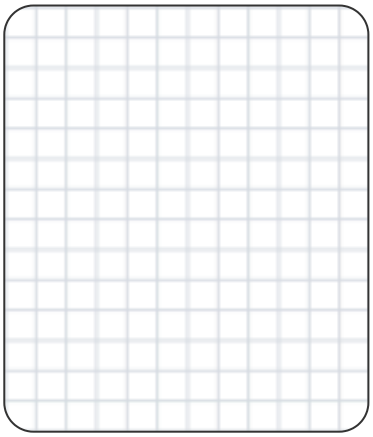
A _____

B _____

Konsole: _____

Bilder, Töne, Dateien: _____

Skizze des Bildschirms (1 Kästchen = 20 Pixel)



Aufgabe 3 ⭐ Der Stufenplan

Teile dein Programm in Stufen. Jede Stufe muss für sich laufen. Hake ab, wenn sie getestet ist.

Stufe	Was kommt dazu?	Fertig, wenn …	✓
1			
2			
3			
4			
Extra			

Aufgabe 4 ⭐ Bauen, testen, Fehler notieren

Speichere jede funktionierende Stufe unter eigenem Namen (`projekt_1.py`, `projekt_2.py` …). Notiere Fehler, die dich länger aufgehalten haben. Tipps: Fehlermeldung von unten lesen · `print("name:", name)` einbauen · eine Kopie verkleinern, bis nur der Fehler übrig ist.

Was ging schief? (Meldung oder Beobachtung)	Ursache	Wie gelöst?

Aufgabe 5 ⭐ Aufräumen und vorstellen

☐ läuft ohne Fehlermeldung
☐ Kopfkommentar: Name, Zweck, Bedienung
☐ sprechende Namen, Funktionen, Konstanten
☐ Test-`print` entfernt, keine Umlaute auf dem Display

Vorstellen (3–5 min): Was ist es? · Vorführen · ein Stück Code erklären · Was war schwierig? · Was kommt als Nächstes?

Aufgabe 6 ⭐⭐ / ⭐⭐⭐ Extras

Baue die Zeile „Extra“ aus deinem Stufenplan ein, ohne dass eine frühere Stufe kaputtgeht. Wer noch Zeit hat, plant und baut ein zweites Extra.

Das habe ich gelernt:

☐ Ich kann eine eigene Idee in Steckbrief und Stufenplan beschreiben.
☐ Ich baue in kleinen Schritten und teste nach jedem Schritt.
☐ Ich finde Fehler mit Fehlermeldung, `print` und Verkleinern.

Mehr dazu im Buch: Kapitel 22 – Dein eigenes Projekt