

Schüler-PDF - Learn Python with Turtle Graphics

Diese Anleitung begleitet dich durch sieben Levels, um die spannende Welt der Programmierung mit Python zu entdecken. Du wirst die Grundlagen dieser mächtigen Programmiersprache kennenlernen, indem du die Turtle (Schildkröte) durch Python-Code auf einem Canvas (Zeichenfläche) steuerst.

Lass uns deine aufregende Reise in die Python-Programmierung starten!

Inhaltsverzeichnis

1 Wie wird es ablaufen?	1
2 Programmierung mit Python	1
3 Turtle Graphics	4
4 Bedienungsanleitung Webanwendung	5
5 Programmierkonzepte	9

1 Wie wird es ablaufen?

In der Webanwendung gibt es sieben Level, die euch die Grundlagen der Programmierung in Python näher bringen, sowie Level 0, was ein bisschen vorbereitet, und Level Sandbox, wo man Sachen ausprobieren kann.

Das Ziel für jedes Level ist es, die Aufgaben zu lösen, indem Ihr die Turtle durch Programmierung das erwünschte Ergebnis zeichnen lässt. Jedes Level wird einem ein neue Grundlage gezeigt, die dann angewendet werden muss.

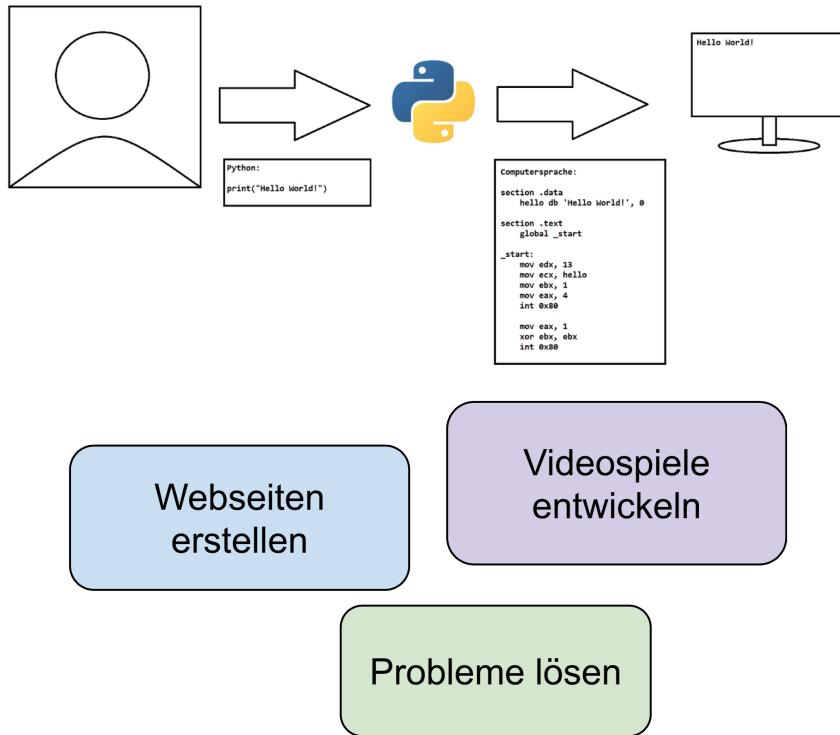
2 Programmierung mit Python



Was ist eine Programmiersprache?

Eine Programmiersprache ist wie ein Werkzeug für den Menschen, um mit dem Computer zu sprechen. Dieses Werkzeug ist dazu da, um dem Computer zu sagen, was es tun soll.

Wenn man eine Programmiersprache benutzt, gibt man dem Computer Anweisungen, damit verschiedene Aufgaben gelöst werden können, wie z.B.:



Ist die Reihenfolge im Code wichtig?

In der Programmierung ist die Reihenfolge der Anweisungen wichtig, damit am Ende alles funktioniert. Du möchtest ja nicht sagen, er soll zuerst den Text schreiben, und dann das Heft öffnen, da sonst alles übereinander stehend vorne auf dem Heft steht, sondern er soll ja zuerst das Heft öffnen, dann den Text schreiben, umblättern, und den Text zu Ende schreiben.

Es gibt viele verschiedene Programmiersprachen, die ihre eigenen Regeln und Möglichkeiten bieten. Manche Sprachen sind einfacher zu lernen und zu verwenden, andere bieten mehr Möglichkeiten, sind aber vielleicht etwas komplizierter.

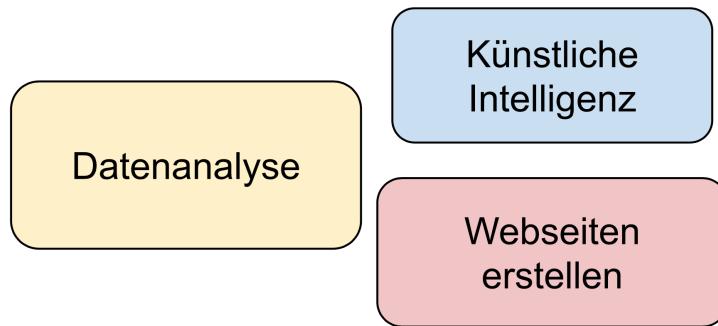
Einer der einfacheren Sprachen ist Python, worum es in diesem Projekt auch geht.



Aber wieso ist Python gut für Anfänger?

Python ist eine Sprache, die gut für Anfänger ist, da sie einfach zu lesen und zu verstehen ist. Die Art und Weise, wie der Code geschrieben wird, ähnelt der englischen Sprache sehr. Es gibt weniger Klammern und Sonderzeichen als in anderen Sprachen, was es einfacher macht, zu verstehen, was der Code macht.

Die Python-Sprache ist eine sehr beliebte Sprache für die vielen Ressourcen, die einem helfen könnten, sein Ziel zu erreichen. Es ist mit Python möglich, eine Vielzahl von Projekten umzusetzen:



Python bietet dem Nutzer auch ein schnelles Erfolgserlebnis, was in anderen Sprachen meist nicht so schnell zu sehen ist.



Gibt es auch Nachteile an Python?

Python kann langsamer sein als andere Programmiersprachen wie C, oder C++. Was für Standardanwendungen kein Problem sein sollte, aber für Anwendungen, die hohe Leistungen erfordern, ein Nachteil sein könnte. Der Speicherplatzverbrauch von Python ist auch höher als bei anderen Sprachen, was bei Geräten mit begrenztem Speicher zu Problemen führen könnte.

Die Grundlagen könnten für den Anfang euch vielleicht zu viel sein, aber durch häufige Wiederholung und aktive Programmierung werden diese euch später leicht fallen. Es ist normal anfangs, von dem vielen neuen Zeug überfordert zu sein. Nehmt euch Zeit und geht die Aufgaben in eurem Tempo durch, um die Inhalte wirklich zu verstehen.

3 Turtle Graphics

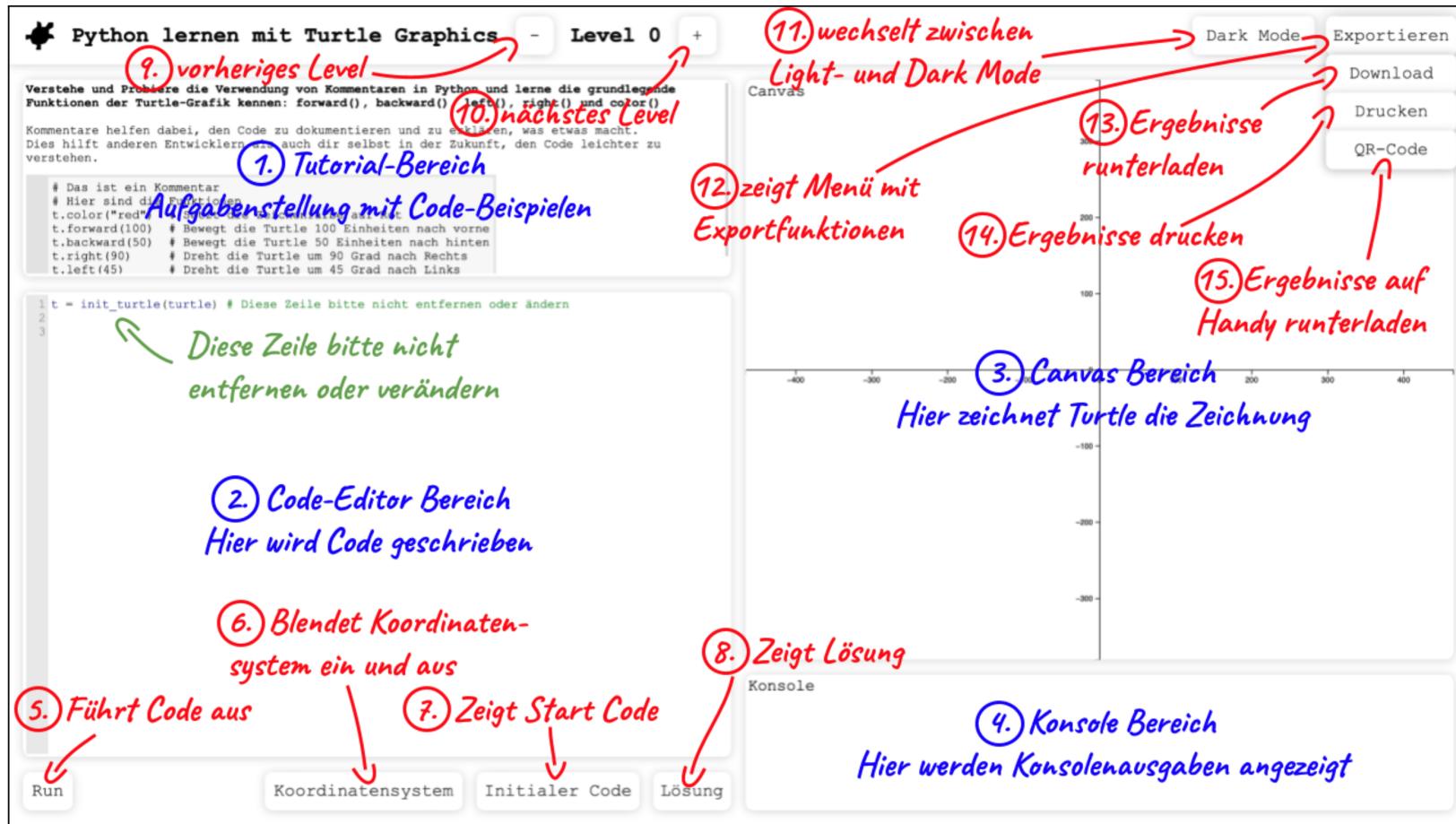
In dieser Webanwendung verwendet ihr Turtle Graphics, unterstützt durch die Python-Bibliothek "turtle". Turtle Graphics ermöglicht es euch, Zeichnungen mithilfe von Befehlen zu erstellen. Die Turtle (Schildkröte) fungiert dabei als euer virtueller Stift.

Hier ist eine einfache Tabelle mit einigen grundlegenden Turtle-Befehlen die du gleich verwenden wird. Mach dir keinen Kopf, wenn du beim Lesen verwirrt bist. In jedem Level werden die Funktionen die du im Level brauchst einzeln mit Beispielen erklärt.

Befehl	Beschreibung
t.forward(distance)	Bewegt die Turtle vorwärts um die angegebene Distanz.
t.backward(distance)	Bewegt die Turtle rückwärts um die angegebene Distanz.
t.left(angle)	Dreht die Turtle nach links um den angegebenen Winkel (in Grad).
t.right(angle)	Dreht die Turtle nach rechts um den angegebenen Winkel (in Grad).
t.color(color)	Setzt die Farbe für zukünftige Zeichnungen (z.B. ' <code>red</code> ' oder ' <code>#00FF00</code> ').
t.begin_fill()	Startet das Ausfüllen des Bereichs, der von zukünftigen Zeichnungen umschlossen wird.
t.end_fill()	Beendet das Ausfüllen des Bereichs und füllt ihn mit der ausgewählten Farbe.
t.width(width)	Setzt die Breite des Stifts für zukünftige Zeichnungen.
t.hideturtle()	Blendet die Turtle aus, sodass sie nicht mehr sichtbar ist.
t.showturtle()	Zeigt die Turtle an, wenn sie zuvor ausgeblendet wurde.
t.speed(speed)	Setzt die Zeichengeschwindigkeit der Turtle. Die Geschwindigkeit kann Werte von 0 bis 10 annehmen, wobei 1 die langsamste und 10 die zweit schnellste und 0 die schnellste Geschwindigkeit ist.
t.goto(x, y)	Bewegt die Turtle an die angegebene Position (x, y) im Zeichenfeld.
t.penup()	Hebt den Stift der Turtle an, sodass keine Spur hinterlassen wird.
t.pendown()	Senkt den Stift der Turtle, um eine Spur zu hinterlassen.

4 Bedienungsanleitung Webanwendung

In der folgenden Darstellung ist die Webanwendung zu sehen. Jedes Level hat eine Aufgabenstellung (1). Code wird im Code-Editor Bereich (2) geschrieben. Sobald der Run-Button (5) gedrückt wird, wird die im Code-Editor definierte Zeichnung auf dem Canvas (3) gezeichnet.



Bereiche (blau)

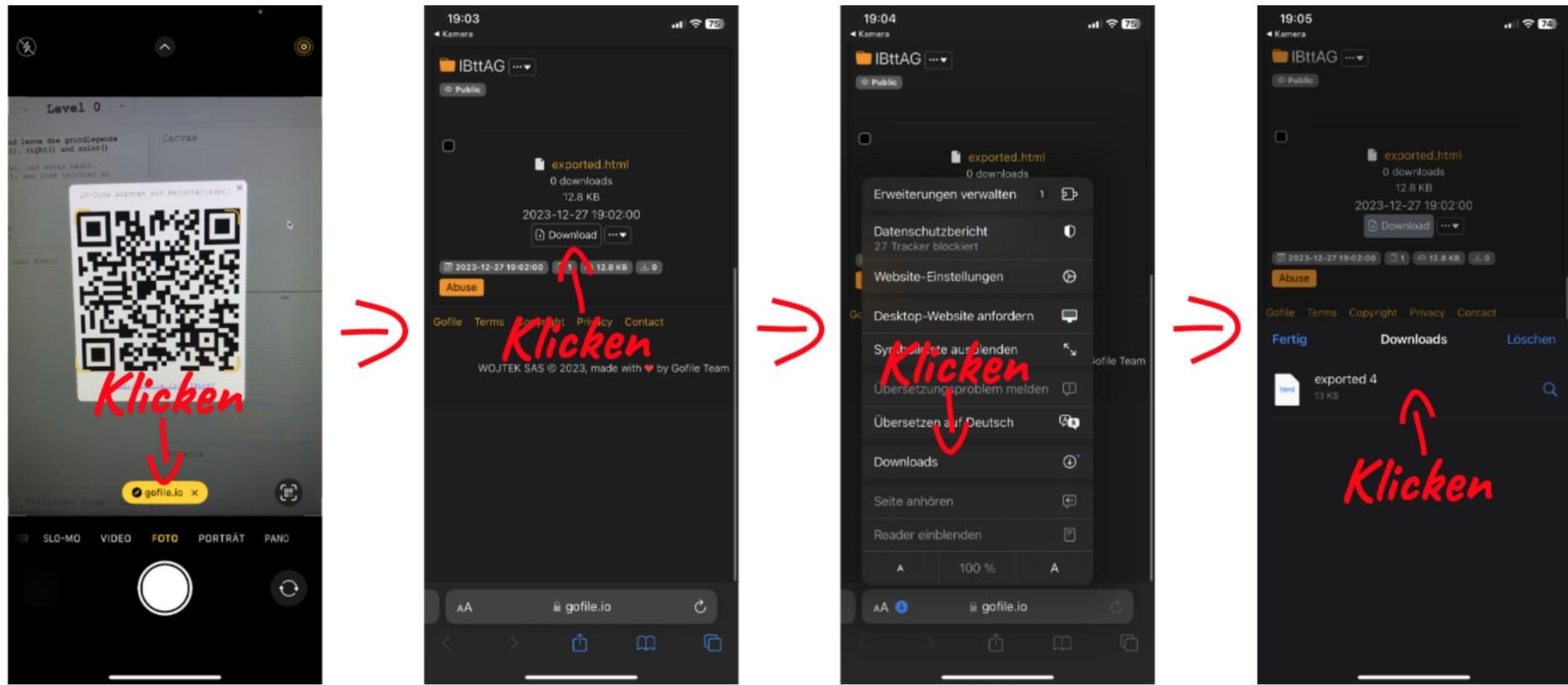
1. **Tutorial Bereich:** Hier steht eine Aufgabenstellung des Levels und Code-Beispiele, die dabei helfen, das Level erfolgreich abzuschließen.
2. **Code-Editor Bereich:** Hier wird Code geschrieben. Hier steht schon zu Beginn für jedes Level ein initialer Code. Jedes Level hat dieselbe erste Zeile, die die Turtle initialisiert. **Die erste Zeile soll nicht entfernt oder verändert werden.** Ansonsten kann es zu Problemen bei der Ausführung kommen.
3. **Canvas Bereich:** Hier zeichnet die Turtle die Zeichnung, sobald der **Run-Button (5)** gedrückt wird.
4. **Konsolen Bereich:** Hier werden Konsolenausgaben angezeigt.

Buttons/Knopfe (rot)

5. **Run:** Führt Code aus dem **Code-Editor (2)** aus.
6. **Koordinatensystem:** Blendet Koordinatensystem im **Canvas (3)** ein und aus. Beim Zeichnen hilft es, mit Längen und Koordinaten zu zeichnen.
7. **Initialer Code:** Zeigt Start Code und hilft dabei, wenn man sich verrannt hat oder die erste Zeile entfernt wurde.
8. **Lösung:** Zeigt die Lösung in zwei Schritten. Erster Schritt nur Kommentare der Lösung angezeigt. Zweiter Schritt zeigt die Codelösung mit Kommentaren. Für diese gesamte Lösung braucht man ein Passwort. Dieses erhält man vom Dozenten.
9. -: Vorheriges Level
10. +: Nächstes Level
11. **Dark Mode / Light Mode:** Beim Starten der Anwendung werden Systemeinstellungen verwendet, um zu entscheiden, ob Anwendung im Light- oder Dark Mode ist. Das kann über diesen Button nachträglich gewechselt werden.
12. **Exportieren:** Zeigt Menü mit mehreren Buttons zum Exportieren der Ergebnisse. Die exportierte HTML-Datei kann in jedem Browsern auf PC und Smartphone geöffnet werden.
13. **Download:** Lädt die exportierte HTML-Datei herunter und speichert diese im Download Ordner. Aus diesem kann man die Datei auf einen USB Stick kopieren.
14. **Drucken:** Druckt die exportierte HTML-Datei. Nicht wundern: Der Druckvorgang kann erst nach einigen Sekunden gestartet werden.
15. **QR-Code:** Zeigt QR-Code an. Dieser kann mit Smartphone-Kamera gescannt werden. QR-Code leitet auf dem Smartphone auf der Downloadseite weiter. Hier kann die exportierte HTML-Datei heruntergeladen und später heruntergeladen werden.

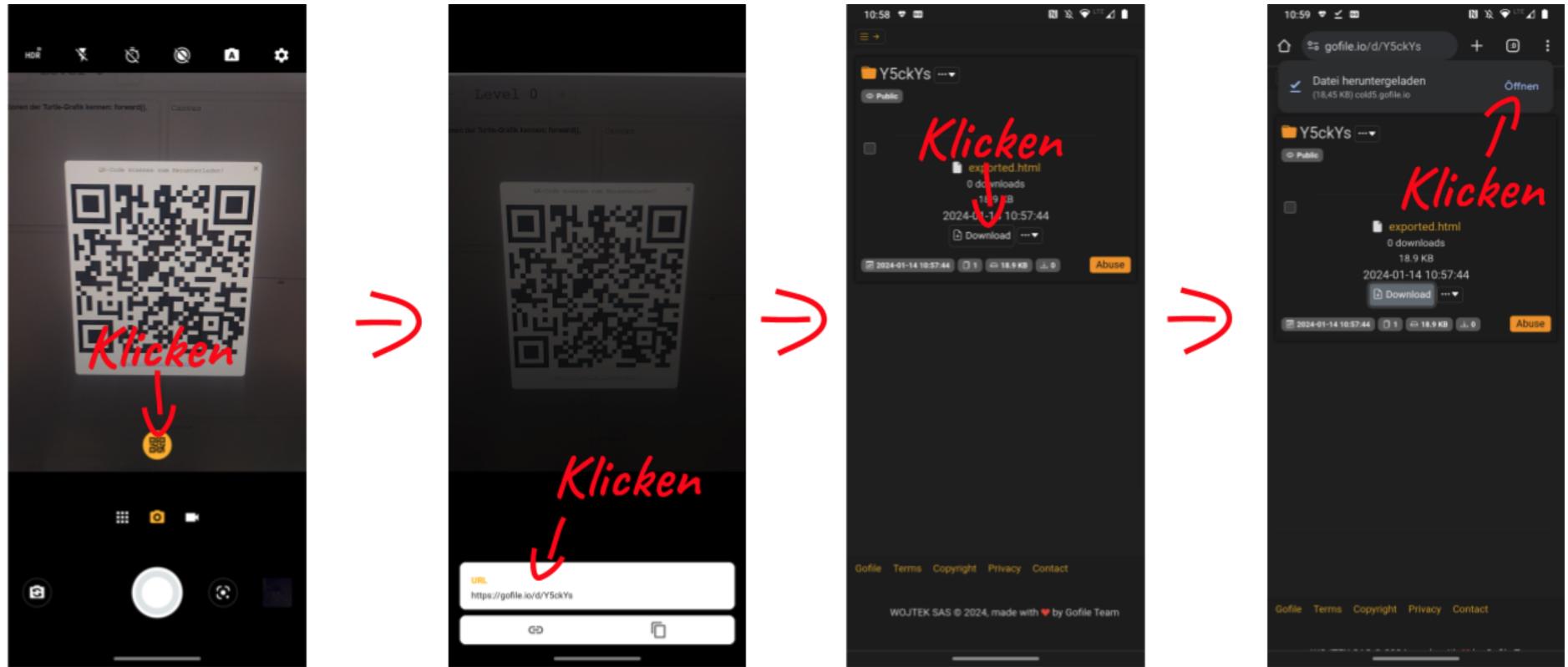
iPhone/iOS

Hier ist dargestellt, welche Schritte man befolgen muss, um die exportierte HTML-Seite auf dem iPhone herunterzuladen und zu öffnen.



Android

Hier ist dargestellt, welche Schritte man befolgen muss, um die exportierte HTML-Seite auf einem Android-Handy herunterzuladen und zu öffnen



5 Programmierkonzepte

Hier werden Programmierkonzepte erklärt. Diese dienen als Hilfe zum Bearbeiten der Level.

Variablen (Level 1, 2, 3, 4, 5, 6, 7)

Variablen sind wie Container für Informationen in der Programmierung. Du kannst dir eine Variable wie eine Schublade vorstellen, in der du Werte speichern kannst. Diese Werte können Zahlen, Strings (Zeichenfolgen, die Text enthalten) oder andere Daten sein.

Wertzuweisung

Du kannst Variablen einen Namen geben. Über den Namen einer Variable kannst du auf den Wert der Variable zugreifen.



Welchen Wert hat die Variable **b**?

- Richtig, der Wert ist 1, weil der Wert von Variable **a** den Wert auf **b** zugewiesen hat.

Rechnen mit Variablen (Arithmetische Operationen)

Mit Variablen, die Zahlen enthalten, kann gerechnet werden.

```

</> a = 2
    b = 2

    summe = a + b      # Ergebnis einer Addition
    differenz = a - b  # Ergebnis einer Subtraktion
    produkt = a * b    # Ergebnis einer Multiplikation
    quotient = a / b    # Ergebnis einer Division
    rest_der_division = a % b # Liefert Rest einer Division
  
```

← Kennst du noch aus der Grundschule

Ohne zu wissen, dass du es wieder brauchst, hast du den `%` Modulo Operator schon in der Grundschule verwendet.



Wenn du noch weißt wie in Grundschule geteilt mit Rest gerechnet wurde,

- nenne den Wert von `rest_der_division`?

Richtig, der Wert von `rest_der_division` ist 0.



Hinter einem `#` werden **Kommentare** geschrieben. Kommentare helfen dabei, eigenen und fremden Code zu verstehen.



Wie du siehst, sind alle Variablen vorne kleingeschrieben. Wenn man mehrere Wörter in einem Variablenamen nutzen möchte, dann trennt man diese durch einen Unterstrich. Diesen Code-Stil nennt man **snake_case**. Es gibt auch andere Code-Stile z.B. **camelCase**. Das Programm funktioniert natürlich auch ohne Code-Stil. Aber Informatiker brauchen Regeln, ansonsten gibt es beim Zusammenarbeiten nur Streit.

Schleifen (Level 2, 3, 4, 5, 6, 7)

Schleifen sind einer der Grundbausteine der Programmierung, welche die Programmierung vereinfachen.

Einsatz und Vorteile von Schleifen

Aber wieso sind Schleifen überhaupt so wichtig? Angenommen, du willst 1000-mal das Gleiche schreiben. Das könnte man auf zwei Wege umsetzen.

1. Variante:

```
</> print(0) ← Konsolenausgabe
    print(1)
    print(2)
    print(3)
    print(4)
    print(5)
    print(6)
    print(7)
    print(8)
    print(9)
    # ...
    print(999) # ... 1000 Zeilen später
```

2. Variante:

```
</> for i in range(1000): # [0, 1, 2, ..., 999] ← Array
    print(i) ← startet mit 0
    ↪ eingerückt
```

(Annotations: 'Doppelpunkt' points to the colon after 'range'. 'Array' points to the brackets after 'range'. 'Laufvariable' points to 'i'. 'eingerückt' points to the indented 'print(i)' line.)



Welche der zwei Varianten ist sinnvoller?

- Ich wette, du hast an Weg 2 gedacht. Informatiker versuchen Code so effizient wie möglich zu schreiben. Das Programmieren geht so nicht nur viel schneller, sondern ist auch viel übersichtlicher und es ist einfacher, etwas am Code zu ändern.

Stell dir vor, du willst statt 0 bis 999 die Zahlen 0 bis 999999 ausgeben. Bei der 1. Variante braucht man eine Million Zeilen. Bei der 2. Variante muss man nur die Zahl n innerhalb von **range (n)** ändern.

Eine For-Schleife hat einen Schleifenkopf, der mit dem Schlüsselwort **for** beginnt. Die Laufvariable **i** (kann auch anders heißen) läuft in diesem Beispiel durch die Werte 0 bis 999 durch.

Für jeden Wert, den **i** dabei annimmt, wird der Schleifenkörper ausgeführt. In dem Schleifenkörper kann die Laufvariable verwendet werden. Außerdem davon geht das nicht.



Einrückung des Schleifenkörpers

In Python wird der Schleifenkörper mit der Taste TABULATOR eingerückt, um zu zeigen das der eingerückte Code von der **for**-Schleife mehrmals ausgeführt werden soll. Legt man eine Variable innerhalb des Schleifenkörpers an, dann kann man diese wie die Laufvariable nur innerhalb des Schleifenkörpers nutzen.



Beim Programmieren fängt man mit 0 statt mit 1 beim Zählen an. Dadurch wird bei **range(1000)** auch nur bis 999 (1000 - 1) gezählt.



Mit **print()** kann man Werte (auch aus Variablen) auf der Konsole ausgeben.
Mit **print('Text' str(number))** kann man Texte oder Zahlen kombinieren.

Arrays (Level 4, 5)

Ein Array ist eine Datenstruktur in Python und kann mehrere Werte speichern. Das können nicht nur Zahlen, sondern auch alle anderen Werte, wie Strings (Zeichenfolgen) sein. Sogar Arrays können in Arrays gespeichert werden.



Bei der 2. Variante steht hinter **range(1000)** als Kommentar **[0, 1, 2, ..., 999]** um zu zeigen, dass aus **range(1000)** ein Array mit 0 bis 999 generiert wird.

Folgende zwei Codebeispiele zeigen unterschiedlichen Zugriffsmöglichkeiten auf Werte in Arrays. Beide Codes sind beide sinnvoll programmiert und ergeben die gleiche Ausgabe.

Bei einem **direkten** Zugriff wird direkt auf den Wert der Laufvariable **string** zugriffen. Die Laufvariable ändert sich in jedem Durchlauf des Schleifenkörpers.

```
</> string_array = ['eins', 'zwei', 'drei'] ← Array mit Strings
    for string in string_array:
        print(string) ← direkter Zugriff auf Laufvariable
```

Bei einem **indirekter Zugriff mittels Index**, wird mit dem Index `i` auf den Wert auf der Stelle `i` im Array `string_array` zugegriffen.

```
</> string_array = ['eins', 'zwei', 'drei']
    for i in range(3): # [0, 1, 2]
        print(string_array[i]) ←indirekter Zugriff mittels Index
```

Verschachtelte Schleifen (Level 6)

Schleifen lassen sich verschachteln. Hier ist ein Beispiel:

```
</> for i in range(3): # [0, 1, 2]
    for j in range(4): # [0, 1, 2, 3]
        print(i * j) ← zweimal eingerückt
# Ausgabe
# 0, 0, 0, 0, 0, 1, 2, 3, 0, 2, 4, 6
```

If-Bedingung (Level 3, 4, 5)

In der Programmierung gibt es oft Situationen, in denen dein Code Entscheidungen treffen muss. Du kannst es dir so vorstellen, als ob du in deinem Programm verschiedene Pfade vor dir hast und je nach bestimmten Bedingungen einen bestimmten Weg einschlagen musst.

Die **If-Bedingung** sind ein solches Konzept, um diese Situationen zu bewältigen und sicherzustellen, dass dein Code auf verschiedene Szenarien richtig reagieren kann.

```
</> if age < 13: Bedingung
    print('Du bist unter 13 Jahre alt') eingerückt
```

Damit die **If-Bedingung** funktioniert, muss die Frage oder Bedingung, die sie stellt, mit Ja oder **True** (Wahr) beantwortet werden. Wenn die Antwort Nein oder **False** (Falsch) ist, macht das Programm etwas anderes oder geht einfach weiter.

```
</> if age < 13: 1. If-Bedingung
    print('Du bist unter 13 Jahre alt')

if True: 2. If-Bedingung
    print('Dies wird immer ausgeführt')

if False: 3. If-Bedingung
    print('Dies wird nie ausgeführt')
```

Hier überprüft die erste **If-Bedingung**, ob du jünger als 13 Jahre bist. Wenn das stimmt, sagt der Code Du bist unter 13 Jahre alt.

Die zweite **If-Bedingung** ist ein bisschen wie ein Scherz. Er sagt immer Dies wird immer ausgeführt, egal wie alt du bist. Das liegt daran, dass **True** so viel wie immer wahr bedeutet. Deswegen wird dieser Teil immer passieren.

Der letzte Teil ist genau das Gegenteil. Er hat **False** drin, was so viel wie Immer falsch bedeutet. Deshalb wird der Satz Dies wird nie ausgeführt auch wirklich nie gesagt.

Erweiterte If-Bedingungen

```
</> if age < 13:
    print('Du bist unter 13 Jahre alt')
elif age >= 13 and age < 18:
    print('Du bist ein Teenager, aber nicht erwachsen')
elif 18 <= age < 20:
    print('Du bist ein Teenager und erwachsen')
else:
    print('Du bist erwachsen')
```



Wenn du 15 Jahre alt bist, was für eine Ausgabe wird ausgegeben?

- Richtig, die Ausgabe ist '**Du bist ein Teenager, aber nicht erwachsen**', da 15 größer gleich 13, aber kleiner als 18 ist.

Um eine **If-Bedingung** zu erweitern, kannst du mit **elif** (eine Abkürzung für "else if", was "sonst wenn" bedeutet) eine weitere **If-Bedingung** hinzufügen, die geprüft wird, wenn die vorherige Wahl nicht wahr war.

Mit **else** (also "sonst") bestimmst du, was passieren soll, wenn keine der vorherigen Wahlen zutrifft. So kannst du sicherstellen, dass immer etwas ausgeführt wird, egal ob Bedingungen erfüllt werden können oder nicht.



Bei den If-Bedingungen können auch **and** und **or** verwendet werden, um mehr Szenarien zu berücksichtigen. Bei **and** müssen alle Bedingungen wahr sein, während bei **or** nur einer der Bedingungen wahr sein muss.

Funktionen (Level 5)

Funktionen in der Programmierung sind wie Wege, die du in deinem Code festlegst. Stell dir vor, du hast einen Rucksack voller Werkzeuge (**Funktionen**), die du immer wieder verwenden kannst, ohne sie neu zu packen. Diese Werkzeuge helfen dir, bestimmte Aufgaben wie die Addition von zwei Zahlen schnell und effizient zu erledigen.

Genau wie bei Weggabelungen in einem Programm, wo du mit **If-Bedingungen** entscheiden musst, welchen Pfad du einschlagen willst, rufst du eine **Funktion** auf, wenn du sie benötigst. Diese Entscheidungspunkte stellen sicher, dass dein Code flexibel auf verschiedene Situationen reagieren kann.

Damit man eine **Funktion** erstellt, muss diese vorher definiert werden, dies tut man mit **def**, anschließend den **Namen der Funktion** und die erwünschten, wenn gebrauchten, **Übergabeparameter**.

```
</>      ↗ Funktion
        def funktionsname(a,b):
            ↗ Übergabeparameter
            print(a, 'und', b) ↗ eingerückt
            return a ↗ Rückgabewert
```

Wenn keine **Übergabeparameter** benötigt werden, können diese entfernt werden. Die Klammern müssen aber, vollständigkeitshalber, vorhanden sein.

```
</> def funktion_2():
        ↗ Rückgabe
        return 'Rückgabe'

print(funktion_2())
```



Funktionsnamen, Duplikate und Schlüsselwörter

Funktionen brauchen einen eigenen Namen, der nicht von anderen Funktionen, sowie von Schlüsselwörtern wie “**for**, **if**, **elif**,...”, verwendet wird.

Um eine Funktion zu verwenden, muss diese vorher aufgerufen werden.

```
</> def addition(x,y):
    return (x+y)

print(addition(1.2,2))
print(addition(15,-2))
print(addition(0,56))
print(addition((3+4),56))
```

Man braucht nicht mit `return` etwas zurückzugeben. Manche Funktionen führen nur etwas aus und müssen nichts zurückzugeben.

Falls man aber keine Werte für einen der Übergabeparameter hat, kann man in der Funktion selbst einen Standardwert vordefinieren. Dies macht es, dass selbst wenn kein Parameter übergeben wird, die Funktion aufrufbar ist.

Aber was ist, wenn man nur auf `y` zugreifen möchte. Dann kann man in der Übergabe dies definieren, indem man den direkten Variablennamen der Funktion "überschreibt".

```
</> def subtraktion(x=3,y=5):
    return (x-y)

print(subtraktion()) # Ausgabe: -2
print(subtraktion(x=17)) # Ausgabe: 12
print(subtraktion(y=2)) # Ausgabe: 1
print(subtraktion(y=2, x=17)) # Ausgabe: 15
print(subtraktion(x=15, y=-17)) # Ausgabe: 32
```



Was ist die Ausgabe der Subtraktions-Funktion, wenn keine Übergabeparameter übergeben werden?

Richtig, die Ausgabe ist **-2**, da `x=3` und `y=5` ist und somit $x - y = 3 - 5 = -2$ ist.

In den vorherigen Kapiteln haben wir schon die `print()` Funktion benutzt, die Texte in der Konsole anzeigt. Normalerweise macht `print()` am Ende einen Zeilenumbruch "`\n`", aber mit dem `end` Parameter kannst du das ändern und entscheiden, was am Ende stehen soll.

```
</> print('Das ist der Übergabeparameter', end='Ende')
```