Anhang

Anhang

Skript Informatik · Klasse 10 · MINT

Im Anhang findest du drei Hilfen zum Nachschlagen: wie du Python zu Hause einrichtest, einen Spickzettel mit allen Python-Befehlen aus Kapitel 1 und eine Übersicht über die häufigsten Fehlermeldungen mit ihren Ursachen.

A.1 Python zu Hause einrichten

Um die Beispiele und Aufgaben dieses Skripts zu Hause auszuprobieren, brauchst du eine **Programmierumgebung**: ein Programm, in dem du Python-Code schreiben und starten kannst. Du hast drei Möglichkeiten. Für Anfängerinnen und Anfänger empfehlen wir Thonny.

A.1.1 Möglichkeit 1: Thonny (empfohlen)

Thonny ist eine kostenlose Programmierumgebung, die extra für Anfänger entwickelt wurde. Sie bringt Python gleich mit: Du musst nur ein Programm installieren.

1. Öffne die Seite **thonny.org** und lade die Version für dein Betriebssystem herunter (Windows, macOS oder Linux).
2. Installiere Thonny wie jedes andere Programm.
3. Starte Thonny. Oben siehst du den **Editor**: Hier schreibst du dein Programm. Unten siehst du die **Kommandozeile** (Shell): Hier erscheinen die Ausgaben, und hier tippst du Eingaben für `input()` ein.
4. Tippe `print("Hallo Welt!")` in den Editor und klicke auf den grünen Pfeil (**Run**). Thonny fragt beim ersten Mal, unter welchem Namen das Programm gespeichert werden soll. Wähle einen Namen, der auf `.py` endet, zum Beispiel `hallo.py`.

Tipp

Thonny kann dir zeigen, was in deinem Programm passiert. Öffne im Menü **Ansicht** (englisch *View*) das Fenster **Variablen** (*Variables*). Dort siehst du jederzeit, welche Variablen es gibt und welchen Wert sie haben. Mit dem Käfer-Symbol (**Debug**) und den Pfeilen daneben kannst du ein Programm Schritt für Schritt ablaufen lassen. Das ist ein Ablaufprotokoll, das der Computer für dich schreibt.

A.1.2 Möglichkeit 2: Jupyter im Browser, ohne Installation

Wenn du nichts installieren kannst oder willst, kannst du Python direkt im Browser ausprobieren. Öffne **jupyter.org/try** und starte dort die Vorführung **Jupyter Notebook**. Sie läuft vollständig in deinem Browser.

- Ein Notebook besteht aus **Zellen**. In eine Code-Zelle schreibst du dein Programm und startest es mit **Umschalt + Enter**.
- Deine Dateien werden nur im Browser gespeichert. Lade wichtige Notebooks deshalb über das Menü herunter.
- Nicht jede Funktion klappt im Browser zuverlässig. Wenn zum Beispiel `input()` nicht funktioniert, nutze Thonny.

A.1.3 Möglichkeit 3: Die Umgebung aus dem Unterricht

Im Unterricht arbeitest du mit Notebooks. Wenn deine Schule eine Notebook-Umgebung im Internet bereitstellt, kannst du auch von zu Hause aus damit arbeiten. Frag deine Lehrkraft nach dem Zugang.

Stolperstein

Ein Unterschied zwischen Notebook und Thonny: Im Notebook wird der Wert der **letzten Zeile** einer Zelle automatisch angezeigt, auch ohne `print`. In Thonny (und in jedem normalen Python-Programm) erscheint nur, was du mit `print(...)` ausgibst. Die Programme in diesem Skript benutzen deshalb immer `print`.

Gymnasium Ottweiler

A.2 Python-Spickzettel

Hier stehen alle Python-Befehle aus Kapitel 1 auf einen Blick. Die Spalte „Ergebnis“ zeigt, was herauskommt oder ausgegeben wird.

A.2.1 Ausgabe, Eingabe, Variablen

Was	Beispiel	Ergebnis
Text ausgeben	`print("Hallo")`	`Hallo`
mehrere Werte ausgeben	`print("Alter:", 15)`	`Alter: 15`
Variable anlegen (Zuweisung)	`alter = 15`	`alter` hat den Wert `15`
Variable verändern	`alter = alter + 1`	`alter` hat den Wert `16`
Text einlesen	`name = input("Name? ")`	`name` ist immer ein `str`
ganze Zahl einlesen	`zahl = int(input("Zahl? "))`	`zahl` ist ein `int`
Kommazahl einlesen	`x = float(input("Wert? "))`	`x` ist ein `float`
Datentyp anzeigen	`print(type(1.5))`	`<class 'float'>`
in Text umwandeln	`str(15)`	`"15"`
in Zahl umwandeln	`int("15")` / `float("2.5")`	`15` / `2.5`
runden	`round(3.14159, 2)`	`3.14`

A.2.2 Datentypen

Datentyp	Bedeutung	Beispiele
`int`	ganze Zahl	`15`, `-3`, `0`
`float`	Kommazahl (mit Punkt!)	`1.68`, `3.0`
`str`	Zeichenkette (Text)	`"Hallo"`, `"15"`
`bool`	Wahrheitswert	`True`, `False`

A.2.3 Operatoren

Art	Operatoren	Beispiel	Ergebnis
Grundrechenarten	`+ - * /`	`7 / 2`	`3.5`
ganzzahlige Division	`//`	`7 // 2`	`3`
Rest der Division	`%`	`7 % 2`	`1`
Potenz	`**`	`2 ** 3`	`8`
Vergleiche	`== != < <= > >=`	`3 != 4`	`True`
logisches UND	`and`	`True and False`	`False`
logisches ODER	`or`	`True or False`	`True`
logisches NICHT	`not`	`not True`	`False`
De Morgan	`not (A and B)`	wie `not A or not B`	
Kurzschreibweise	`+= -= *=`	`x += 2`	wie `x = x + 2`

A.2.4 Verzweigungen

```
if bedingung:
    anweisungen        # werden nur ausgeführt, wenn die Bedingung wahr ist
elif andere_bedingung:
    anweisungen        # sonst: wenn diese Bedingung wahr ist
else:
    anweisungen        # sonst: in allen übrigen Fällen
```

Doppelpunkt am Ende der Zeile, eingerückte Anweisungen (4 Leerzeichen). `elif` und `else` sind freiwillig.

Mehrfachauswahl (`CASE OF`)

```
match wert:
    case 1:
        anweisungen    # wenn wert == 1
    case 2:
        anweisungen    # wenn wert == 2
    case _:
        anweisungen    # Auffangfall: alle anderen Werte (wie else)
```

Nur für den Vergleich **eines** Werts mit **festen** Werten. Für Bereiche (`alter < 18`) nimmt man `if`. Ab Python 3.10.

A.2.5 Schleifen

Was	Beispiel	Ergebnis
Zählschleife	`for i in range(5):`	`i` = 0, 1, 2, 3, 4
mit Startwert	`for i in range(1, 5):`	`i` = 1, 2, 3, 4
mit Schrittweite	`for i in range(0, 10, 3):`	`i` = 0, 3, 6, 9
über Zeichen	`for z in "Hi":`	`z` = `"H"`, `"i"`
über Listenelemente	`for x in [4, 7]:`	`x` = 4, 7
bedingte Schleife	`while zahl > 0:`	wiederholt, solange die Bedingung wahr ist
Schleife verlassen	`break`	beendet die Schleife sofort

for oder while? Wenn du vorher weißt, wie oft wiederholt wird (oder über alle Elemente gehst): `for`. Wenn es von einer Bedingung abhängt, die sich erst während des Ablaufs entscheidet: `while`.

A.2.6 Zufall und Mathematik

Was	Beispiel	Ergebnis
Zufallszahl (nach `import random`)	`random.randint(1, 6)`	1, 2, 3, 4, 5 oder 6
Kreiszahl π (nach `import math`)	`math.pi`	`3.141592653589793`

`import random` bzw. `import math` steht einmal am Anfang des Programms.

A.2.7 Funktionen

```
1 def quadrat(x):           # Funktion mit dem Parameter x definieren
2     return x * x          # Rückgabewert
3
4 ergebnis = quadrat(4)     # Funktion aufrufen: ergebnis bekommt den Wert 16
5 print(quadrat(5))         # gibt 25 aus
```

A.2.8 Zeichenketten

Was	Beispiel (`s = "Hallo"`)	Ergebnis
Länge	`len(s)`	`5`
Zeichen an Index	`s[0]` / `s[4]`	`"H"` / `"o"`
letztes Zeichen	`s[-1]`	`"o"`
verketten	`s + " Welt"`	`"Hallo Welt"`
vervielfachen	`"ab" * 3`	`"ababab"`
enthalten?	`"ll" in s`	`True`
Groß-/Kleinbuchstaben	`s.upper()` / `s.lower()`	`"HALLO"` / `"hallo"`
nur Buchstaben / Ziffern?	`s.isalpha()` / `s.isdigit()`	`True` / `False`
f-String	`f"{s}, Tom!"`	`"Hallo, Tom!"`
Zeichencode	`ord("A")` / `chr(66)`	`65` / `"B"`

A.2.9 Listen

Was	Beispiel (`z = [4, 7, 1]`)	Ergebnis
Element lesen	`z[0]` / `z[-1]`	`4` / `1`
Element ändern	`z[1] = 9`	`z` ist `[4, 9, 1]`
Länge	`len(z)`	`3`
anhängen	`z.append(5)`	`z` ist `[4, 7, 1, 5]`
einfügen	`z.insert(0, 8)`	`z` ist `[8, 4, 7, 1]`
entfernen (Wert)	`z.remove(7)`	`z` ist `[4, 1]`
entfernen (letztes)	`z.pop()`	gibt `1` zurück, `z` ist `[4, 7]`
entfernen (Index)	`z.pop(0)`	gibt `4` zurück, `z` ist `[7, 1]`
enthalten? / Index suchen	`7 in z` / `z.index(7)`	`True` / `1`
Summe, Minimum, Maximum	`sum(z)`, `min(z)`, `max(z)`	12, 1, 7
sortiert	`sorted(z)`	`[1, 4, 7]`
leere Liste	`z = []`	`z` hat 0 Elemente

Jede Zeile der Tabelle geht von der ursprünglichen Liste `z = [4, 7, 1]` aus.

A.3 Fehlermeldungen verstehen

Eine Fehlermeldung liest du **von unten nach oben**: In der letzten Zeile stehen die Art des Fehlers und eine kurze Beschreibung. Darüber steht, in welcher Zeile (*line*) der Fehler aufgetreten ist. Die Tabellen zeigen die häufigsten Meldungen, ihre Bedeutung und was du tun kannst. Die Meldungen stammen aus Python 3.13; andere Versionen formulieren manche etwas anders, die Fehlerart am Anfang bleibt aber gleich.

A.3.1 Fehler beim Schreiben (das Programm startet gar nicht)

Meldung	Bedeutung	Was du tun kannst
`SyntaxError: unterminated string literal …`	Ein Text ist nicht abgeschlossen.	Fehlendes Anführungszeichen ergänzen.
`SyntaxError: '(' was never closed`	Eine Klammer wurde nicht geschlossen.	Klammern zählen: Zu jeder `(` gehört eine `)`.
`SyntaxError: expected ':'`	Nach `if`, `elif`, `else`, `for`, `while` oder `def` fehlt der Doppelpunkt.	Doppelpunkt am Zeilenende ergänzen.
`SyntaxError: invalid syntax. Maybe you meant '==' …`	In einer Bedingung steht `=` statt `==`.	Zum Vergleichen `==` schreiben.
`IndentationError: expected an indented block …`	Nach einer Zeile mit Doppelpunkt fehlt die Einrückung.	Die folgenden Zeilen um 4 Leerzeichen einrücken.
`IndentationError: unexpected indent`	Eine Zeile ist eingerückt, obwohl sie es nicht sein darf.	Leerzeichen am Zeilenanfang entfernen.
`IndentationError: unindent does not match any outer indentation level`	Die Einrückung passt zu keiner Ebene.	Gleich viele Leerzeichen verwenden wie die Zeilen darüber.

Gymnasium Ottweiler

A.3.2 Fehler beim Ausführen (das Programm bricht ab)

Meldung	Bedeutung	Was du tun kannst
`NameError: name 'x' is not defined`	Der Name ist unbekannt.	Tippfehler? Groß-/Kleinschreibung? Variable vorher angelegt? Anführungszeichen vergessen?
`TypeError: can only concatenate str (not "int") to str`	Text und Zahl wurden mit `+` verbunden.	Zahl mit `str(...)` umwandeln oder Kommas in `print` verwenden.
`TypeError: unsupported operand type(s) for +: 'int' and 'str'`	Mit einem Text wurde gerechnet.	Eingabe mit `int(...)` oder `float(...)` umwandeln.
`TypeError: f() missing 1 required positional argument: 'a'`	Beim Aufruf einer Funktion fehlt ein Wert für einen Parameter.	Beim Aufruf so viele Werte angeben, wie die Funktion Parameter hat.
`ValueError: invalid literal for int() with base 10: '3.5'`	Der Text lässt sich nicht in eine ganze Zahl umwandeln.	Eingabe prüfen; für Kommazahlen `float(...)` verwenden, Punkt statt Komma.
`ValueError: could not convert string to float: '2,5'`	Der Text ist keine Kommazahl (oft: Komma statt Punkt).	Punkt statt Komma eingeben.
`ZeroDivisionError: division by zero`	Es wurde durch 0 geteilt.	Vorher mit `if` prüfen, ob der Teiler 0 ist.
`IndexError: list index out of range`	Es gibt keine Listenposition mit diesem Index.	Index beginnt bei 0; der größte Index ist `len(liste) - 1`.
`IndexError: string index out of range`	Dasselbe bei einer Zeichenkette.	wie oben
`ValueError: list.remove(x): x not in list`	Der Wert, der entfernt werden soll, steht nicht in der Liste.	Vorher mit `in` prüfen, ob der Wert enthalten ist.
`AttributeError: 'int' object has no attribute 'append'`	Die Methode passt nicht zum Datentyp.	Prüfen, ob die Variable wirklich eine Liste (bzw. Zeichenkette) ist.

A.3.3 Fehler ohne Fehlermeldung

Manche Fehler meldet Python nicht, weil das Programm „richtig geschrieben“ ist – es tut nur nicht das, was du willst. Sie sind am schwersten zu finden.

Was passiert	mögliche Ursache
Statt `7` kommt `34` heraus.	Eingaben nicht umgewandelt: `"3" + "4"` hängt Texte aneinander.
Das Programm hört nicht mehr auf.	Endlosschleife: Die Bedingung der `while`-Schleife wird nie falsch. Abbrechen mit dem Stopp-Knopf bzw. **Strg + C**.
Eine Schleife läuft einmal zu wenig.	`range(1, 10)` endet bei 9, nicht bei 10.
Die Summe ist falsch, obwohl die Schleife stimmt.	Die Summenvariable wird in der Schleife (statt davor) auf 0 gesetzt.
Eine Funktion gibt `None` zurück.	In der Funktion steht `print(...)` statt `return ...`, oder `return` fehlt.
Eine Zeile wird zu oft oder zu selten ausgeführt.	Falsche Einrückung: Die Zeile gehört (nicht) zum Schleifen- oder `if`-Block.

Tipp

Wenn du einen Fehler nicht findest: Lass dir mit zusätzlichen `print`-Anweisungen anzeigen, welche Werte die Variablen gerade haben. Oder schreibe ein Ablaufprotokoll. Lösche die zusätzlichen Ausgaben, wenn der Fehler gefunden ist.