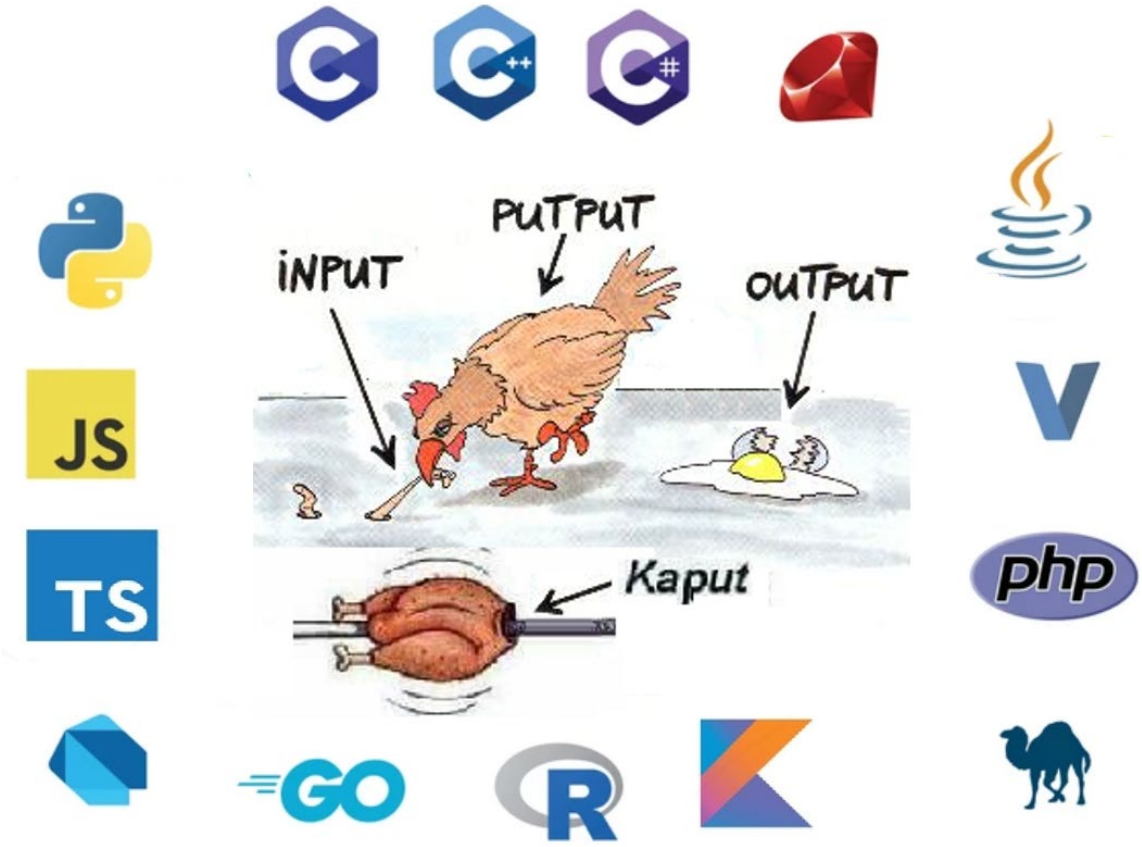
Lösungsalgorithmen & Programmieren

- Block 01 (Grundlagen & algorithm. Denken)-

Abend 01



🎯 Leistungs-/Lernziele dieses Blocks:

Die Studierenden …
kennen den Aufbau und Ablauf des Faches LAP und wissen wie die Bewertung (Notengebung) erfolgen wird
kennen die gängigsten Programmiersprachen und können zwischen interpretierenden und kompilierenden Sprachen unterscheiden.
können Python einrichten und ein erstes Skript mit der integrierten IDE ausführen
können VS Code installieren und für die Verwendung von Python einrichten
können Variablen und Datentypen (int, float, str, bool) korrekt einsetzen (Casten)
können einfache Python-Applikationen mit EVA (Eingabe Verarbeitung und Ausgabe) realisieren
können Verzweigungen mit if, elif, else umsetzen
einfache Alltagsprobleme (ohne Iterationen) algorithmisch beschreiben und in Python umsetzen

Inhaltsverzeichnis:

1 Einstieg

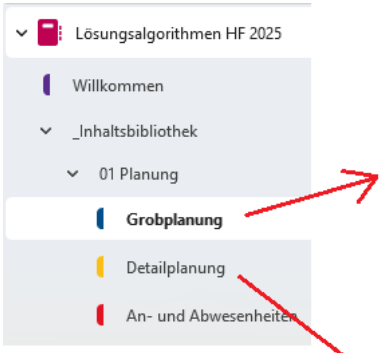
1.1 Begrüssung -Vorstellung Lehrperson

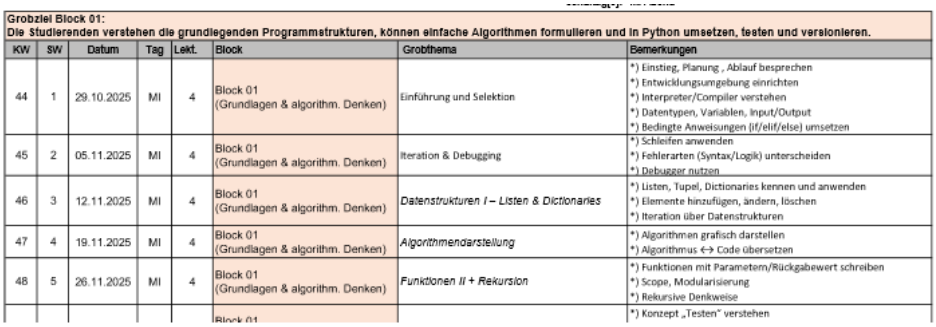
Ich stelle mich mit einer PowerPoint-Präsentation kurz vor.

Roland Bucher
Martinshöhe 1B
6204 Sempach
Mobile +41 79 520 26 72
roland.bucher@edu.teko.ch

1.2 Unterrichtsplanung

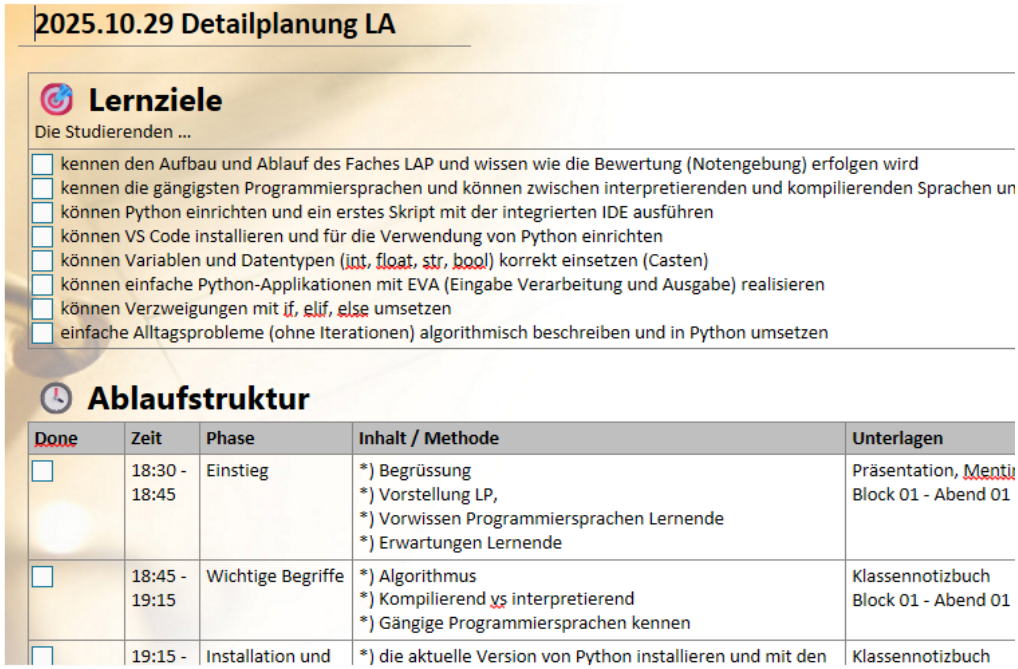
Die Planungsdokumente sind im Klassennotizbuch ersichtlich. Zur Unterrichtsplanung existiert eine Grobplanung (Überblick über die 18 Abende) und eine Detailplanung (Überblick über die Tätigkeiten der einzelnen Abende). Solltest du also einmal fehlen, kannst du dir gut in der Detailplanung einen Überblick verschaffen.





Grobziel Block 01:
Die Studierenden verstehen die grundlegenden Programmstrukturen, können einfache Algorithmen formulieren und in Python umsetzen, testen und versionieren.

KW	SW	Datum	Tag	Lekt.	Block	Grobthema	Bemerkungen
44	1	29.10.2025	MI	4	Block 01 (Grundlagen & algorithm. Denken)	Einführung und Selektion	*) Einstieg, Planung , Ablauf besprechen *) Entwicklungsumgebung einrichten *) Interpreter/Compiler verstehen *) Datentypen, Variablen, Input/Output *) Bedingte Anweisungen (if/elif/else) umsetzen
45	2	05.11.2025	MI	4	Block 01 (Grundlagen & algorithm. Denken)	Iteration & Debugging	*) Schleifen anwenden *) Fehlerarten (Syntax/Logik) unterscheiden *) Debugger nutzen
46	3	12.11.2025	MI	4	Block 01 (Grundlagen & algorithm. Denken)	*Datenstrukturen I – Listen & Dictionaries*	*) Listen, Tupel, Dictionaries kennen und anwenden *) Elemente hinzufügen, ändern, löschen *) Iteration über Datenstrukturen
47	4	19.11.2025	MI	4	Block 01 (Grundlagen & algorithm. Denken)	*Algorithmendarstellung*	*) Algorithmen grafisch darstellen *) Algorithmus ↔ Code übersetzen
48	5	26.11.2025	MI	4	Block 01 (Grundlagen & algorithm. Denken)	*Funktionen II + Rekursion*	*) Funktionen mit Parametern/Rückgabewert schreiben *) Scope, Modularisierung *) Rekursive Denkweise
					Block 01		*) Konzept „Testen" verstehen



2025.10.29 Detailplanung LA

Lernziele

Die Studierenden ...

- [ ] kennen den Aufbau und Ablauf des Faches LAP und wissen wie die Bewertung (Notengebung) erfolgen wird
- [ ] kennen die gängigsten Programmiersprachen und können zwischen interpretierenden und kompilierenden Sprachen un
- [ ] können Python einrichten und ein erstes Skript mit der integrierten IDE ausführen
- [ ] können VS Code installieren und für die Verwendung von Python einrichten
- [ ] können Variablen und Datentypen (int, float, str, bool) korrekt einsetzen (Casten)
- [ ] können einfache Python-Applikationen mit EVA (Eingabe Verarbeitung und Ausgabe) realisieren
- [ ] können Verzweigungen mit if, elif, else umsetzen
- [ ] einfache Alltagsprobleme (ohne Iterationen) algorithmisch beschreiben und in Python umsetzen

Ablaufstruktur

Done	Zeit	Phase	Inhalt / Methode	Unterlagen
☐	18:30 - 18:45	Einstieg	*) Begrüssung *) Vorstellung LP, *) Vorwissen Programmiersprachen Lernende *) Erwartungen Lernende	Präsentation, Menti Block 01 - Abend 01
☐	18:45 - 19:15	Wichtige Begriffe	*) Algorithmus *) Kompilierend vs interpretierend *) Gängige Programmiersprachen kennen	Klassennotizbuch Block 01 - Abend 01
☐	19:15 -	Installation und	*) die aktuelle Version von Python installieren und mit den	Klassennotizbuch

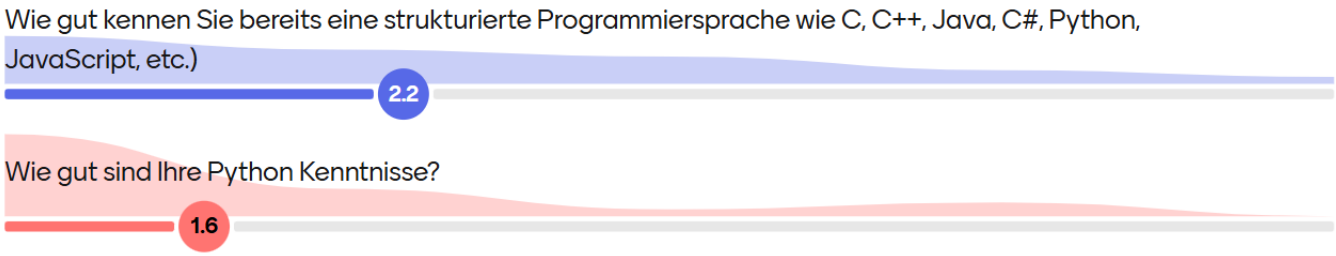
1.3 Vorwissen

Ich interessiere mich für dein Vorwissen bezüglich:

- der generellen Thematik Programmierung
- der Programmiersprache Python

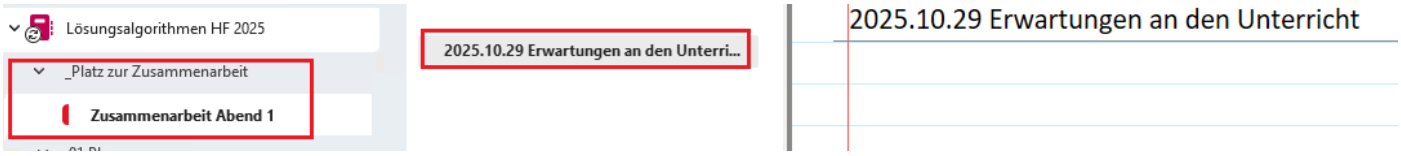
Dazu habe ich ein Mentimeter vorbereitet. Bitte gehe auf die Webseite www.menti.com (QR-Code rechts) und gib den Code 4278 8008 ein, um zu den Fragen zu kommen:

Link zur Auswertung für die Lehrperson: https://www.mentimeter.com/app/dashboard



1.4 Erwartungen an den Unterricht des Faches LAP

Im Klassenotizbuch unter «_Platz für Zusammenarbeit» existiert eine vorbereitete Seite, bei der du deine Erwartungen an den Unterricht notieren kannst.
Nimm dir 5 Minuten Zeit um dort deine Erwartungen zu notieren und die Erwartung der anderen Studierenden zu lesen.

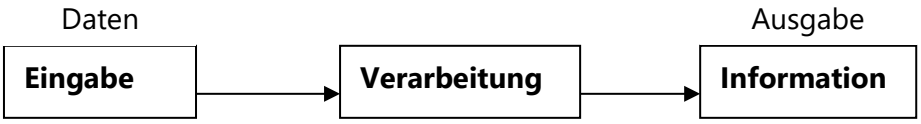


2 Wichtige Begriffe

2.1 Algorithmus

Viele Problemstellungen sind umfangreich und kompliziert. Deshalb erfordern sie eine systematische Vorarbeit, um den Lösungsweg zu beschreiben. Die strukturierte Programmierung verlangt eine gut durchdachte Vorbereitung, bevor Sie mit dem eigentlichen Programmieren (Codieren) beginnen.

In der untenstehenden Grafik sehen wir die Hauptgliederung eines Programms. Es besteht aus Daten, welche dem Programm als Eingabe dienen, einer Datenverarbeitung (nach einem bestimmten Algorithmus) und einer Ausgabe der Informationen. Man spricht auch vom EVA-Prinzip (**E**ingabe-**V**erarbeitung-**A**usgabe)



Was ist ein Algorithmus:
Erklären Sie mit eigenen Worten, was ein Algorithmus ist. Nehmen Sie dazu, falls nötig, ein Online-Lexikon oder eine KI zur Hand.

Ein Bild sagt bekanntlich mehr als tausend Worte, deshalb existieren verschiedene grafische Möglichkeiten um die Programmlogik darzustellen. Mit deren Hilfe lassen sich Kontrollstrukturen und Verzweigungen anschaulich darstellen. Am 4. Abend werden wir die wichtigsten dieser Entwurfstechniken betrachten:
*) Programmablaufplan (https://de.wikipedia.org/wiki/Programmablaufplan)
*) Struktogramm nach Nassi Shneiderman (https://de.wikipedia.org/wiki/Nassi-Shneiderman-Diagramm)
Es existiert auch eine textuell-strukturierte Möglichkeiten der Programmlogik-Darstellung. Dies nennt man Pseudocode (https://de.wikipedia.org/wiki/Pseudocode)

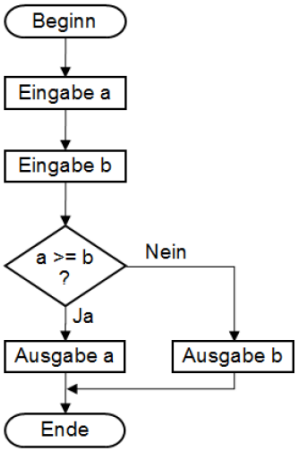


Abbildung: Programmablaufplan (PAP)

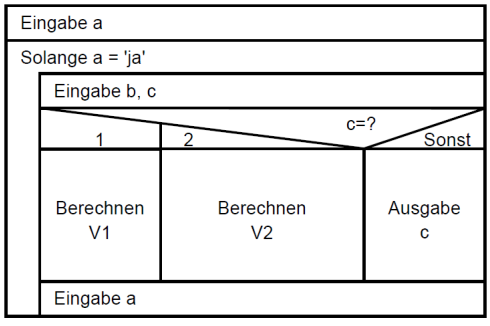


Abbildung: Struktogramm

```
begin
BetragPruefen
   Eingabe(a);
   Eingabe(b);
   if a >= b then
     Ausgabe(a);
   else
   Ausgabe(b);
   end if
 end BetragPruefen
```

Abbildung: Pseudocode

2.2 Kompilierende vs interpretierende Programmiersprachen

Der Zentralprozessor bzw. die ALU (Arithmetic and Logic Unit) eines Computers kann nur Maschinenbefehle lesen. Maschinenbefehle sind Elemente einer Sprache, die aus Zahlen bestehen (0 und 1) und sich deshalb schwer programmieren lassen. Aus diesem Grund wurden höhere Programmiersprachen entwickelt, die mit sprachlichen Elementen arbeiten. Programme, die in solchen Sprachen geschrieben sind, müssen in die Maschinensprache des jeweiligen Prozessors übersetzt werden. Die Aufgabe des Übersetzens übernimmt normalerweise die Software Interpreter oder Compiler.

Interpreter: übersetzen Programme Zeile für Zeile **zur Laufzeit** in Maschinensprache und werden vom Prozessor ausgeführt. Aus diesem Grund laufen die Programme **langsamer** als kompilierte Programme ab, und es können aus zeitlichen Gründen weniger Optimierungen des Programmcodes vorgenommen werden.

Compiler: übersetzen Programme vollständig in Maschinensprache und speichern das Ergebnis in einer ausführbaren Datei (z.B. exe oder dll). Während der Kompilierung optimiert der Compiler die Programmgrösse und -geschwindigkeit. Dadurch laufen diese Programme viel **schneller** ab als interpretierte Programme.

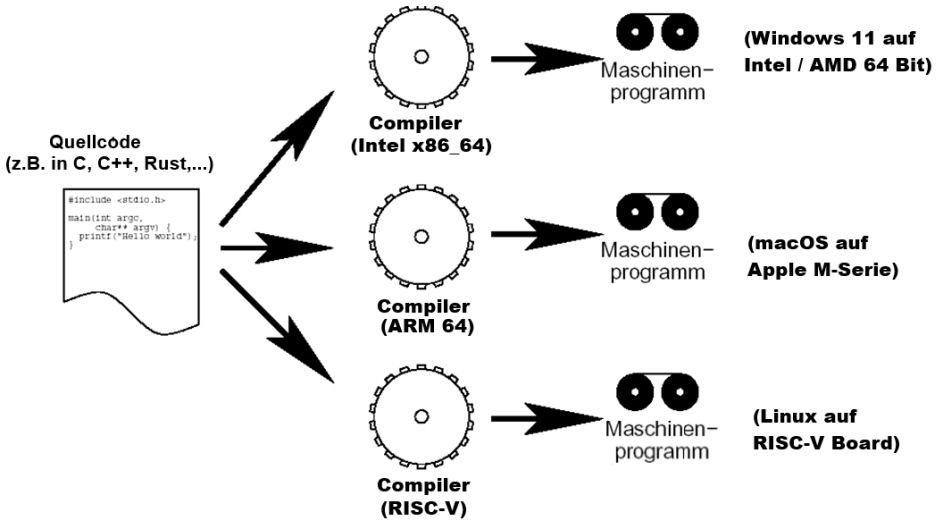
Man unterscheidet drei Arten/Typen, wie Programmcode ausführbar gemacht werden kann:

Reines Kompilieren Vorkompilation in Maschinencode	Quellcode wird in plattformspezifisches Maschinenprogramm übersetzt (z. B. C, C++, Rust)
Hybride Form / JIT-Kompilieren (Bytecode + Laufzeitumgebung)	Quellcode → Bytecode → JIT-Compiler wandelt während der Laufzeit in Maschinencode um (z. B. Java, Kotlin, C#)
Reines Interpretieren (Direkte Ausführung des Quellcodes)	uellcode wird zeilenweise interpretiert und sofort ausgeführt (z. B. Python, JavaScript, PHP)

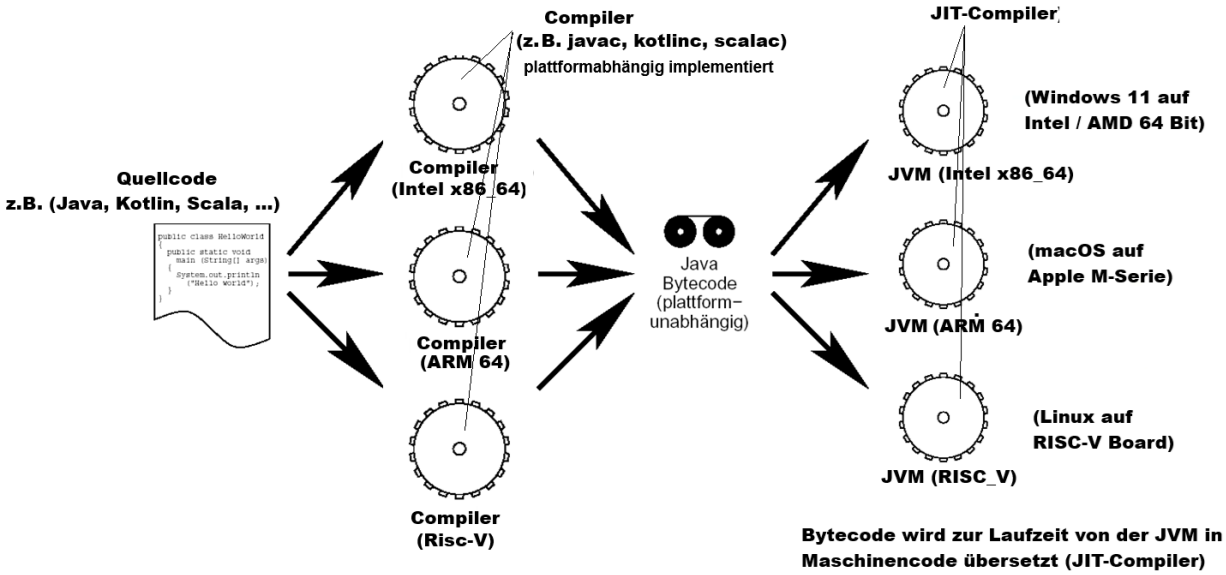
Der Interpreter führt den Quellcode *direkt* aus. Der **JIT-Compiler** übersetzt *zur Laufzeit* in Maschinencode, um die Ausführung zu beschleunigen. Beide arbeiten *zur Laufzeit*, aber:

- der Interpreter führt den Quellcode *direkt* aus. Er erstellt keine Maschinencode-Datei. Der Maschinencode bleibt im Arbeitsspeicher.
- der JIT kompiliert dynamisch optimierte Maschinencode-Blöcke und speichert sie für Wiederverwendung (Performance-Optimierung)

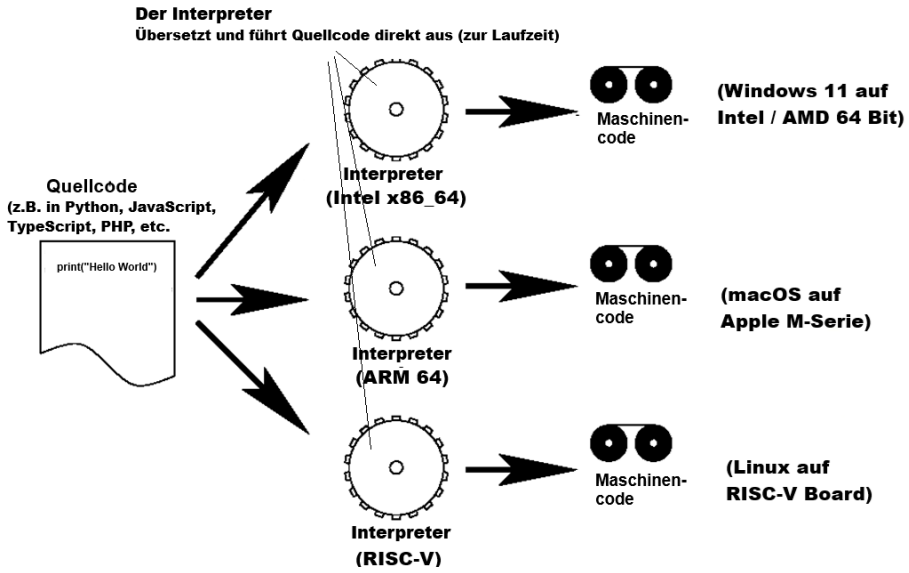
Typ I) Reines Kompilieren:



Typ II) Hybride Form / JIT-Kompilieren



Typ III) Reines Interpretieren



Python wird also interpretiert, nicht vorkompiliert. Der Interpreter erzeugt intern Bytecode und führt ihn direkt (zur Laufzeit) aus.

Java Virtual Machine /JVM) vs Python Virtual Machine (PVM)

In Java bedeutet „Virtual Machine“: eine Software-Schicht, die auf realer Hardware läuft und dort eine virtuelle CPU simuliert, die den Bytecode versteht und ausführt. Die JVM (Java Virtual Machine) ist also:

- eine vollständige Ausführungsumgebung,
- mit eigenem Speicher-Management, Bytecode-Verarbeitung, JIT-Compiler, Garbage Collector usw.
- und sie sorgt dafür, dass derselbe Bytecode auf verschiedenen Plattformen läuft.

Kurz: Die JVM verhält sich wie eine „virtuelle Hardware für Java-Programme“.

Bei Python ist der Begriff „Virtual Machine“ weniger formal. Man meint damit den Teil des Python-Interpreters, der den Bytecode ausführt — also den Laufzeitkern. Die PVM ist keine virtuelle CPU wie in Java, sondern eher ein Softwaremodul, das den Bytecode interpretiert und ausführt. Die PVM:

- liest den Python-Bytecode (aus .pyc-Dateien),
- führt ihn Instruktion für Instruktion auf dem echten Prozessor aus,
- verwaltet Variablen, Stack, Garage Collection usw.

Aber sie emuliert keine Hardware und bietet keine standardisierte Laufzeitumgebung (wie die JVM).

Aufgabe 1 zu Kap. 2.2

01 Was geschieht beim kompilierten Modell (z. B. C, C++)?

02 Was unterscheidet das hybride Modell (JIT) von reinem Kompilieren?

03 Was macht der Interpreter bei jeder Ausführung?

04 Warum ist das Interpreter-Modell meist langsamer als das Compiler-Modell?

05 Warum kann ein Java-Programm auf verschiedenen Betriebssystemen laufen, ein C-Programm aber nicht?

2.3 Programmiersprachen, ein Überblick

(Dieses Subkapitel enthält Auszüge aus dem Herdt-Buch: Programmierung Grundlagen, ISBN: 978-3-98569-150-0)

Werfen wir einen genaueren Blick auf die Klassifizierung von Programmiersprachen aufgrund ihrer historischen Entwicklung.

Erste Generation: Maschinensprachen (Maschinencode)

Damit ein Computer Probleme lösen kann, muss ihm der Lösungsweg in einer ihm verständlichen Art und Weise mitgeteilt werden. Eine Maschinensprache erfüllt genau diese Bedingung. Intern arbeitet ein Computer mit einem Register aus Befehlen, die zusammen diese Maschinensprache bilden und mit seinem Prozessor verbunden sind.

Beschreibung	Sowohl die Operationen als auch die Daten werden vom Programmierer ausschliesslich als Bitfolge aus Nullen und Einsen eingegeben. Eine übliche Operation ist z. B. der Transport von Daten aus dem Speicher in ein Register.
Nachteile	Für jeden Computer müssen die Maschinenbefehle neu entwickelt werden, da diese Sprache von den Eigenschaften der Hardware (Prozessoren) abhängig ist. Programme in Maschinensprache sind schwer lesbar und mit einem hohen Programmieraufwand verbunden. Deshalb wird diese Sprache heute kaum mehr direkt eingegeben (allerdings werden auch heute noch alle Befehle für Computer letztendlich in Maschinenbefehle umgewandelt).
Beispiel	00011010 0011 0100

Zweite Generation: Assembler-Sprachen

Assembler-Sprachen sind wie Maschinensprachen an bestimmte Prozessoren gebunden. Übersetzungsprogramme, die Assembler-Programme in Maschinencode umwandeln, werden ebenfalls als Assembler bezeichnet.

Einsatz	Assembler-Sprachen werden überwiegend zur Programmierung der Hardware oder für schnelle, zeitkritische Programme eingesetzt.
Vorteile	Im Vergleich zur Maschinensprache bieten Assembler-Sprachen dem Programmierer durch Operationskürzel, z. B. ADD für addieren, wesentliche Erleichterungen. Da Assembler-Sprachen auf die maschinenspezifischen Besonderheiten des jeweiligen Computers abgestimmt sind, verbrauchen die Programme im Allgemeinen weniger Speicherplatz und sind meist auch schneller als ein entsprechendes Programm in einer anderen Programmiersprache.
Besonderheit	Die einzelnen Befehle der Assembler-Sprachen verwenden direkt die internen Befehle des Prozessors. Wer eine Assembler-Sprache erlernt, erfährt dabei viel über die Arbeitsweise des jeweiligen Prozessortyps.
Beispiel	ADD ax, 10 (=Addiere den Wert 10 zum Inhalt des Registers ax)

Dritte Generation: höhere Programmiersprachen (prozedural, objektorientiert, funktional,..)

Anstoss zur Weiterentwicklung der Programmiersprachen der dritten Generation gaben die mangelhafte Eignung der maschinenorientierten Sprachen zum Erstellen komplexer Anwendungsprogramme und die schlechte Lesbarkeit für Menschen.

Beschreibung	Prozedurale Sprachen sind (weitgehend) unabhängig von einem Computersystem. Da Programmiersprachen ab der dritten Generation vom spezifischen Computersystem abstrahieren, werden sie auch als höhere Programmiersprachen bezeichnet. Damit ein Computer ein Programm in einer höheren Programmiersprache versteht, muss das Übersetzungsprogramm (Compiler oder Interpreter) an das jeweilige System angepasst sein und den entsprechenden Maschinencode erzeugen. Denn der Prozessor muss auch beim Einsatz einer höheren Programmiersprache auf ihn speziell abgestimmten Maschinencode „vorgesetzt" bekommen – nur den versteht ein Prozessor.

Einsatz	Die Sprachen der dritten Generation sind in ihrer Struktur und ihrem Befehlsvorrat auf bestimmte Anwendungsbereiche zugeschnitten. Allerdings nimmt die Tendenz zu, diese Sprachen immer umfassender auszugestalten.
Vorteile	Höhere Programmiersprachen sind im Allgemeinen leichter zu erlernen als maschinenorientierte Sprachen. Der Programmcode kann auch bei anderen Rechnersystemen wiederverwendet werden – bei einigen (älteren) Sprachen müssen allerdings bei einer Portierung einige Anpassungen vorgenommen werden.
Nachteile	Das Programm der höheren Programmiersprache verbrauchte früher mehr Speicherplatz und war meist auch langsamer als das vergleichbare Maschinenprogramm. Durch optimierte Übersetzung verschwinden diese Nachteile aber bei vielen modernen höheren Programmiersprachen.
Programmiersprachen	Python, Cobol, RPG, Fortran, Pascal, PL/1, Basic, Ada, C/C++, Java und viele mehr
Beispiel	flaeche = laenge * breite;

Vierte Generation: deklarative / problemorientierte Sprachen

Während die ersten drei Generationen noch relativ klar voneinander getrennt werden können, fehlen bei der folgenden Generation eindeutige Kriterien.

Beschreibung	Bei 4GL-Programmiersprachen beschreibt der Programmierer lediglich, was das Programm leisten soll, ohne den genauen algorithmischen Weg anzugeben. Eine einzelne Anweisung löst eine ganze Folge von internen Einzelschritten aus.
Einsatz	4GL-Programmiersprachen sind auf spezielle Anwendungsgebiete ausgerichtet, z. B. auf das Bearbeiten und Auswerten von Dateien, Datenbanken, Tabellenkalkulationen oder auf das Erstellen von Bildschirmformularen
Vorteile	Zum Programmieren stehen fortschrittliche, einfach zu bedienende und leistungsfähige Entwicklungssysteme zur Verfügung. Die Erstellung eines Programms wird dadurch wesentlich erleichtert.
Nachteile	Die grösstmögliche Effizienz der Programme ist nicht gegeben. Da bestimmte Folgen von Arbeitsschritten innerhalb einer Anweisung automatisch ausgeführt werden, hat der Programmierer kaum einen Einfluss auf die internen Abläufe. Die Ausführungsgeschwindigkeit dieser Programme ist aufgrund der mächtigeren Sprache langsamer als bei prozeduralen Sprachen.
Programmiersprachen	SQL, Natural, Symphony, Open Access
Beispiel	CREATE Adresskartei SELECT Kunde FROM Tabelle WHERE KdNr = 10

Fünfte Generation: Künstliche Intelligenz

Beschreibung	Der Grundgedanke des Forschungsgebietes der künstlichen Intelligenz (KI) ist es zu untersuchen, unter welchen Bedingungen Computer menschliche Verhaltensweisen, die auf Intelligenz beruhen, nachvollziehen können. Für diese Forschungszwecke sind einige spezielle Programmiersprachen entwickelt worden. Diese Sprachen gehören meist zu den logischen oder funktionalen Programmiersprachen, die Sie im weiteren Verlauf dieses Kapitels kennenlernen. KI als Konzept ist jedoch nicht an konkrete Sprachen gebunden, obwohl manche Sprachen sich besonders gut zur Umsetzung eignen. Aktuell gewinnt KI allgemein in der Praxis immer mehr an Bedeutung.
Einsatz	Inzwischen umfasst dieses Gebiet mehrere Fachbereiche, beispielsweise die Robotik, die zur Entwicklung von Robotern und deren komplizierten Bewegungsabläufen geführt hat, die Wissensverarbeitung und die Spracherkennung.
Programmiersprachen	Prolog, Lisp, Smalltalk
Beispiel (Prolog)	grossvater(X,Y) :- vater(X,Z), vater(Z,Y). Bedeutung: X ist der Grossvater von Y, wenn es eine Person Z gibt, die Kind von X und Vater von Y ist.

Kompakt:

1. Generation (1GL): Maschinensprache
Programmierer schreiben direkt Binärcode, der vom Prozessor ausgeführt wird. Jede Anweisung entspricht einem Hardwarebefehl. Beispiel: 10110000 01100001
Extrem schnell, aber kaum lesbar und fehleranfällig.

2. Generation (2GL): Assemblersprache
Maschinenbefehle werden durch mnemonische Kürzel ersetzt (z. B. MOV, ADD, JMP). Der Code ist symbolisch, aber noch hardwareabhängig. Beispiel:
```
MOV AX, 01
ADD BX, AX
```
Besser lesbar, aber immer noch sehr nah an der Hardware.

3. Generation (3GL): Höhere, prozedurale Sprachen
Programmierer schreiben strukturierte, menschenähnliche Anweisungen mit Kontrollstrukturen (if, for, while) und Funktionen. Beispiel: C, Java, Python, Pascal, Fortran
Lesbar, portabel, gut strukturiert – Fokus auf Wie ein Problem gelöst wird.

4. Generation (4GL): Problem- / anwendungsorientierte Sprachen
Ziel: Weniger Code, mehr Automatisierung. Oft domänenspezifisch (z. B. Datenbanken, Statistik).
Der Programmierer beschreibt Was er erreichen möchte – nicht mehr, Wie.
→ Beispiel: SQL, R, MATLAB, SAS
Schnellere Entwicklung, weniger Flexibilität, oft auf bestimmte Aufgaben spezialisiert.

5. Generation (5GL): Wissens- & KI-basierte Sprachen
Programme sollen selbständig schlussfolgern oder lernen, anstatt exakte Befehle auszuführen.
Basieren auf Logik, Regeln oder neuronalen Netzen. Beispiel: Prolog, LISP, Mercury, moderne KI-Frameworks
Fokus auf Wissen und Entscheidungslogik statt auf Ablaufsteuerung.

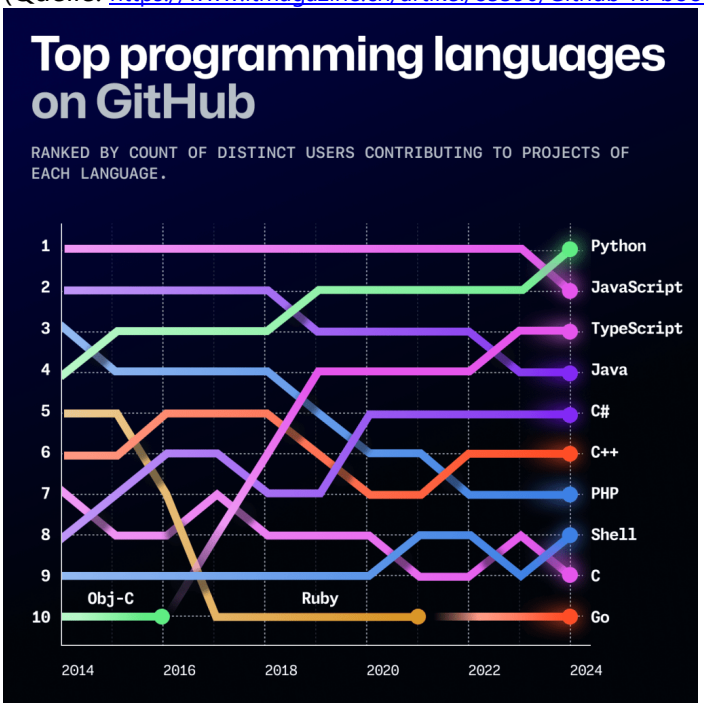
Generation	Programmierstil	Fokus	Beispiel
1GL	Maschinencode	Hardwarebefehle	Binär 0/1
2GL	Assembler	Symbolische Befehle	MOV, ADD
3GL	Prozedural / Strukturiert	Wie wird das Problem gelöst?	C, Java, Python
4GL	Deklarativ / Problemorientiert	Was soll geschehen?	SQL, R
5GL	Logik- & Wissensbasiert	Warum & mit welchen Regeln	Prolog, LISP

Historische Entwicklung

Das nachfolgend verlinkte Video zeigt die meistverwendeten Programmiersprachen während den Jahren 1965 und 2019. https://www.youtube.com/watch?v=Og847HVwRSI

Im Jahr 2024 hat Python JavaScript auf Platz 1 verdrängt als meistgenutzte Sprache auf GitHub

(Quelle: https://www.itmagazine.ch/artikel/83390/Github_KI_boomt_Schweiz_sticht_dank_Open_Source_aus_der_Masse.html)



2.4 Warum Python?

Python ist die ideale Sprache für unseren Kurs. Python ist leicht zu lernen, universell einsetzbar und ein direkter Türöffner zu weiterführenden Kursen wie OOP, DBB, Data Science, etc.

- Python wurde entwickelt, um Lesbarkeit und Verständlichkeit zu fördern.
- Die Syntax ist nahe an der natürlichen Sprache, was den Fokus auf Algorithmisches Denken statt auf komplizierte Schreibweise legt. Beispiel:

```
if x > 10:
    print("Grösser als 10")
```

  → kein unnötiger Syntax-Ballast (z. B. geschweifte Klammern oder Semikolons).
- Python ist eine der vielseitigsten Programmiersprachen überhaupt:
  Sie wird in Webentwicklung, Datenanalyse, Künstlicher Intelligenz, Automatisierung, Simulation, Systemtechnik und Lehre eingesetzt. Das ist ideal für HF-Studierende, da sie später in verschiedensten Modulen (z. B. OOP, DBB, Data Science) damit weiterarbeiten können
- Es existieren zehntausende Module und Bibliotheken (z. B. NumPy, Pandas, Matplotlib, TensorFlow …).Das ermöglicht, auch komplexe Themen wie Statistik, KI oder grafische Darstellungen mit wenig Code zu realisieren. Für dich als Lernende heisst das: *Schnell produktive Ergebnisse sehen.*
- Python nimmt dir viele technische Details ab (z. B. Speicherverwaltung, Datentyp-Deklaration). Dadurch können wir uns im Fach „Lösungsalgorithmen & Programmieren" auf das Wesentliche – die Logik der Algorithmen – konzentrieren. Diese Denkweise lässt sich später leicht auf andere Sprachen (C, C++, Java, C#) übertragen.
- Laut den meisten Rankings (TIOBE, Stack Overflow, IEEE Spectrum) gehört Python seit Jahren zu den Top 3 weltweit genutzten Sprachen. Es ist offen, kostenlos und plattformunabhängig (Windows, macOS, Linux). Wird in Bildung, Forschung und Industrie gleichermassen eingesetzt – also zukunftssicher.

3 Installation Python

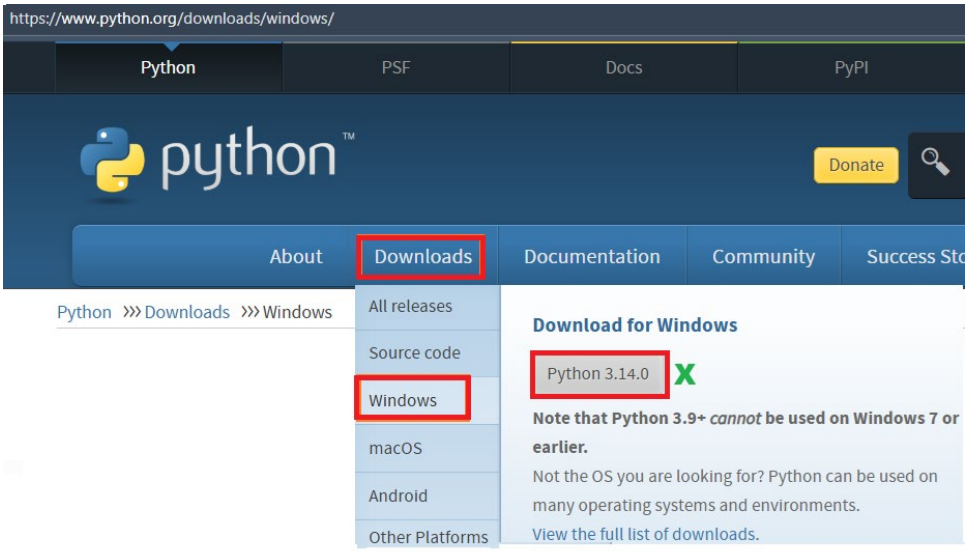
3.1 Python herunterladen und installieren (unter Windows)

Starte das Konsolenfenster und teste mit dem Befehl «python --version», ob Python bereits installiert ist:

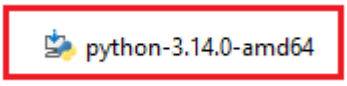
```
C:\Users\rolan>python --version
Python 3.14.0
```

Python kannst du von der Webseite *http://www.python.org*, *dem* zentralen Webportal von Python, herunterladen. Hier findet man die wichtigsten Informationen zu Python und die notwendigen Ressourcen. Über den Menüpunkt Downloads erhältst du die aktuelle, aber auch frühere Versionen für verschiedene Betriebssysteme. Empfehlenswert ist, immer die neuste Version von Python für Ihr Betriebssystem herunterzuladen.

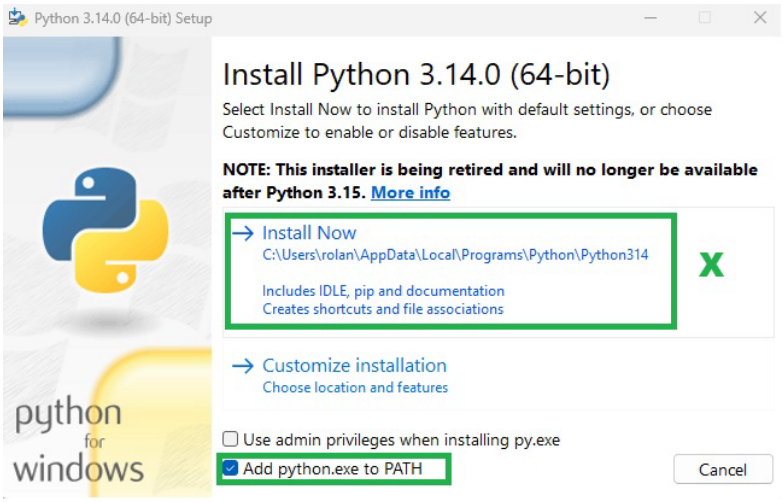
Sie sehen, dass Python auch für andere Plattformen (Android, MacOS, etc.) bereitsteht. Hier betrachten wir die Installation unter Windows. Laden Sie die neuste Version für Windows herunter.



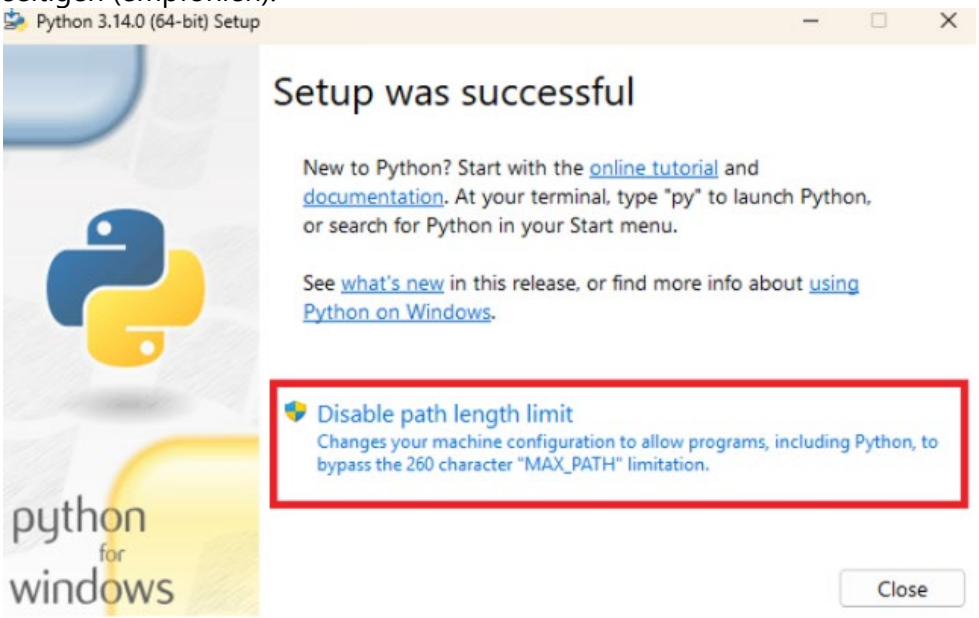
Starten Sie danach die heruntergeladene Installationsanwendung:



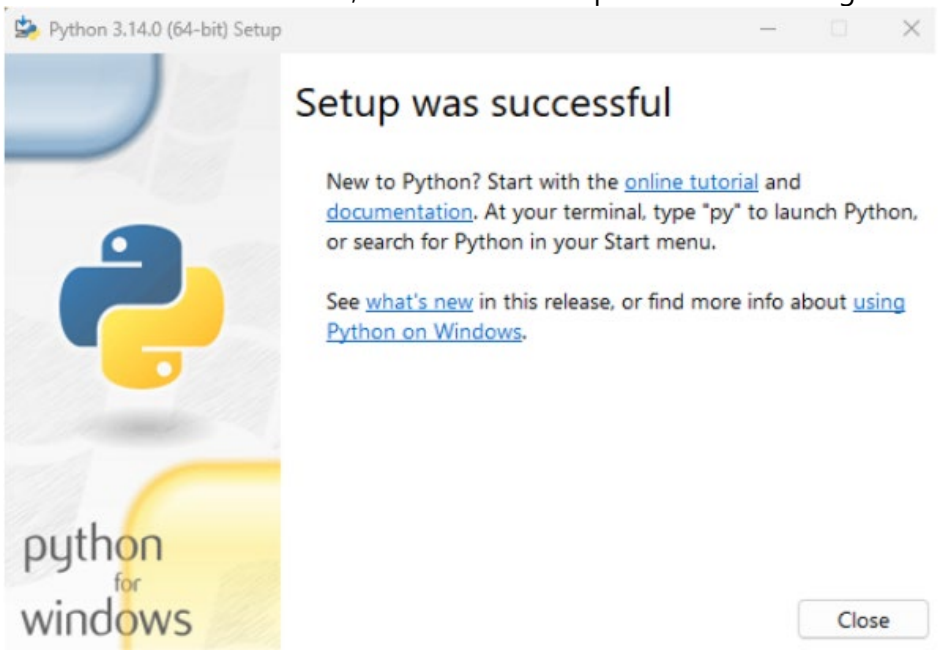
Wichtig ist das Hinzufügen von Python zur Path-Angabe, dem Suchpfad von Windows. Klicke dazu im Dialog die entsprechende Option an:



Falls auf deinem Betriebssystem noch eine Path Limitierung auf 260 Zeichen existiert, erscheint eine entsprechende Meldung und man kann mit dem Klick auf den rot markierten Bereich diese Einschränkung beseitigen (empfohlen).



Wenn alles funktioniert hat, erscheint die entsprechende Meldung:



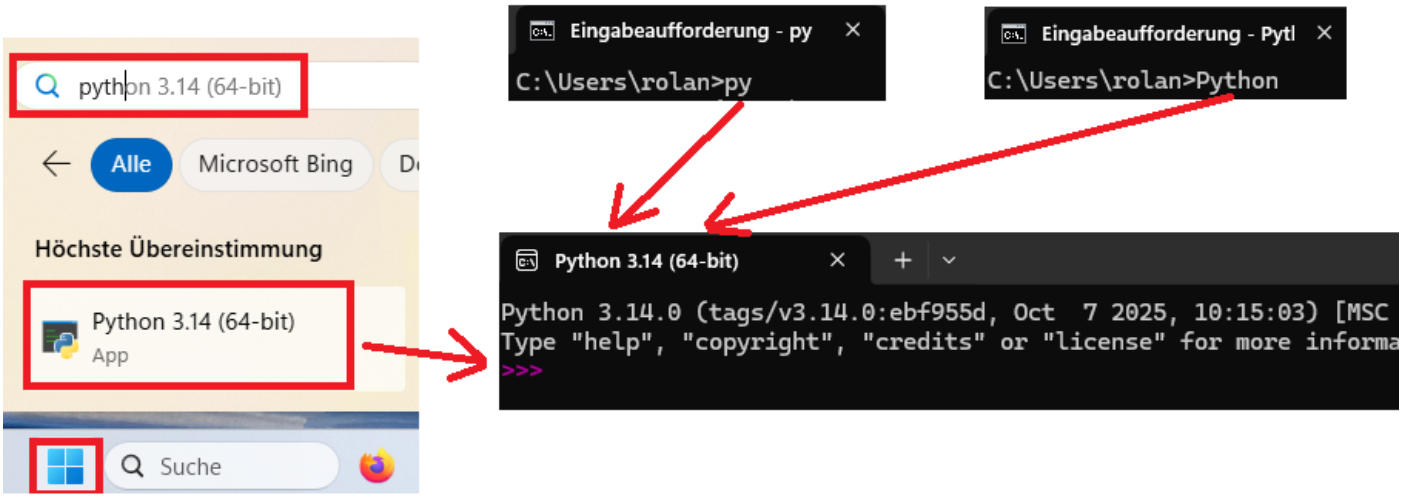
Teste erneut via Konsolenfenster und dem Befehl «python --version», ob Python sich in der installierten Version meldet:

```
C:\Users\rolan>python --version
Python 3.14.0
```

3.2 Python im Interaktivmodus verwenden

Wenn Python installiert ist, kann man im Interaktivmodus bereits mit Python arbeiten. Den Interaktivmodus erkennt man am Eingabeprompt «>>>» (siehe Bilder unten).
Es gibt versch. Wege zum Interaktivmodus zu gelangen, die auf dem untenstehenden Bild dargestellt sind:

- Via Windowssymbol und Startmenü kann man Python im Suchfeld eingeben und als App öffnen
- Via Windowssymbol und Startmenü kann man im Suchfeld cmd oder Eingabeaufforderung eingeben. Im Konsolenfenster kann man dann den Befehl «py» oder «python» eingeben und mit <Enter> bestätigen.
- Via <Windows> Taste +<R> kann man cmd eingeben und den Befehl «py» oder «python» eingeben und mit <Enter> bestätigen.
- Via <Windows> Taste +<X> kann man Terminal wählen und im PowerShell-Fenster den Befehl «py» oder «python» eingeben und mit <Enter> bestätigen.



Start als Administrator
Wenn du das Terminal als Administrator öffnest und dann python startest, läuft Python mit erhöhten Rechten. Das bedeutet:

- Du kannst in geschützten Systemordnern schreiben,
- systemweite Pakete mit pip install installieren,
- auf privilegierte Systemressourcen zugreifen.
- Aber Vorsicht: Das ist nur nötig, wenn du tatsächlich systemweite Änderungen vornehmen willst. <Für normales Programmieren genügt ein normales Terminal völlig.

Den Python-Interaktivmodus verlassen
Durch den Aufruf der Funktionen quit() oder exit() verlässt man den Python Interaktivmodus.

```
>>> quit()
```

```
>>> exit()
```

Aufgabe 1 zu Kap. 3.2

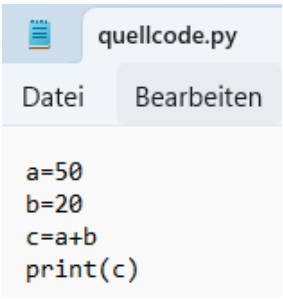
Machen Sie sich mit dem Wechsel zum Python Interaktivmodus vertraut.
a) Öffnen Sie auf versch. Wegen den Interaktivmodus und Schliessen sie diesen wieder.
b) Tippen Sie das folgende kleine Programm (Zeile für Zeile) ein um testen, ob Python korrekt funktioniert.

```
>>> a=50
>>> b=20
>>> c= a+b
>>> print(c)
70
>>>
```

3.3 Pythonprogramme mit dem Editor erstellen und ausführen

Als reine Eingabemöglichkeit des Quelltextes kann man im Grunde jeden Editor verwenden, der in einem typischen Betriebssystem vorhanden ist. Das möchten wir in diesem Subkapitel betrachten.

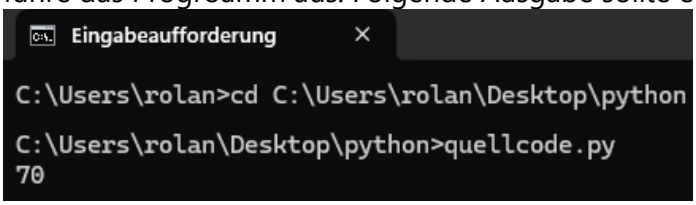
Öffne einen beliebigen Editor (z.B. Editor App von Wondows) und tippe das kleine Programm vom Kapitel 3.2 im Editor ab. Vorsicht, sei Exakt bezüglich Leerschlägen und Gross und Kleinschreibung.
Speichere das Dokument an beliebiger Lokation als MyFirstPythonApp.py ab (Die Extension .py ist wichtig).
Schliesse den Editor und starte das gespeicherte Pythonprogramm durch einen Doppelklick.



Das Programm wird ausgeführt, die Ausgabe erscheint aber es ist kaum erkennbar, da das Ausführungsfenster gleich wieder geschlossen wird.

Aufgabe 1 zu Kap. 3.3

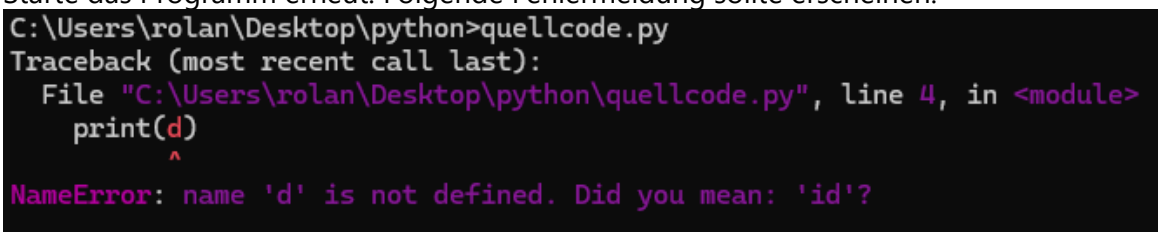
Öffne das Pythonprogramm erneut, diesmal aber via Konsolenfenster. Wechsel mit cd (**c**hange **d**irectory) zum Pfad der Lokation des Programms (Diesen Pfad kann man auch per drag & drop hineinziehen) und führe das Progroamm aus. Folgende Ausgabe sollte erscheinen:



Öffne das Pythonprogramm erneut im Editor (rechte Maustaste auf die Datei → öffnen mit→ Editor) und baue einen Fehler ein (wir geben die Variable d aus, die gar nicht erstellt wurde).

```
a=50
b=20
c=a+b
print(d)
```

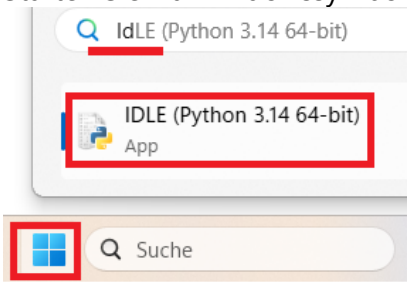
Starte das Programm erneut. Folgende Fehlermeldung sollte erscheinen:



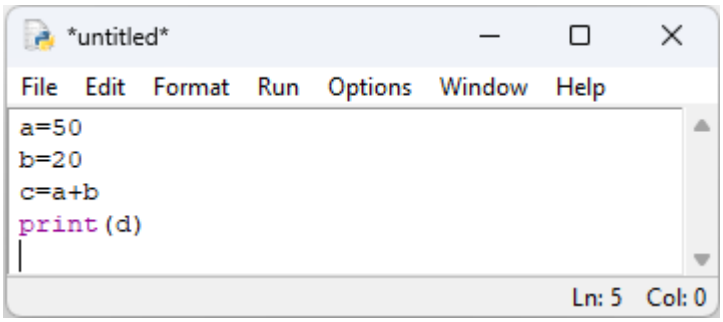
3.4 IDLE von Python verwenden

DAls Beispiel für eine komfortable integrierte Entwicklungsumgebung werden wir erst **IDLE** (Integrated Development and Learning Environment) verwenden. Diese gehört zum Python-System und wird meist mit Python automatisch installiert. Die Verwendung vereinfacht die Arbeit mit Python erheblich.
Später (ab Kapitel 5) werden wir die Entwicklungsumgebung Visual Studio Code installieren und verwenden. Dort muss man jedoch in der Regel die Unterstützung für eine Sprache wie Python nachträglich über Erweiterungen installieren.

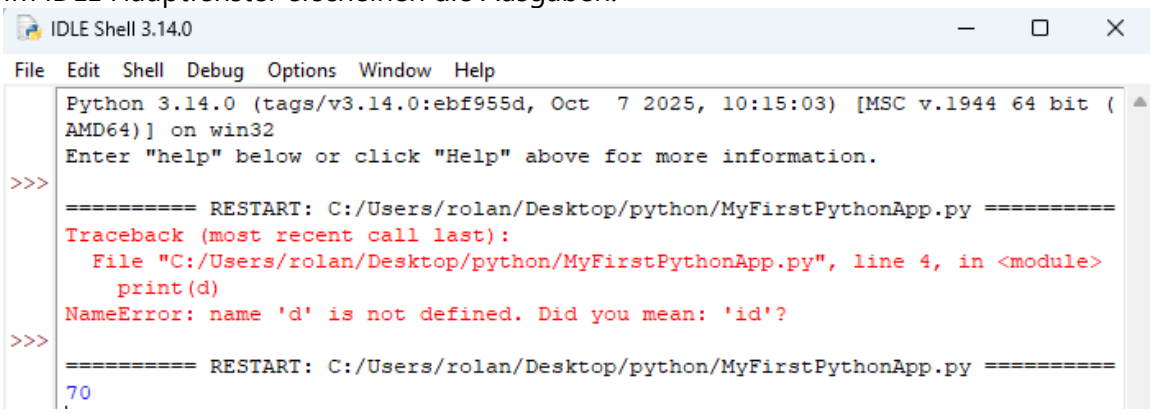
Starten Sie via Windowssymbol und Startmenü die integrierte Entwicklungsumgebung IDLE):



Um ein neues Programm zu erstellen, wählen wir den Menüpunkt "File" → "New File". Tippen Sie dort das zuvor erstellte Proramm ein:



Speichern Sie das Programm via "File"→ "Save As" an einer beliebigen Lokation und führen Sie danach das Programm mit dem Menüpunkt "Run" → "Run Module" oder einfach mit der Taste <F5>.
Dies liefert natürlich eine Fehlermeldung, weil wir die Variable d ausgeben, die gar nicht existiert. Korrigiere den Fehler und starte es erneut.
Im IDLE Hauptfenster erscheinen die Ausgaben:



4 Mit den Grundelementen arbeiten

4.1 *Python-Dokumentation* und Hilfe

Generell ist die Python-Dokumentation auf folgender Webseite zugänglich: https://docs.python.org

Du kannst auch im *Python*-Interaktivmodus zur Hilfe wechseln indem du help eingibst:

```
>>> help
```

Du merkst, dass der Prompt sich ändert:

```
help>
```

Hier kannst du z.B. Informationen zu den keywords von Python erhalten:

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Auch könntest du genauere Beschreibung zur if Anweisung, die wir im Kapitel 6 betrachten werden erhalten:

```
help> if
The "if" statement
******************

The "if" statement is used for conditional execution:

   if_stmt: "if" assignment_expression ":" suite
            ("elif" assignment_expression ":" suite)*
            ["else" ":" suite]

It selects exactly one of the suites by evaluating the expressions one
```

Den Hilfebereich verlässt man durch Eingabe der Funktionen quit() oder exit().

4.2 Variablen, Datentypen und Literale

In Python kann ich durch die folgende Anweisung eine Variable erstellen: **a=50**

- a ist eine Variable
- der Wert 50 nennt man Literal
- die Variable wird mit dieser Anweisung erstellt (deklariert und initialisiert)

Entgegen streng typisierten Sprachen wie Java oder c / c++ muss man der Variable bei Python keinen Datentyp zuordnen (short, int, long, single/float oder double, ...). Python leitet den Datentyp automatisch aus dem Literal ab. Man nennt das Duck-Typing.

Während Sprachen wie Java / C, etc. über eine grosse Menge an Datenypen mit festgelegten Speichergrösse existieren, gibt es in Python nur wenige Grundtypen:

- int (beliebig grosse ganze Zahl)
- float (64-bit Gleitkommazahl)
- complex (komplexe Zahl, selten verwendet)
- str (String, Zeichenketten)
- bool (Wahrheitswert True oder False)

Auch char gibt es in Python **nicht** — einzelne Zeichen sind einfach **Strings der Länge 1**.
Python kümmert sich selbst um die Speicherverwaltung. Die Grösse einer Zahl hängt **nicht** von einem fixen Datentyp ab. Typen können sich dynamisch ändern (der graue Text ist Kommentar):

```
a = 5       # int
a = 5.0     # jetzt float
a = "fünf"  # jetzt str
```

Wenn Sie mehr über Datentypen erfahren möchten finden sie viele Infos dazu in der Python Dokumentation: https://docs.python.org/3/library/stdtypes.html

4.3 Anweisungen, Operatoren, Leerzeichen

Anweisungen sind Befehle an den Computer. Sie führen bestimmte Dinge aus. Aus Anweisungen entsteht ein Programmfluss.

Es gibt bei Python verschiedene Anweisungsarten. Man unterscheidet:

- Blockanweisungen (fassen versch. einzelne Anweisungen zusammen und führen sie als Block aus)
- Deklarationsanweisungen (Anweisungen die Dinge einführen, Variablen, Funktionen, etc.)
- Ausdrucksanweisungen (tun gewisse Dinge, z.B. berechnen z.B. etwas)
- Kontrollanweisungen (steuern den Programmfluss durch Sequenz, Selektion, Iteration, Sprung)
- Die leere Anweisung (pass, werden immer dann notiert, wenn eine syntax gefordert wird, aber man nichts konkretes tun möchte)

Operatoren (z.B. + für Addition, - für Subtraktion) verhalten sich je nach Variablentyp unterschiedlich. Einige wichtige Operatoren für nummerische Typen liste ich hier auf, damit wir grundlegende Arbeiten machen können ohne recherchieren zu müssen:

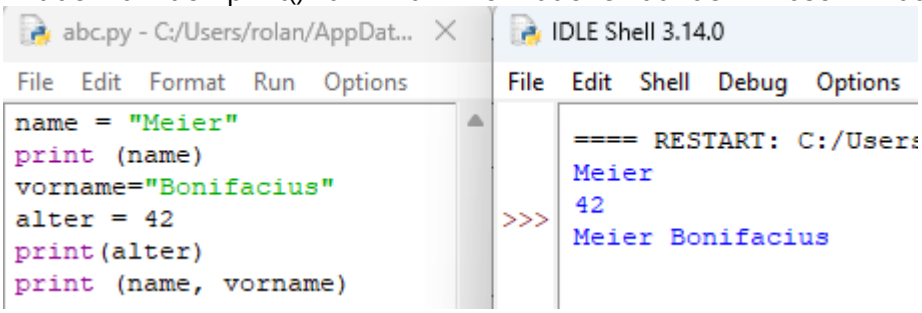
Operation	Bedeutung	Beispiel
x + y	Addition: x und y werden addiert	z=5+3 # ergibt 8
x - y	Subtraktion: x und y werden subtrahiert	z=5-3 # ergibt 2
x * y	Multiplikation: x und y werden multipliziert	z=5*3 # ergibt 15

x / y	Division: x und y werden dividiert	z=5/3 # ergibt 1.666...
x // y	Ganzzahlige Division	z=5//3 # ergibt 1
x % y	Modulo (Rest der ganzzahligen Division)	z=5%3 # ergibt 2
abs(x)	Absolutwert	z=abs(3-5) # ergibt 2
int(x)	Konvertiert den Inhalt der Variablen x in eine ganze Zahl	z=int(3.852) # ergibt 3
float(x)	Konvertiert den Inhalt der Variablen x in eine gebrochene Zahl	
pow(x, y)	rechnet x^y	z=pow(5,3) # ergibt 125
x ** y	rechnet x^y	z=5**3 # ergibt 125

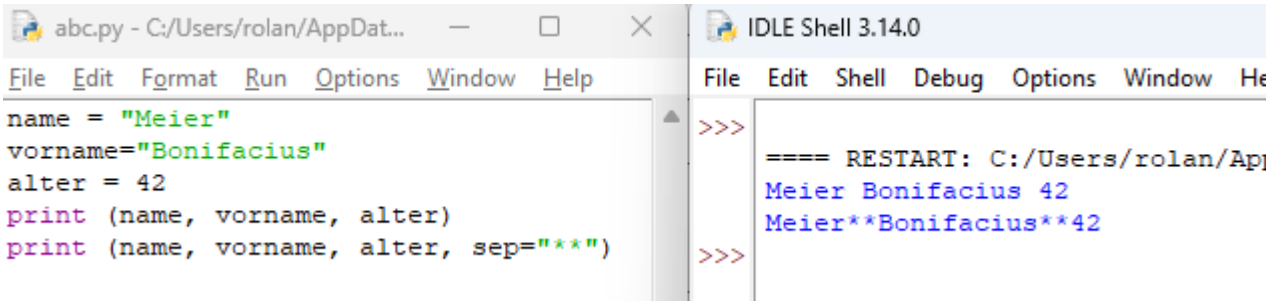
Leerzeichen haben bei Python eine höhere Bedeutung als einfach nur Token zu trennen wie sonst üblich bei anderen Programmiersprachen. Sie haben am Anfang einer Zeile eine bestimmte Bedeutung zur Bildung von Blöcken die zur Strukturierung von Blockanweisungen verwendet werden.

4.4 Ausgabe mit der print() Funktion

Mit der Funktion print() kann man Informationen auf dem Bildschirm ausgeben.



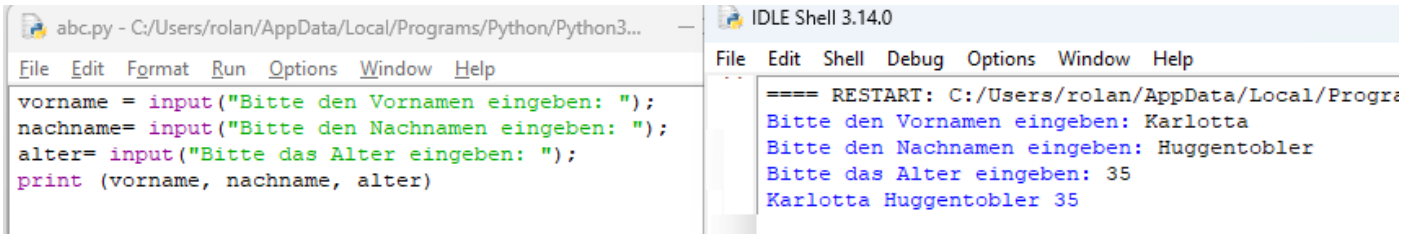
Der Funktion print() kann man beliebig viele Parameter übergeben. Alle übergebenen Werte werden durch ein Leerzeichen getrennt ausgegeben.
Das Standardverhalten der Funktion Print ist die Trennung durch ein Leerzeichen. Dieses Trennzeichen kann man aber ändern:



Sehr oft wird man bei Python-Standardfunktionen ein Vorgabeverhalten vorfinden, welches man durch Optionen/ Attribute anpassen kann.

4.5 Eingabe mit der Input() Funktion

https://docs.python.org/3/library/functions.html#input



Hingegen führt das folgende Skript nicht zu dem erwarteten Ergebnis:

```
z1 = input("Bitte Zahl 1 eingeben: ");
z2 = input("Bitte Zahl 2 eingeben: ");
print (z1 + z2)
```

```
Bitte Zahl 1 eingeben: 52
Bitte Zahl 2 eingeben: 38
5238
```

Dies liegt daran, dass die Input-Funktion immer eine Zeichenkette (String) zurückgibt. Der Operator + wurde beim Datentyp String so definiert, dass die Zeichenketten aneinandergehängt werden.
Wenn wir also rechnen möchten, müssen wir die Rückgabe der Inputfunktion umwandeln.

```
z1 = int(input("Bitte Zahl 1 eingeben: "));
z2 = int(input("Bitte Zahl 2 eingeben: "));
print (z1 + z2)
```

```
Bitte Zahl 1 eingeben: 52
Bitte Zahl 2 eingeben: 38
90
```

Teste was passiert, wenn man statt 52 und 38 gebrochene Werte wie 12.5 und 8.2 eingibt.

Bei der Eingabe von 12.5. erfolgt ein Fehler (Absturz), weil 12.5 nicht einfach so in einen ganzzahligen Wert umgerechnet werden kann:

```
Bitte Zahl 1 eingeben: 12.5
Traceback (most recent call last):
  File "C:/Users/rolan/AppData/Local/Programs/Python/Python314/abc.py", line 1,
in <module>
    z1 = int(input("Bitte Zahl 1 eingeben: "));
ValueError: invalid literal for int() with base 10: '12.5'
```

Wir lösen das Problem, in dem wir nicht die Funktion int( ) sondern float( ) verwenden:

```
z1 = float(input("Bitte Zahl 1 eingeben: "));
z2 = float(input("Bitte Zahl 2 eingeben: "));
print (z1 + z2)
```

```
==== RESTART: C:/Users/rolan/AppData
Bitte Zahl 1 eingeben: 12.5
Bitte Zahl 2 eingeben: 8.2
20.7
```

4.6 Build-in Functions

Print und Input sind nur zwei Funktionen einer vielzahl von bereits existierenden Funktionen (sog. Build-in functions). Der Python Interpreter kennt viele weitere integrierte Funktionen die verfügbar sind.
Eine komplette Liste finden sie in der Python Dokumentation (https://docs.python.org/3/library/) unter Build-in functions oder direkt hier: https://docs.python.org/3/library/functions.html

5 VS Code installieren & konfigurieren

https://www.jetbrains.com/pycharm/

5.1 Vergleich VS Code vs PyCharm

VS Code ist schnell, schlank und übersichtlich – perfekt für Einsteiger. Es lenkt den Fokus auf das Verständnis des Codes, nicht auf komplexe IDE-Menüs. Besonders im Unterricht ist das wichtig: weniger Klicks, mehr Programmieren

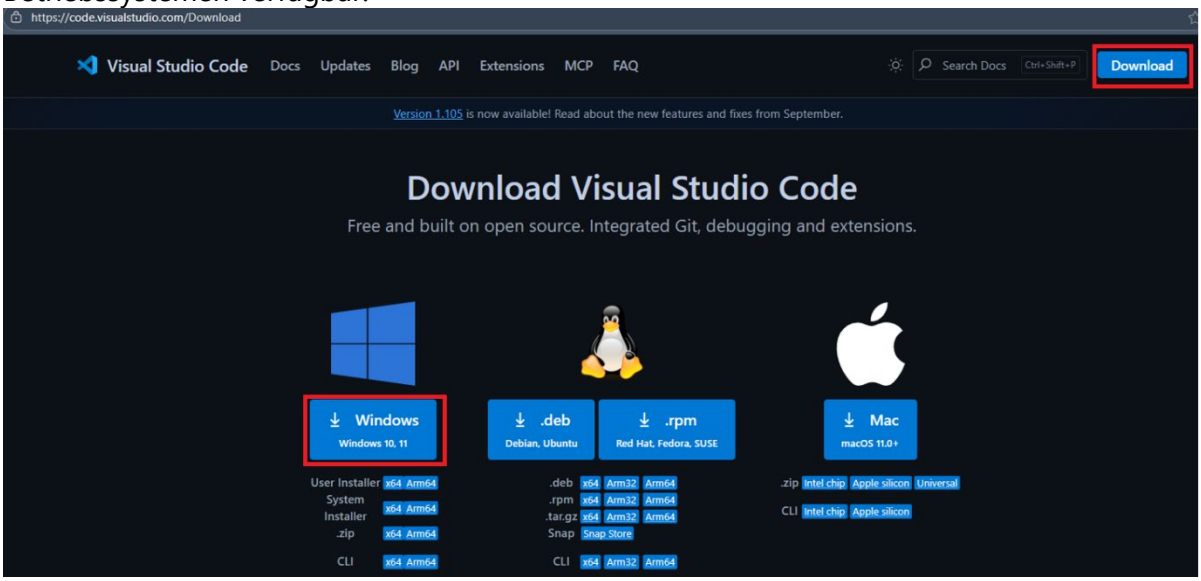
VS Code unterstützt alle wichtigen Programmiersprachen (Python, Java, C#, HTML, SQL usw.). Damit bleibt das Tool in späteren Modulen (OOP, DBB, Web Engineering, Data Science …) dasselbe. Studierende müssen nicht für jedes Fach eine neue IDE lernen.

VS Code ist komplett kostenlos, auch für Schulen und Unternehmen. Läuft zuverlässig auf Windows, macOS und Linux. Ideal für heterogene Geräte in der Klasse (eigene Notebooks der Studierenden).

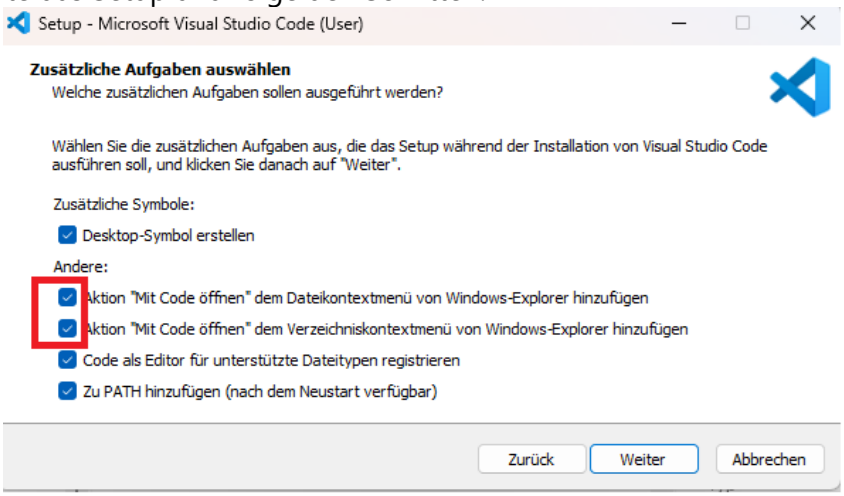
Wir verwenden VS Code, weil es leicht verständlich, universell, kostenlos und damit ideal für einen algorithmusorientierten Unterricht ist. Es lässt sich später für alle anderen Module weiterverwenden – und entspricht der modernen Arbeitsumgebung vieler Softwareentwickler.

5.2 VS Code herunterladen und konfigurieren

Die Installation von VSCode ist super einfach. Lade die stabile Version von der offiziellen Webseite https://code.visualstudio.com herunter und installiere sie. Das Installationsprogramm ist für alle Arten von Betriebssystemen verfügbar.

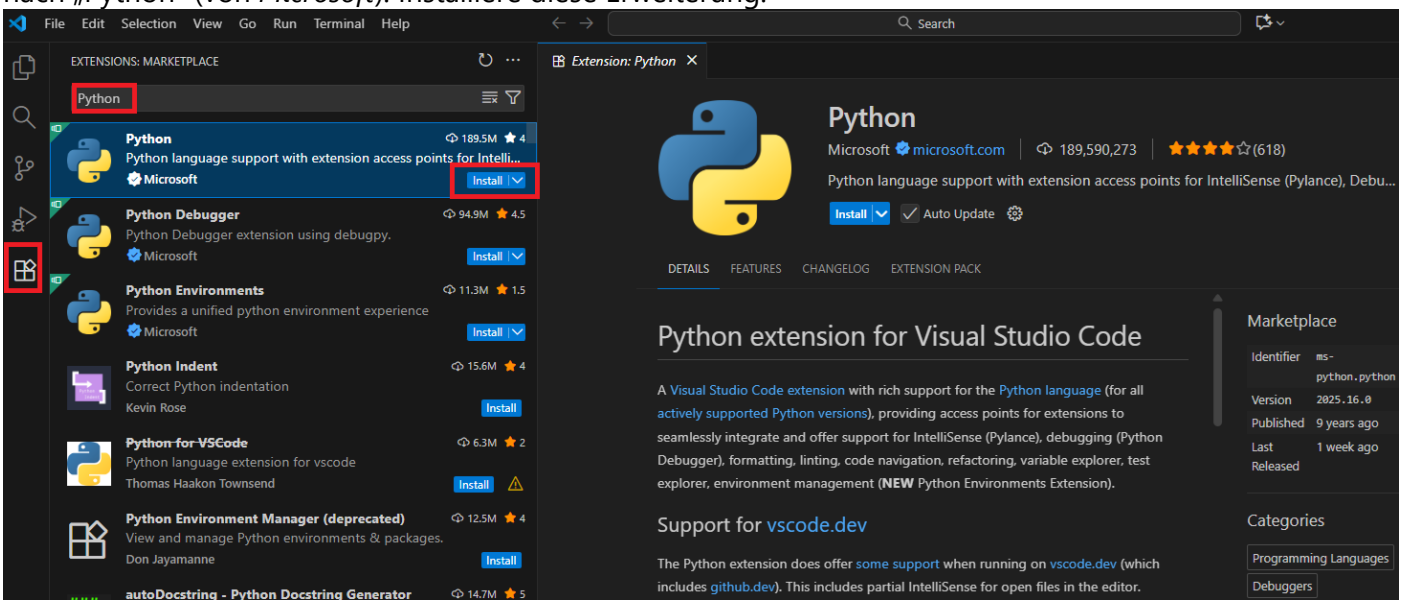


Starte das Setup und folge den Schritten.



Starte nach der Installation Visual Studio Code.

Öffne in VS Code links die Erweiterungen (Symbol mit den vier Quadraten oder Ctrl + Shift + X) und suche nach „Python“ (von *Microsoft*). Installiere diese Erweiterung.

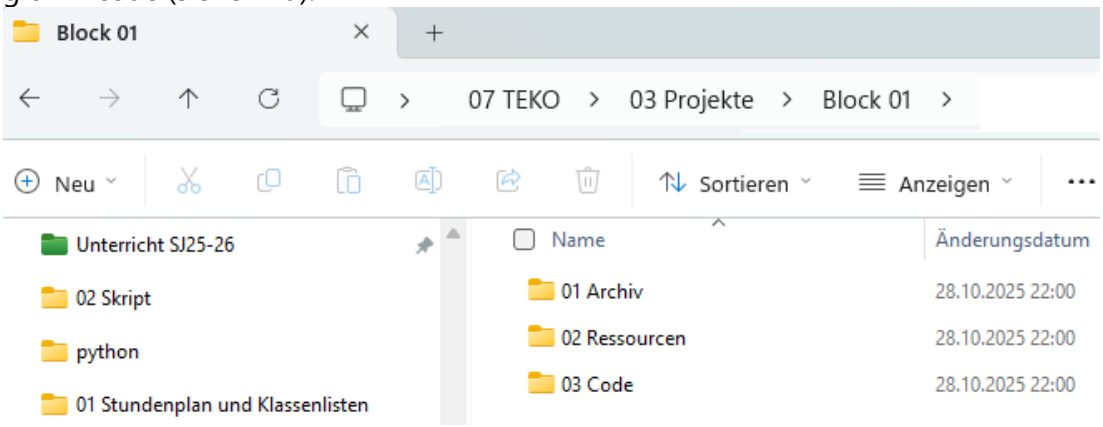


Wenn du die „Python“-Extension von Microsoft installierst (so wie auf dem Screenshot), werden mehrere Zusatzmodule automatisch mit installiert, u. a.:

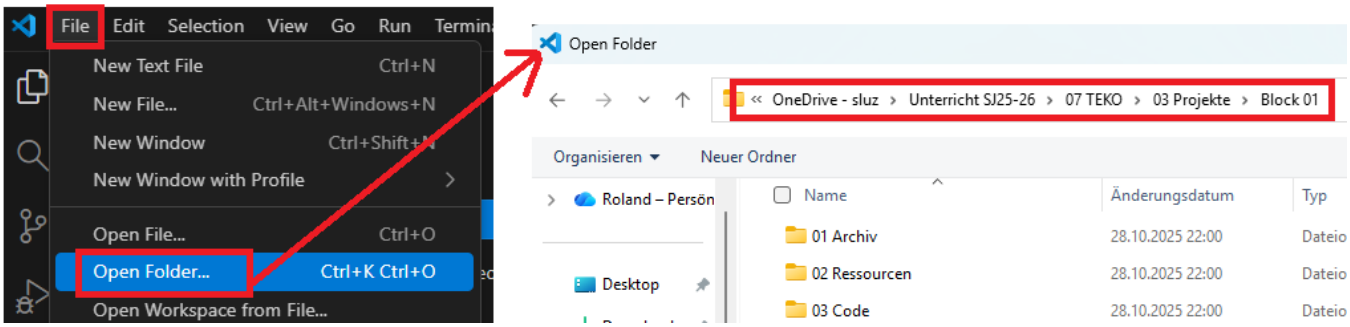
- Pylance → für IntelliSense, Typprüfung und Codeanalyse
- Python Debugger → für Breakpoints und Laufzeit-Debugging
- Python Environments → zum Verwalten mehrerer Python-Interpreter

5.3 Ein Python Programm mit VS Code schreiben und starten

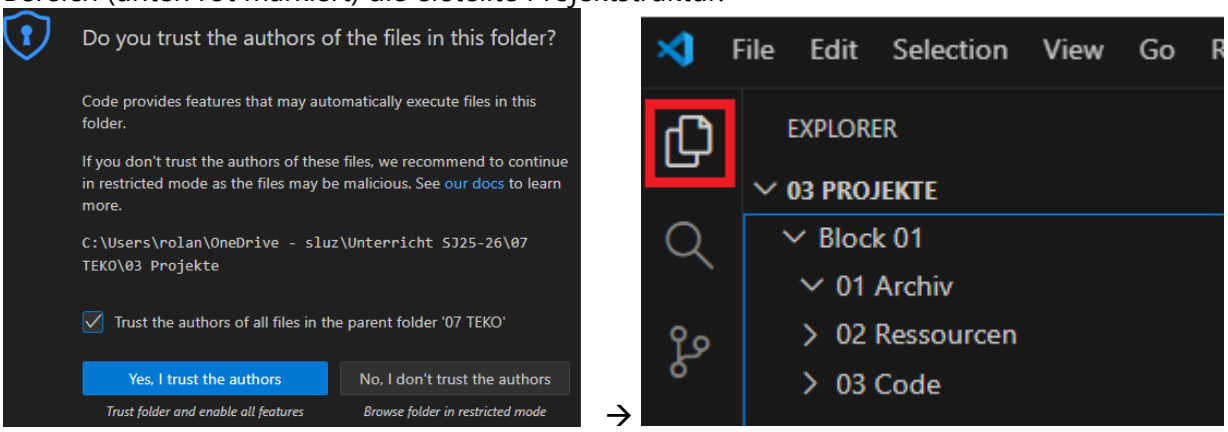
Erstelle mit Hilfe des Windowsexplorers an einer sinnvollen Lokation einen Folder für deine Python-Projekte. Sinnvoll ist dies z.B. irgendwo im persönlichen Sharepoint, damit die Dateien automatisch gesichert werden. Ich erstelle den Ordner 03 Projekte und mache einen Subordner für die Projekte des ersten Blockes (Ordner Block 01). Innerhalb dieses Ordners erstelle ich drei Unterordner für Archiv, Ressourcen und aktuellen Programmcode (siehe Bild):



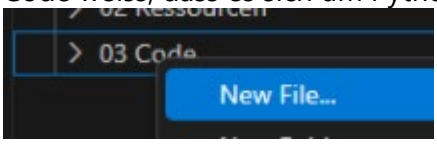
Danach öffne ich den Projektordner (Block 01) via Menüpunkt File→ Open Folder:



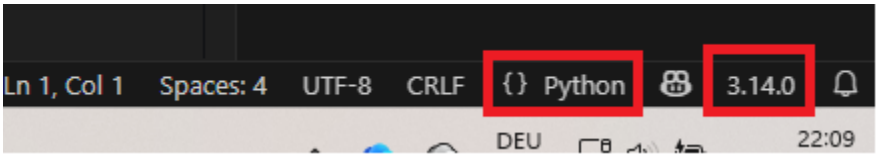
Die Frage, ob wir dem Autor des Ordners vertrauen können wir bestätigen. Danach sehen wir im Explorer Bereich (unten rot markiert) die erstellte Projektstruktur:



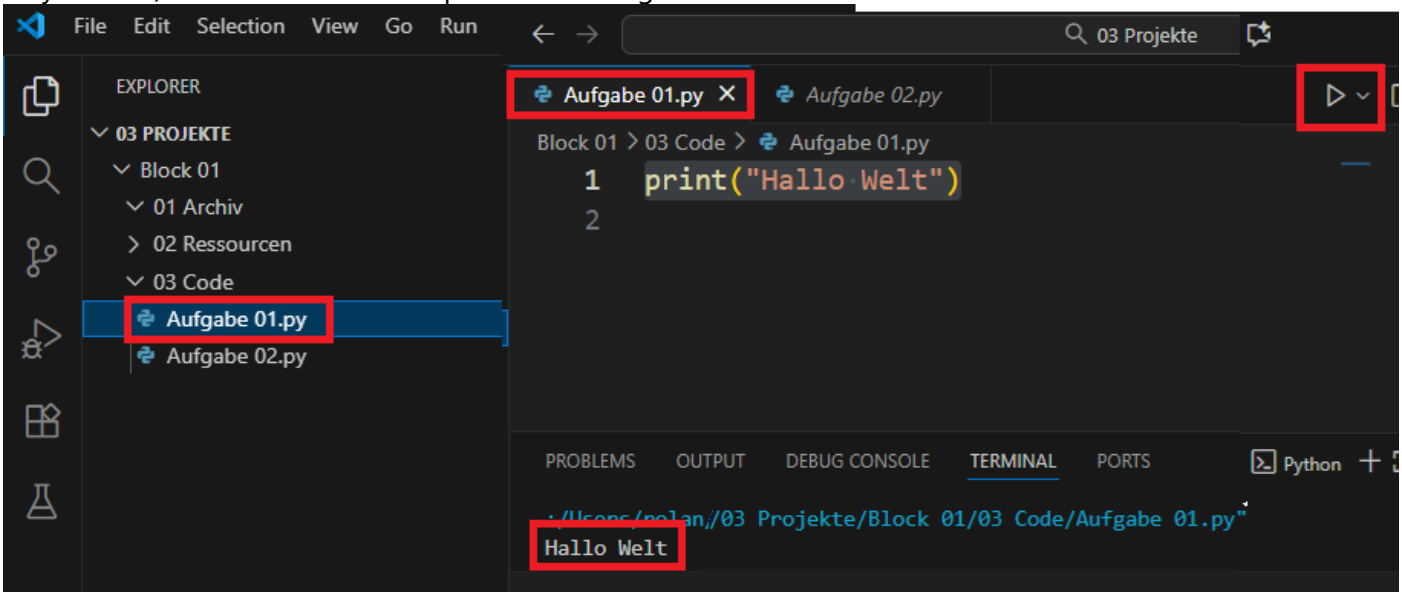
Via Kontextmenü (rechte Maustaste→ New File) fügen dem noch leeren Ordner Code eine Pythondatei hinzu. Nennen Sie die Datei z.B. Aufgabe01.**py**. Achtung die Extension .py ist wichtig, damit VisualStudio Code weiss, dass es sich um Pythoncode handelt:



In der rechten unteren Ecke von Visualstudio Code sollte die Sprache und die Interpreterversion korrekt erscheinen:

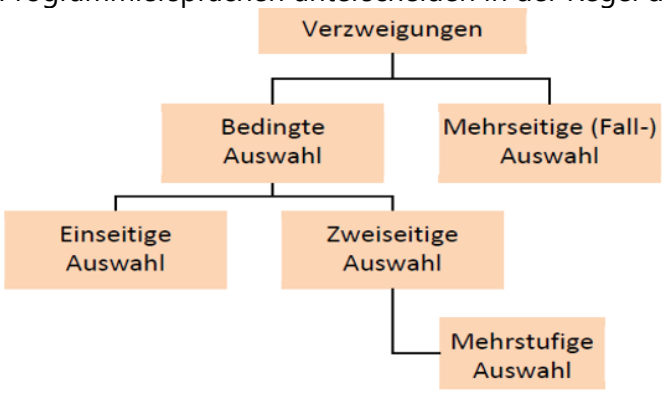


Schreiben Sie einen beliebigen Pythoncode in die Datei und führen Sie den Code aus (rechts oben mit dem Play-Button). Unten sollte die entsprechende Ausgabe erscheinen:



6 Verzweigungen mit Python realisieren

Programmiersprachen unterscheiden in der Regel die folgenden Verzweigungen:



6.1.1 Einseitige Verzweigung (if)

Bei vielen Problemstellungen ist die Verarbeitung von Bedingungen abhängig. Nur wenn die Bedingung erfüllt ist, wird die betreffende Anweisung (bzw. der Anweisungsblock) ausgeführt. Anderenfalls wird die Anweisung übersprungen.

Python

```
if condition:
    # Anweisungsblock
    # kann aus mehreren Befehlen bestehen
```

- Die einseitige Auswahl beginnt mit dem Schlüsselwort `if`.
- Dahinter wird eine Bedingung (condition) formuliert. Die Bedingung liefert als Ergebnis einen Wert des Typen `bool` zurück.
- Der Anweisungsblock (kann aus einem oder mehreren Befehlen bestehen) ist eingerückt (Tab).
- Der Anweisungsblock wird ausgeführt, wenn die Auswertung der Bedingung den Wert `True` ergibt.
- Der Anweisungsblock wird übersprungen, liefert die Bedingung den Wert `False`.

6.1.2 Zweiseitige Verzweigung (if-else)

Im Vergleich zur einseitigen Auswahl hat die zweiseitige Auswahl am Schluss einen else-Teil. Während die einseitige Auswahl nur entschieden kann, ob ein Anweisungsblock ausgeführt werden soll oder nicht, kann die zweiseitige Auswahl entscheiden, welcher der beiden Teile (if oder else) ausgeführt werden soll.

Python

```
if condition:
    # Anweisungsblock
else:
    # Anweisungsblock
```

- Nach dem Schlüsselwort `if` steht die Bedingung.
- Anschliessend folgt eingerückt der Anweisungsblock, der ausgeführt werden soll, wenn die Bedingung erfüllt ist.
- Das Schlüsselwort `else` befindet sich horizontal auf der gleichen Position wie das Schlüsselwort `if`.
- Anschliessend folgt eingerückt der Anweisungsblock, der ausgeführt werden soll, wenn die Bedingung nicht erfüllt ist.

6.1.3 Mehrstufige Verzweigungen

Eine mehrstufige Auswahl führt mehrere Prüfungen durch, bis eine der Bedingungen True ist. Der zugehörige Anweisungsblock wird ausgeführt, sollten weitere Prüfungen vorhanden sein, werden diese übersprungen.

Python

```
if condition:
    # Anweisungsblock
elif condition2:
    # Anweisungsblock
elif condition3:
    # Anweisungsblock
else:
    # Anweisungsblock
```

- Nach dem Schlüsselwort `if` steht die Bedingung.
- Ist die erste Bedingung `True`, wird der erste Anweisungsblock ausgeführt und die weiteren Bedingungen nicht mehr geprüft.
- Ist die erste Bedingung `False`, wird die zweite Bedingung hinter dem Schlüsselwort `elif` geprüft.
- Ist die zweite Bedingung `True`, wird der zweite Anweisungsblock ausgeführt und die weiteren Bedingungen nicht mehr geprüft.
- Ist die zweite Bedingung `False`, werden nach demselben Muster die weiteren Bedingungen geprüft.
- Sind alle Bedingungen `False`, wird der Anweisungsblock bei `else` ausgeführt.
- Die Verwendung von `else` ist bei dieser Auswahl optional.

Python

```
month = 4

if month == 1:
    print("Januar")
elif month == 2:
    print("Februar")
elif month == 3:
    print("März")
elif month == 4:
    print("April")
```

6.1.4 Auswahlen verschachteln

Eine mehrstufe Auswahl erreichen Sie, indem Sie mehrere if- bzw. if-else-Anweisungen schachteln. Dazu schreiben Sie innerhalb eines Anweisungsblocks einer if- bzw. if-else-Anweisung eine weitere if- bzw. if-else-Anweisung.

Zu welcher if-Anweisung gehört die else-Alternative?

Python

```
if condition:
    if condition2:
        # Anweisungsblock
    else:
        # Anweisungsblock
```

Die else-Anweisung wird ausgeführt, wenn condition2 False ist.

Python

```
if condition:
    if condition2:
        # Anweisungsblock
else:
    # Anweisungsblock
```

Die else-Anweisung wird ausgeführt, wenn die condition False ist.

6.1.5 Vergleichs- und Verknüpfungsmöglichkeiten für Bedingungen

Vergleichsoperatoren

Eine Bedingung (Condition) muss immer auf True oder False auswertbar sein. Um Vergleiche anstellen zu können, existieren verschiedene Möglichkeiten.

Operator	Beschreibung	Beispiel
==	Überprüft zwei Ausdrücke auf Gleichheit	x == y
!=	Überprüft zwei Ausdrücke auf Ungleichheit	x != y
>	Liefert True, wenn der erste Ausdruck grösser als der zweite ist	x > y
<	Liefert True, wenn der erste Ausdruck kleiner als der zweite ist	x < y
>=	Liefert True, wenn der erste Ausdruck grösser oder gleich dem zweiten ist	x >= y
<=	Liefert True, wenn der erste Ausdruck kleiner oder gleich dem zweiten ist	x <= y

Logische Operatoren

Nicht selten gibt es mehrere Bedingungen, welche für eine Bedingung von Relevanz sind. Diese können logisch miteinander verknüpft werden.

Python

```
If x > 10 and x < 20:
    # Anweisungsblock
```

Diese Operatoren sind nur mit boolschen Datentypen verwendbar.

Operator	Beschreibung	Beispiel
and	Das Ergebnis ist nur dann True, wenn beide Ausdrücke True sind	x and y
or	Das Ergebnis ist True, wenn mindestens einer der Ausdrücke True ist	x or y
not	Das Ergebnis liefert das Gegenteil des Ausdrucks	not x

Aufgabe 1 zu Kap. 6	Zeit: 10 Minuten	Arbeitsform: Partnerarbeit

Was gibt das untenstehende Programm aus, wenn der Benutzer die folgenden Werte eingegeben würden?

a. value1 = 6
   value2 = 3
b. value1 = 7
   value2 = 7

Python Code

```
value1 = int(input("Zahl 1: "))
value2 = int(input("Zahl 2: "))

if value2 < value1 and value1 > 5:
        temp = value1
        value1 = value2
        value2 = temp
else:
        if value2 == value2:
                value1 = 5
        value2=8

print("Ausgabe 1 = " + str(value1))
print("Ausgabe 2 = " + str(value2))
```

Resultat:

📝 **Aufgabe 2 zu Kap. 6**	⏳ Zeit: 15 Minuten	Arbeitsform: Alleine 👤 👫

Schreibe ein Python-Programm, das eine Jahreszahl einliest und überprüft, ob es sich um ein Schaltjahr handelt oder nicht. Verwende ausschliesslich Verzweigungen (if / elif / else) – keine Listen, Schleifen oder Funktionen.
Lies dazu eine Jahreszahl vom Benutzer ein. Entscheide mit Hilfe von Bedingungen (if, elif, else), ob das Jahr ein Schaltjahr ist.

Logik (Gregorianischer Kalender):
Ein Jahr ist ein Schaltjahr, wenn es durch 4 teilbar ist, aber nicht durch 100 teilbar ist, ausser es ist durch 400 teilbar. Gib am Schluss eine passende Meldung aus:

- "Das Jahr 2024 ist ein Schaltjahr. "
- "Das Jahr 2023 ist kein Schaltjahr. "

Testfälle zur Überprüfung der Korrektheit:

Test-Nr	Jahr	durch 4 teilbar	durch 100 teilbar	durch 400 teilbar	Erwartung
01	1996	✓	×	×	Schaltjahr
02	1997	×	×	×	Kein Schaltjahr
03	1999	×	×	×	Kein Schaltjahr
04	2000	✓	✓	✓	Schaltjahr
05	1900	✓	✓	×	Kein Schaltjahr
06	2100	✓	✓	×	Kein Schaltjahr
07	2400	✓	✓	✓	Schaltjahr
08	2020	✓	×	×	Schaltjahr
09	2021	×	×	×	Kein Schaltjahr

Erweiterungsoption: Julianischer vs. Gregorianischer Kalender
Der ursprüngliche römische Kalender war ein Mondkalender mit 355 Tagen. Er war viel zu kurz und musste ständig manuell angepasst werden.
Julius Caesar reformierte das System im Jahr 46 v. Chr. Er führte den nach seinem Namen benannten Julianischen Kalender ein. Im julianischen Kalender hat ein Jahr 365 Tage, alle 4 Jahre ist ein Schaltjahr (29. Februar). Schaltjahr-Regel: Jedes Jahr, das durch 4 teilbar ist, ist ein Schaltjahr.
Ein echtes Sonnenjahr dauert etwa 365,2422 Tage, der julianische Kalender war aber mit 365,25 Tagen rund 11 Minuten zu lang pro Jahr. Der Julianische Kalender galt in Europa bis 1582.
Papst **Gregor XIII.** führte in diesem Jahr den ebenfalls nach ihm benannten **Gregorianischen Kalender** ein (der heute weltweit gilt). Darin gilt die Eingangs beschriebene Logik.

Erweitere dein bestehendes Schaltjahr-Programm so, dass der Benutzer zusätzlich angeben kann, welcher Kalender verwendet werden soll:

- **„J"** → Julianischer Kalender
- **„G"** → Gregorianischer Kalender

Eine mögliche Lösung mit einem if-else:

```
1  year = int(input("Bitte Jahreszahl eingeben: "))
2  if(year % 4 == 0) and (year % 100 != 0) or (year % 400 == 0)
3      print ("Das Jahr "+ str(year) + " ist ein Schaltjahr")
4  else:
5      print ("Das Jahr " + str(year) + " ist kein Schaltjahr")
```

#obwohl nicht konsistent wurden beide Möglichkeiten bei der Ausgabe , und + vewendet

Eine zweite mögliche Lösung mit einem verschachteltem if-else

```
year = int(input("Bitte Jahreszahl eingeben: "))
if year % 4 != 0:
    print("Das Jahr " + str(year) + " ist kein Schaltjahr")
else:  # es könnte ein Schaltjahr sein
    if (year % 100 == 0) and (year % 400 != 0):
        print("Das Jahr " + str(year) + " ist kein Schaltjahr")
    else:
        print("Das Jahr " + str(year) + " ist ein Schaltjahr")
```

7 Miniprojekte realisieren

Löst in Zweierteams eine der untenstehenden Problemstellungen.

Team	Mitglied (TIP)	Mitglied (TIA)	Aufgabe
A	Andrist David	Esch Maëlle	Projekt 1 (Wochentag)
B	Eggerschwiler Gabriel	Grob Maxim	Projekt 3 (Josefsrappen)
C	Birrer Damian	Häfliger Lisbeth	Projekt 2 (Osterdatum)
D	Kostadinov Kostadin	Koch Flavio	Projekt 4 (Zugersee)
E	Bucher Simon	Baysal Emre	Projekt 5 (Gasförmig)
F	Kirchgessner Pascal, Saul Adrian	Imfeld Erich	Projekt 2 (Osterdatum)
G	Wüest Valentin	Ott Hendrik	Projekt 1 (Wochentag)
H	Röthlin Janis	Rebmann Flavio	Projekt 5 (Gasförmig)
J	Pavlovic Adrian	Sprenger Jyodisen	Projekt 4 (Zugersee)

7.1 Projekt 01 (Wochentag) [Schwierigkeitsgrad: *]

Projekt Wochentag	Zeit: 30 Minuten	Arbeitsform: Partnerarbeit

Untenstehend liegt eine Anleitung in Pseudocode / Textform vor, anhand der zu einem beliebigen Datum den zugehörigen Wochentag ermittelt werden kann.
Schreiben Sie ein Programm, das vom Benutzer die drei ganzzahligen Werte d (Day), m (Month) und y (Year) entgegennimmt und eine Aussage in Textform macht, um welchen Wochentag es sich handelt.

Logik (Pseudocode):

1. Lass den Benutzer einen Tag, Monat und ein Jahr eingeben und speichere die Werte in den Variablen (d, m, y).
2. Berechne den Wochentag h nach folgender Logik:
   a. Falls m <= 2 ist, erhöhe m um 10 und erniedrige y um 1, andernfalls erniedrige m um 2.
   b. Berechne die ganzzahligen Werte c = y // 100 und y= y Modulo 100 (Modulo -> %)
   c. Berechne den ganzzahligen Wert
      h = (((26 * m – 2) // 10) + d + y + y // 4 + c // 4 – 2 * c) Modulo 7
   d. Falls h kleiner 0 ist, erhöhe h um 7
   e. Anschliessend hat h einen Wert zwischen 0 und 6, wobei die Werte 0, 1, …, 6 den Tagen Sonntag, Montag, …, Samstag entsprechen.
3. Geben Sie das Ergebnis in der Form "Der 24.12.2001 ist ein Montag" aus.

Testfälle zur Überprüfung der Korrektheit:

Test-Nr	Eingabe (Monat)	Erwartung
01	29.10.2025	Mittwoch
02	07.04.1972	Freitag
03	07.04.2037	Dienstag

Weitere Testdaten können mit einem Kalender überprüft werden.

Projekt 02 (Osterdatum): [Schwierigkeitsgrad: **]

Projekt Osterdatum	Zeit: 30 Minuten	Arbeitsform: Partnerarbeit

Untenstehend existiert eine Anleitung in Pseudocode / Textform, anhand jener zu jedem beliebigen Jahr das Datum des Ostersonntags ermittelt werden kann.
Ostern fällt immer auf den Sonntag nach dem ersten Vollmond im Frühling. Es hat also mit den Bewegungen von Erde, Sonne und Mond zu tun.
Der Mathematiker Carl Friedrich Gauss (1777 – 1855) hat dieses Problem mathematisch gelöst. Nachfolgend wird exemplarisch erklärt, wie der Ostersonntag für das Jahr 2019 berechnet werden kann.
Schreiben Sie ein Programm, das vom Benutzer eine Jahreszahl entgegennimmt und für das gewählte Jahr das Osterdatum ausgibt.

Weitere Infos: https://de.wikipedia.org/wiki/Osterparadoxon

Logik (Pseudocode):

1. Zuerst müssen die Zwischenwerte m und n bestimmt werden, die je nach Jahreszahl variieren können. Z.B. ist im Jahr 2019 m=24 und n=5.
   m und n können berechnet werden:

Schritt	Allgemein	Beispiel für Jahr 2019
01	m = (8 * (Jahr / 100) + 13) / 25 – 2	m = (8 * (2019 / 100) + 13) / 25 – 2 = **4**
02	s = Jahr / 100 – Jahr / 400 – 2	s = 2019 / 100 – Jahr / 400 – 2 = **13**
03	m = (15 + s – m) % 30	m = (15 + **13** – **4**) % 30 = **24**
04	n = (6 + s) % 7	n = (6 + **13**) % 7 = **5**

2. **Die Zwischenwerte d und e berechnen:**

Schritt	Allgemein	Beispiel für Jahr 2019
05	a = Jahr % 19 b = Jahr % 4 c = Jahr % 7	a = 2019 % 19 = **5** b = 2019 % 4 = **3** c = 2019 % 7 = **3**
06	d = (19 * a + m) % 30	d = (19 * **5** + **24**) % 30 = **29**
07	Falls d gleich 29 ist, setze d auf 28. Andernfalls: Falls d gleich 28 und a >= 11 ist, setze d auf 27	d = **28**
08	e = (2 * b + 4 * c + 6 * d + n) % 7	e = (2 * **3** + 4 * **3** + 6 * **28** + **5**) % 7 = **2**

3. **Der Tag und der Monat können bestimmt werden:**

Schritt	Allgemein	Beispiel für Jahr 2019
09	Ostertag = (22 + d + e)	Ostertag = (22 + **28** + **2**) = **52**
10	Falls der Ostertag > 31 ist, setze Ostertag = Ostertag % 31 und Ostermonat auf April	Ostertag = **52** % 31 = **21** Ostermonat = **4**

Testfälle zur Überprüfung der Korrektheit:

Testen Sie die Korrektheit Ihrer Applikation mit den folgenden Testwerten (**fett** = Osterparadoxon-Jahr)

1974: 14.April	2006: 16.April	2011: 24. April	2016: 27. März	2021: 04. April
1899: 02.April	2007: 08.April	2012: 08. April	2017: 16. April	2022: 17. April
1900: 15.April	2008: 23.März	2013: 31. März	2018: 01. April	2023: 09. April
1901: 07.April	2009: 12.April	2014: 20. April	**2019:** 21. April	2024: 31. März
2000: 23.April	2010: 04. April	2015: 05. April	2020: 12. April	**2038:** 25. April

7.2 Projekt 03 (Josefsrappen) [Schwierigkeitsgrad: ***]

Projekt Josefsrappen	Zeit: 30 Minuten	Arbeitsform: Partnerarbeit

Die Theorie des Josefsrappen besagt folgendes:
«Geld, das Zinseszinsen trägt, wächst anfangs langsam; da aber durch den Zinseszinseffekt die Rate fortwährend wächst, wird sie nach einiger Zeit unvorstellbar schnell wachsen. Ein Rappen, ausgeliehen bei der Geburt von Jesus im Jahre 0 zu einem Zinssatz von 5%, würde heute zu einer grösseren Summe herangewachsen sein, als 70 Milliarden Erden aus purem Gold entsprächen.»

Um diese und ähnliche Behauptungen zu überprüfen, wird ein Programm erstellt.
Fragen Sie den Benutzer nach der Eingabe eines Start- und Zieljahres, nach einem Betrag und einem Zinssatz. Als Antwort liefert die Applikation den Betrag in CHF und die Entsprechung in Anzahl Erden aus purem Gold.

Recherchen

- Recherchieren Sie das Volumen der Erde anhand des Erdradius. Obwohl die Erde nicht exakt eine Kugel ist, approximieren wir sie als eine Kugel. Die Formel um ein Kugelvolumen zu berechnen wie auch den Erdradius finden Sie im Web.
- Recherchieren Sie, welches Volumen ein Kilogramm Gold hat und was dies kostet.

Tipp:
Mit der Zinseszinsformel wird angenommen, dass die Zinsen jährlich dem Anlagekapital zugeschlagen und dann ihrerseits verzinst werden. Dieser Zins auf Zinsen wird als Jahreszins bezeichnet und in der Zinseszinsformel berücksichtigt:

$K_n = K_0 * ((p / 100) + 1)^n$

K_n: Endkapital inkl. Zinsen nach n Jahren
K_0: angelegtes Anfangskapital
p : Zinssatz in Prozent
n : Anzahl der Jahre

Hintergrundinfos: (Auszug aus Wikipedia):
Das Gedankenexperiment vom Josephspfennig geht zurück auf den britischen Moralphilosophen, Geistlichen und Ökonom Richard Price (1723-1791) und illustriert in der Zinsrechnung das im englischen Sprachraum als miracle of compound interest bekannte Wachstum eines über einen langen Zeitraum angelegten Vermögens durch Zinseszinsen.

7.3 Projekt 04 (Don't jump in Zugersee) [Schwierigkeitsgrad: **]

Projekt Zugersee	Zeit: 30 Minuten	Arbeitsform: Partnerarbeit

Ich hörte einmal eine Aussage, dass die ganze Menschheit im Zugersee Platz habe.

Schreibe ein Programm, das (stark vereinfacht) prüft, ob die gesamte Menschheit volumenmässig in den Zugersee passen würde. Das Programm soll Eingaben dazu verarbeiten und eine verständliche Aussage machen:

Eingaben durch den Benutzer:

- Namen und Volumen des Sees in m^3 (Der Benutzer muss dies recherchieren und eingeben)
- Durchschnittsgewicht eines Menschen in kg (Der Benutzer muss dies recherchieren und eingeben)

Annahmen/Vereinfachungen:

- Ein Mensch besteht zu ca. 70 % aus Wasser → Dichte ≈ 1000 kg/m^3
- Näherung für Personenvolumen: $V_{Person} = \frac{0.70 \cdot Gewicht\ (Kg)}{1000\frac{Kg}{m^3}}\ [m^3]$
- Keine Abstände/Luft/„Packlücken" zwischen Menschen

Das Programm berechnet:
1) Gesamtvolumen aller Menschen
2) Prozentualen Anteil am Seevolumen
3) Aussage, ob das Volumen reicht.

Recherchen

- Recherchiere die aktuelle Weltbevölkerung (wird von dir recherchiert und im Programm fix als Variable gespeichert

Plausibilisierungschecks (nur Verzweigungen):

- Wenn see_vol <= 0 → Hinweis und Programm beenden
- Wenn gewicht <= 0 → Hinweis und beenden

7.4 Projekt 05 (Gefrierpunkt) [Schwierigkeitsgrad: *]

Projekt Gefrierpunkt	Zeit: 30 Minuten	Arbeitsform: Partnerarbeit

Schreibe ein Programm, das eine Temperatur in °C vom Benutzer einliest und eine Meldung ausgibt, ob Wasser, Alkohol oder Benzin bei dieser Temperatur flüssig oder gefroren ist

Recherchen

- Recherchiere die Gefrierpunkte von Wasser, Alkohol (Ethanol).
- Recherchiere, ab welcher Temperatur Benzin *beginnt* zu verdampfen (kein exakter Siedepunkt).

Logik ():

1. Der Benutzer soll die Temperatur in °C eingeben
2. Vergleiche die Temperatur mit den jeweiligen Gefrierpunkten.
3. Gib z. B. aus:

   Bei –70 °C ist:
   Wasser ist gefroren
   Alkohol ist flüssig
   Benzin ist gefroren

Testfälle zur Überprüfung der Korrektheit:

Testen Sie die Korrektheit Ihrer Applikation

Test-Nr	Temperatur	Wasser	Alkohol	Benzin
01	-120 °C	gefroren	gefroren	gefroren
02	-80 °C	gefroren	flüssig	gefroren
03	-20 °C	gefroren	flüssig	flüssig
04	+10 °C	flüssig	flüssig	flüssig
05	+50 °C	flüssig	flüssig	gasförmig
06	+120 °C	gasförmig	gasförmig	gasförmig

Erweiterungsoption:

Wasser wird ab ca. 100 °C gasförmig
Alkohol wird ab ca. 78 °C gasförmig und bei Benzin beginnen Bestandteile bereits ab ca. 30 °C sich zu verdampfen.

Erweitere dein Programm, damit auch Aussagen zum gasförmigen Aggregatszustand erscheinen. Z.B.:

Bei 90 °C:
Wasser ist flüssig
Alkohol ist gasförmig
Benzin ist gasförmig