Zusammen Fassung Block 01 Lösungsalgorithmen

Flavio Rebmann

1 Wichtige Begriffe

1.1 Algorithmus

Was ist ein Algorithmus?
Ein Algorithmus ist eine klare Schritt für Schritt Anleitung, um ein Problem oder eine Aufgabe zu lösen.

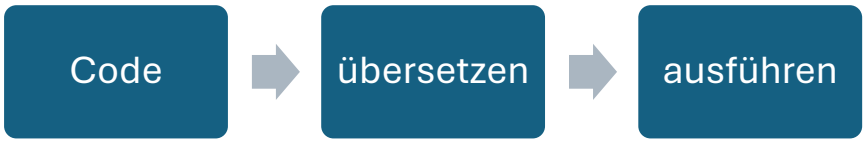
Ein Algorithmus ist immer:

1) Eindeutig: Jeder Schritt ist klar definiert
2) Endlich: Endet nach einer bestimmten Anzahl von Schritten
3) Ausführbar: Jeder Schritt kann tatsächlich ausgeführt werden.
4) Ein und Ausgabe: Beginnt mir bestimmten Daten und liefert ein Ergebnis

1.2 Kompilieren vs Interpretieren

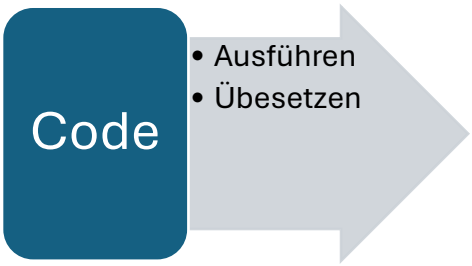
1.2.1 Kompilieren:

Der gesamte Quellcode wird auf einmal komplett in eine Maschinensprache übersetzt



1.2.2 Interpretieren:

Der Code wird Zeile für Zeile während des Ausführens gelesen und umgesetzt.



1.2.3 Vergleich:

	Kompilieren	Interpretieren
Übersetzung	Ganzer Code auf einmal	Zeile für Zeile
Ergebnis	Ausführbare Datei	Sofortige Ausführung
Fehler	Beim Kompilieren	Beim Ausführen
Geschwindigkeit	Schnell	Langsam

1.2.4 Hybride Form JIT-Kompilieren (Just in Time)

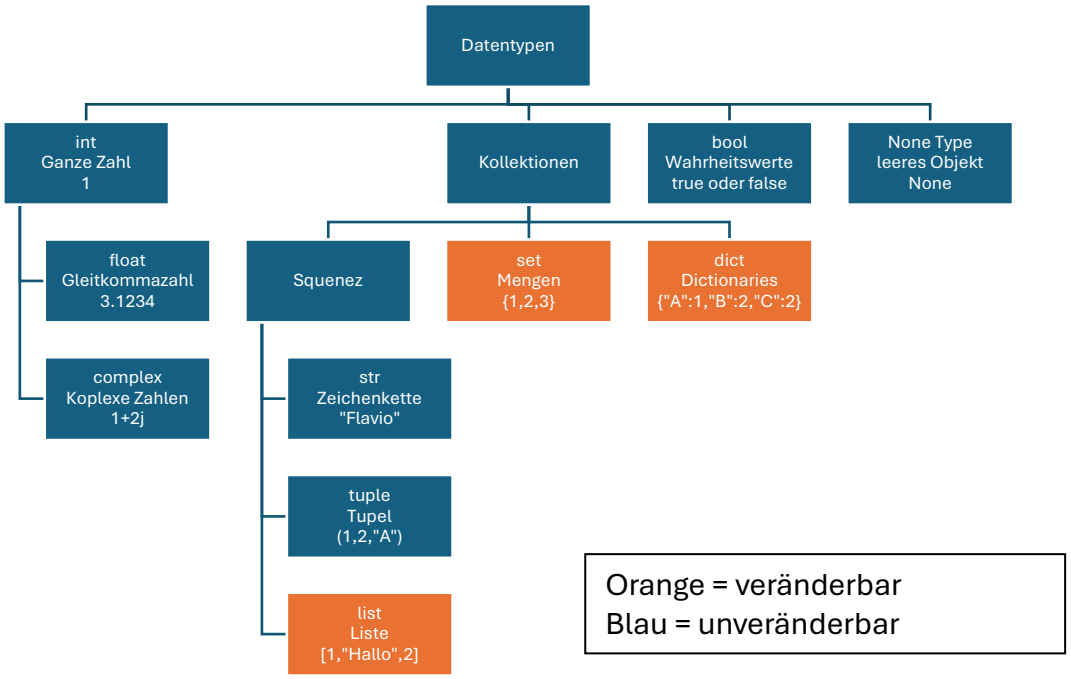
2 Operationen

Operation	Bedeutung	Beispiel
x + y	Addition	5 + 5 = 10
x -y	Subtraktion	10 – 5 = 5
x * y	Multiplikation	5 * 5 = 25
x / y	Division	5 / 3 = 1.666…
x // y	Ganzzahlige Division	5 // 3 = 1
x % y	Modulo (Rest der Division)	5%3 = 2
Abs(x)	Absolutwert	Abs(-5) = 5
Int(x)	Ganze Zahl	Int(5.3) = 5
Float(x)	Komma Zahl	Float(5.3) = 5.3
Pow(x,y)	Potenzieren	Pow(5,5) = 125
xy**	Potenzieren	5 ** 5 = 125

3 Datentypen, Werte und Variable

x = „Flavio“	x ist eine Variable	„Flavio“ ist der Wert	Datentyp String
x = 1	x ist die Variable	1 ist der Wert	Datentyp: integer
x = 3.14	x ist die Variable	3.14 ist der Wert	Datentyp: float
x = True	x ist die Variable	True ist der Wert	Datentyp: bool
x = None	x ist die Variable	x hat keinen Wert	Datentyp: keiner

Weitere Datentypen sind Sammlungen:
Liste, Tuple, Set, Dictionary und Range(Zahlenfolge)



4 Methoden und Operatoren

Alles mit .XXXX() ist eine Methode – alles mit XXXX[] ist ein Operator.

5 Verzweigungen

5.1 IF-ELIF-ELSE

Die IF-Funktion ist eine Verzweigung, welche eine Bedingung überprüft, falls sich diese bewahrheitet, soll eine Anweisung befolgt werden. Ansonsten soll die nächste Bedingung geprüft werden. Falls sich keine Bedingungen bewahrheiten, soll auch eine Anweisung ausgeführt werden.

Die Bedingungen geben immer einen booleschen Wert aus. (True oder Fasle)

If condition (Bedingung)
    Anweisungsblock

Elif condition (Bedingung)
    Anweisungsblock

Elif condition (Bedingung)
    Anweisungsblock

Elif condition (Bedingung)
    Anweisungsblock

Else
    Anweisungsblock

5.2 Vergleichsoperatoren

Operator	Beschreibung	Beispiel
==	Gleicheit	X == Y
!=	Ungleicheit	X != Y
>	Grösser als	X > Y
<	Kleiner als	X< Y
>=	Grösser und gleich als	X >= Y
<=	Kleiner und gleich als	X <= Y

5.3 Logische Operatoren

Operator	Beschreibung	Beispiel
and	Verbindung durch und	X and Y
or	Differenzierung duch oder	X or Y
not	Gegenteil des Ausdrucks	Not Y

6 Schleifen

6.1 While Schleife

Bei der While Schleife wird eine Anweisung solange ausgeführt, bis dich die Bedingung als Wahr behauptet. Die Bedingung wird wie bei if und elif auf boolesche Werte geprüft. While-Schleifen besitzen meistens einen Counter.

```
While condition (Bedingung)
        Anweisungsblock
```

6.1.1 Break

```
Counter = 10
While True:                 <-Endloschschleife
        Counter += 1
        If counter > 10:
                break       <-Durch diesen Befehl wird die Schleife verlassen
```

6.1.2 Continue

```
Counter = 0
While counter < 5:
        Counter += 1
        if counter == 1:
                continue    <-Überspringt die nächsten Anweisungen
        print(„Hallo“)      <-Wird nie ausgeführt, da continue
```

6.2 For Schleife

Für jedes einzelne iterierbare Objekt in einer Sammlung mache die Anweisung.

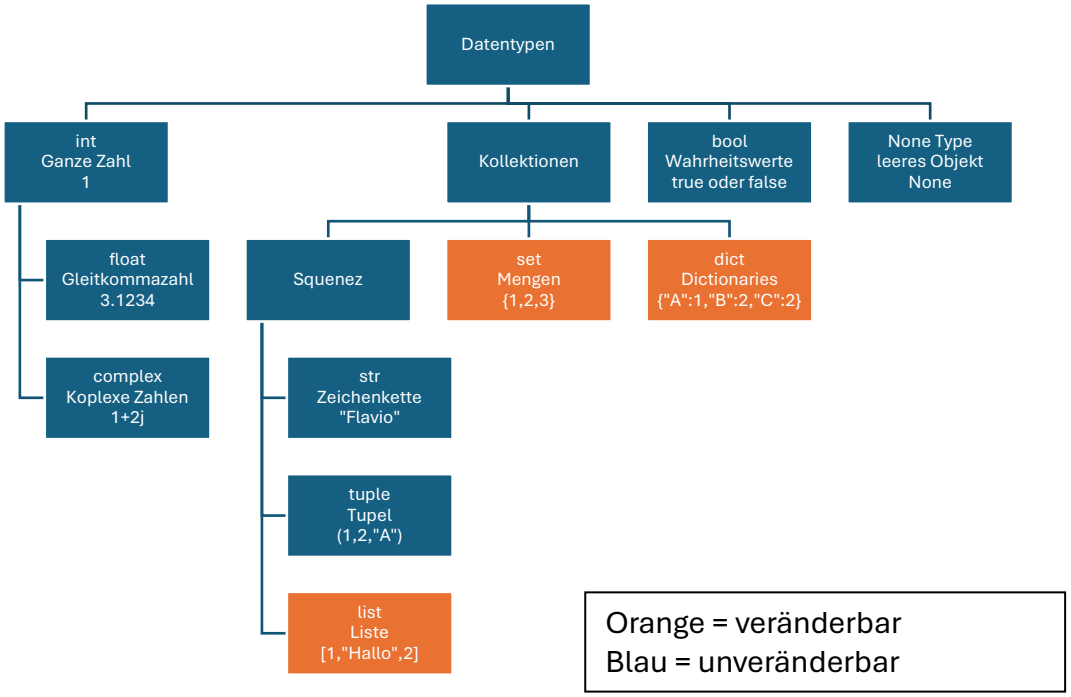
For iterator (Element) in iterierbarem_Objekt (Sammlung)
    Anweisungsblock

Iterierbare Objekte: Range, Liste, Tupel, Set, Dictionary und String

6.2.1 Range

Erklärung	Beispiel	Anzahl Elemente	Werte
Range(stop)	Range(5)	5	0,1,2,3,4
Range(start,stop)	Range(2,8)	6	2,3,4,5,6,7,8
Range(start,stop,steps)	Range(2,16,2)	7	2,4,6,8,10,12,14,16

7 Kollektionen (Sammlungen)



Sequenzen sind geordtnete Sammlungen/Kollektionen auf deren Elemente über den Index zugegriffen werden kann.
Beispiel: string, tuple, liste und range

Mengen sind ungeordnete Sammlungen ohne dopplete Werte
Beispiel: set

Mappings sind Zuordnungen vom Typ mit Key und Value
Beispiel: dictionary

7.1 Tupel

7.1.1 Was ist ein Tuple? Und deren Eigenschaften

Ein Tuple ist eine geordnete Sammlung von Werten, die nicht veränderbar ist.
zahlen = (10, 20, 30)

- Geordnet
- Indexzrugriff möglich
- NICHT veränderbar
- Duplikate erlaubt
- Verschiedene Datentypen

7.1.2 Zugriff auf Elemente via Index

```
zahlen = (10, 20, 30, 40)

zahlen[0]    # 10
zahlen[-1]   # 40
```

7.1.3 Packing und Unpacking

```
x = (1,2)

y, z = x

print(y,z)
#Ausgabe:1 2
```

7.1.4 Verschachtelte Tuple

```
daten = ("Luzern", (47.05, 8.30))
print(daten[1][0])
# 47.05
```

7.1.5 Methoden

```
autos = ("VW", "BMW", "VW", "Audi")

len(autos)           # Anzahl Elemente -> 4
autos.count("VW")    # Wie oft? -> 2
autos.index("VW")  # Position des ersten Element am Fundort -> 0
```

7.1.6 Veränderung

Ein Tuple selbst ist unveränderlich, aber der Inhalt kann veränderlich sein:

```
daten = ("Temperaturen", [12, 15, 14])
daten[1].append(16)   # ✅ erlaubt
```

7.2 Liste

7.2.1 Was ist eine Liste? Und deren Eigenschaften

Eine Liste ist eine geordnete, veränderliche Sammlung von Werten.
zahlen = [10, 20, 30]

- Geordnet
- Indexzugriff möglich
- Veränderlich
- Duplikate erlaubt

7.2.2 Zugriff auf Elemente via Index

```
zahlen[0]    # erstes Element
zahlen[-1]   # letztes Element
```

7.2.3 Veränderungen

```
zahlen[1] = 99         # ändern
zahlen.append(40)     # hinzufügen
zahlen.remove(20)     # löschen
```

7.2.4 Wichtige Methoden

```
zahlen.append(12)   #Fügt 12 am Ende hinzu
zahlen.insert(0,20) #Fügt am Index 0 die Zahl 20 hinzu
zahlen.remove(17)   #Entfernt das erst gefundene 17
zahlen.pop(3)       #Löscht Element am Index, wenn kein Index, dann das letzte Element
zahlen.clear()      #Löscht die ganze Liste
zahlen.count(4)     #Zählt wie oft die Zahl 4 vorkommt
zahlen.index(10)    #Gibt aus an welchem Index die 10 als erstes vorkommt
```

7.2.5 Kopien von Listen

```
a = [1, 2, 3]
b = a          # b referenziert nur auf die Liste a, Methoden von Liste b wirken auf Liste a
c = a.copy()   # Erstellt eine neue echte Kopie von der Liste
```

7.3 Set

7.3.1 Was ist ein set? Und deren Eigenschaften

Ein Set ist eine ungeordnete Sammlung eindeutiger Werte.
a = {1, 2, 3}

- Nicht geordnet
- Kein Indexzugriff möglich zahlen[0] ❌ Fehler
- Sind veränderlich
- Es gibt keine doppelten Elemente

7.3.2 Duplikate

Ein Set löscht alle doppelten Elemente

```
zahlen = {1, 2, 2, 3, 3}
print(zahlen)
#{1, 2, 3}
```

7.3.3 Methoden

```
namen = {"Anna", "Ben"}

namen.add("Clara")        # hinzufügen
namen.remove("Ben")       # entfernen (Fehler, falls nicht da)
namen.discard("Tom")      # entfernen ohne Fehler
```

7.3.4 Set aus Liste erstellen

```
zahlen = [3, 1, 3, 2, 1]
unique = set(zahlen)
```

7.3.5 Leeres Set

```
s = {}      # ❌ Dictionary
s = set()   # ✅ leeres Set
```

7.3.6 Operationen

```
a = {1, 2, 3}
b = {3, 4, 5}

print( a | b ) # Vereinigung    {1, 2, 3, 4, 5}
print( a & b ) # Schnittmenge   {3}
print( a - b ) # Differenz      {1, 2}
```

7.4 Dictionary

7.4.1 Was ist ein Dictionary? Und deren Eigenschaften

Ein Dictionary ist eine Sammlung von Schlüssel (Key)–Wert (Value)-Paaren.
noten = {"Anna": 5.5, "Ben": 4.0}

- Geordnet
- Kein Zugriff über Index möglich, nur über Key
- Veränderlich
- Einzigartige Schlüssel

7.4.2 Zugriff über Keys

```
noten = {"Anna": 5.5, "Ben": 4.0}
noten["Anna"]     # 5.5
```

Wichtig! Fehler wenn der Key nicht existiert!

Zugriff über get() gibt keinen Fehler, nur None falls nicht vorhanden.

```
noten = {"Anna": 5.5, "Ben": 4.0}

noten.get("Clara")        # None
noten.get("Clara", 0)     # 0
```

7.4.3 Werte hinzufügen, ändern und entfernen

```
noten = {"Anna": 5.5, "Ben": 4.0}

noten["Clara"] = 5.0      # hinzufügen
noten["Anna"] = 6.0       # ändern

noten.pop("Ben")          # entfernt & gibt Wert zurück
del noten["Anna"]         # entfernt ohne Rückgabe
noten.clear()             # alles löschen
```

7.4.4 Methoden und iterieren

```
noten = {"Anna": 5.5, "Ben": 4.0}

for key in noten:
    print(key) #Ausgabe Anna Ben

for key, value in noten.items(): #Liefert alle Key und Value Paare
    print(key, value)            #Ausgabe Anna 5.5 Ben 4.0

print(noten.items())    #[('Anna', 5.5), ('Ben', 4.0)]
print(noten.keys())     #['Anna', 'Ben']
print(noten.values())   #[5.5, 4.0]
```

8 Subroutinen

8.1 Warum braucht es Subroutinen?

- Code nicht mehrfach schreiben
- Übersichtlicher
- Code wert und testbar

8.2 Aufbau einer Subroutine

```
def addiere(a, b):
    x = a+b
    return x
```

- def — definierte Soubroutine
- addiere — Name der Subroutine
- a,b — Parameter der Subroutine
- Rumpf — Codeblock
- return — Rückgabewert der Subroutine

8.3 Lokale vs globale Variablen

```python
x = 10 #Globale Variable

def nameFunction():
    y = 5   #Lokale Variable
    global x #Durch global kann die Variable x in der Funktion verändert werden
    ausgabe = 1 + y
    return ausgabe


print(y) #Wird nicht ausgegebn, da y eine lokale Variable
print(nameFunction()) #6
```

8.4 Subroutine ohne Rückgabewert

```python
def hallofunction():
    x = 5 + 5
    print(x)

#Liefert eine Ausgabe im Terminal aber kein Rückgabewert der weiterverwendet werden könnte
```

8.5 Subroutinen mit optionalen Paramter

```
def optionaleParameter(a,b=99):
    return a+b

print(optionaleParameter(900)) #999
print(optionaleParameter(1,5)) #6
```

Wenn kein Parameter mitgegeben wird wird der Optionale verwendent.
Wichtig! Der optionale Parameter muss immer am Ende stehen.

8.6 Subroutine mit variablen Paramater

```
def variableParameter(a,*b):
    print(a,b)

print(variableParameter(1,2,3,4,5)) #1 (2, 3, 4, 5)
```

8.7 Anonyme Funktionen

Eine **anonyme Funktion** ist eine **Funktion ohne Namen**, die man **direkt verwendet**, ohne sie mit def zu definieren. Alles in einer Zeile!

lambda parameter: code

```
f = lambda x: x * 2
print(f(4)) #8
```

8.8 Funktion in Funktionen

```
def aussen():
    def innen():
        print("Hallo")
    innen()

aussen()
innen() #Wird hier nicht ausgeführt, da sie in der äussern Funktion ist
```

Auf äussere Variablen zugreifen:

```
def aussen():
    x = 10
    def innen():
        print(x) #Innere Funktion kann nur die äusseren Variablen lesen aber nicht verändern
    innen()
```

Durch nonlokal kann eine innere Funktion auf eine äussere variable zugreifen.