Einführung in die Programmierung

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- http://www.miwilhelm.de
- Raum 2.202
- Tel. 03943 / 659 338

Inhalt

- Einführung
- **Grundlagen**
  - Datentypen
  - Variablen
  - Zuweisungen
  - Operatoren
  - Verzweigungen und Schleifen
  - Funktionen
  - Listen / Tupel / Mengen
- struct / Objekte
- User Interface mit TKinter

In dieser Vorlesung lernen wir vor allem:

- Einführung in die Programmierung
- Anhand der Programmiersprache Python
- Entwerfen von einfachen Algorithmen
- Einfache Datenstrukturen

  (Darstellungsmöglichkeiten für Daten)
- Grundlagen der Objektorientierung

Meist unterscheidet man drei Säulen:

- Theoretische Informatik
- Praktische Informatik
- Technische Informatik

Sowie angewandte Informatik, z.B. Bio- und Medieninformatik.

Zur praktischen Informatik gehören Gebiete wie:

- Programmiersprachen
- Software-Technik
- Software-Engineering
- Datenbanksysteme
- App-Entwicklung
- Mess-Erfassungssysteme

Algorithmen

Abgeleitet von Muhammad ibn Musa al-Chwarizmi
(lat. al-Gorithmus), persischer Mathematiker 9. Jh. n. Chr.

- Algorithmus ist eine **Handlungsvorschrift**, um ausgehend von bestimmten **Vorbedingungen** ein bestimmtes **Ziel** zu erreichen
- Es ist ein zentraler Begriff der Informatik

Wichtige Inhalte des Informatikstudiums:

- Entwicklung von Algorithmen
- Analyse von Algorithmen (Korrektheit, Laufzeit, Eigenschaften)
- Oft weniger wichtig: Umsetzung in Programmiersprachen

Typische Elemente, die in Algorithmen enthalten sind:

- Schrittweises Vorgehen
- Fallunterscheidungen und Wiederholungen („Schleifen“)
- Zur Berechnung erforderliche Informationen in „Variablen“
- Eingabe: Parameter der Problemstellung
  - z.B. das Wort nach dem im Web gesucht wird
- Ausgabe: Lösung des Problems
- Liste der Treffer-Websites geordnet nach statist. Relevanz

Interaktive Programme (Word/Excel) sind selbst keine Algorithmen, setzen aber zur Lösung von Teilproblemen (z.B. Sortieren, Rechtsschreibung) Programmteile ein, die auf Algorithmen beruhen

Beispiele aus alltäglichem Leben

- Montageanleitungen
  - Ikea
  - neues Smartphone einrich ten
- Reifenwechsel
- Kochrezept (mit Einschränkung)
- Lösen mathematischer Aufgaben
  - Hauptnenner bilden
  - Gleichungen lösen
  - Ableitungen bilden
  - Integrale berechnen
- Puzzle
- Spiele (Alle abschießen bis Ziel erreicht)

Ein Algorithmusist ein Verfahren zur Verarbeitung von Daten mit einer präzisen, endlichen Beschreibung unter Verwendung effektiver, elementarer Verarbeitungsschritte:

- Daten:
  - Die Repräsentation und der Wertebereich von Eingabe und Ergebnissen müssen eindeutig definiert sein.
- Präzise, endliche Beschreibung:
  - Die Abfolge von Schritten muss in einem endlichen Text in einer eindeutigen Sprache genau festgelegt sein.
- Effektiver Verarbeitungsschritt:
  - Jeder einzelne Schritt ist tatsächlich ausführbar.
- Elementarer Verarbeitungsschritt:
  - Jeder Schritt ist entweder eine Basisoperation, oder ist selbst durch einen Algorithmus spezifiziert.

GGT-Algorithmus von Euklid

Algorithmus zur Berechnung des größten gemeinsamen Teilers (GGT) von zwei natürlichen Zahlen a und b.

Beschrieben von Euklid von Alexandria(300 v.Chr.) im Werk „Die Elemente“.

EUKLID‘scher Algorithmus

Eingabe: a, b ∈ ℵ

1. solange b≠ 0
2. wenn a> b
3. dann a:= a−b
4. sonst b:= b −a

Ausgabe: a

Es ergeben sich folgende interessante Fragen:

- Ist der Algorithmus korrekt?
  - Also: berechnet er wirklich den GGT aus (a,b)?
  - Wieso funktioniert dieser Algorithmus überhaupt?
- Ist der Algorithmus vollständig?
  - Kann ich wirklich jedes Paar (a,b) eingeben?
  - Negative Zahlen?
- Ist der Algorithmus terminierend (evtl. eine Endlosschleife)?
- Ist der Algorithmus effizient?
  - Wie viel Speicher benötigt er?
  - Wie viel Zeit (d.h. Verarbeitungsschritte) benötigt er?
  - Wie hängt das von der Eingabe (a,b) ab?

Wir müssen solche Aussagen immer mathematisch beweisen.

Hauptnenner bilden

Algorithmus zur Berechnung des Hauptnenner zweier Brüche

Beispiel: $\frac{2x}{x+1} + \frac{3x}{3x} = \frac{a+b}{c} = \frac{6x^3+6x^2 + 9x^3+9x^2}{3x^2+3x}$

Aufgabe: Gesucht die Terme a,b,c

Beispiellösung / Algorithmus:

1. Multpliziere die beiden, alle, Nenner:
2. c = (x+1)·3x = **$3x^2+3x$**
3. Multpliziere 2x mit dem Hauptnenner
4. a = 2x·(**$3x^2+3x$**) = $6x^3 + 6x^2$
5. Multpliziere 3x mit dem Hauptnenner
6. b = 3x·(**$3x^2+3x$**) = $9x^3 + 9x^2$

Hauptnenner bilden

Algorithmus zur Berechnung des Hauptnenner zweier Brüche

Beispiel: $\frac{2x}{3x+6} + \frac{3x}{3x} = \frac{a+b}{c} = \frac{18x^3+36x^2 + 27x^3+54x^2}{9x^2+18x}$

Aufgabe: Gesucht die Terme a,b,c

Beispiellösung / Algorithmus:

1. Multpliziere die beiden, alle, Nenner:
2. c = (3x+6)·3x = **$9x^2+18x$**
3. Multpliziere 2x mit dem Hauptnenner
4. a = 2x·($9x^2$+18x) = $18x^3 + 36x^2$
5. Multpliziere 3x mit dem Hauptnenner
6. b = 3x·($9x^2$+18x) = $27x^3 + 54x^2$

Projektannahme

Zentrale Aufgabe eines Informatikers:

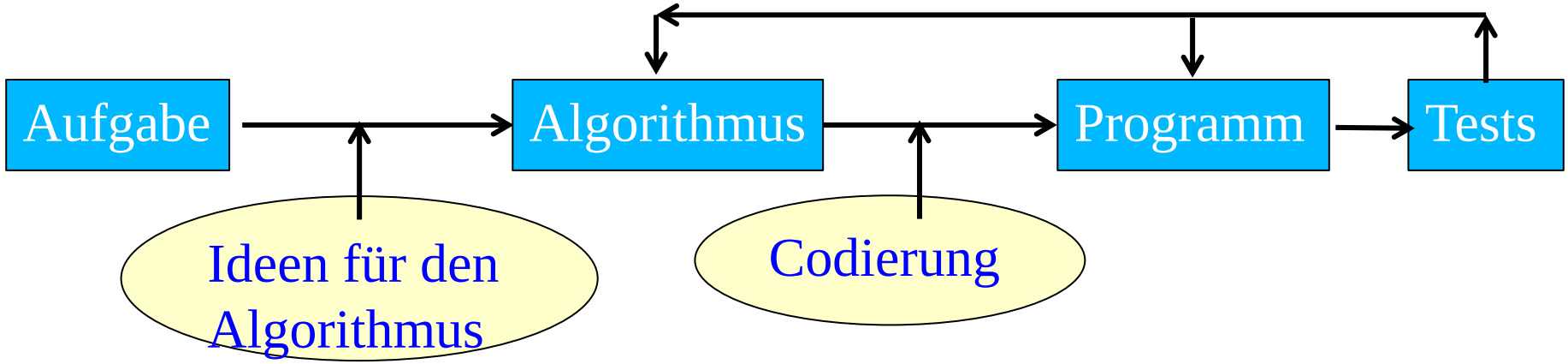
- Die Entwicklung von Algorithmen und deren Realisierung auf einem Rechner als Programm

Programm:

- Ein Programm ist die formale Darstellung eines Algorithmus in einer Programmiersprache (Vorstufe PAP, Nassi-Schneiderman-Diagramm)

Programmiersprache:

- Formale, eindeutige Sprache. Sie stellt elementare Verarbeitungs-schritte und eindeutig definierte Datentypen für die Ein-, Ausgabe und Bearbeitung zur Verfügung.



Ablauf:

- Problemanalyse
- Plattform / Typ:
  - PC (Windows, Linux, OS X)
  - Apps (Android,iOS)
  - Server?
- Algorithmus
- Codierung
- Debugging / Testen
- Programm ist fertig

Programmiersprachenbefehle (imperativ, befehlsbezogen)

- Primitive Datentypen (Integer, Double, String, Boolean)
- Fallunterscheidungen
  - if, then, else
  - switch case
- Schleifen
  - while
  - repeat until
  - for(range)
  - for each
- Funktionen (à la sinus)
- Objectorientierte Programmierung
  - Ableitung
  - Polymorphismus
  - Abstrakte Klasse
  - Interface
- Rekursion

11

-23

3

88

22

144

42

22 +23 13

11 88

42 144

+23

13

22

11

88

42

1126

154

Programmiererfahrung ?

- Assembler
- Basic
- C++
- C#
- Delphi (Turbo Pascal)
- HTML / CSS / Java-Script
- Java
- PHP
- Haskell
- Lisp
- Python
- Prolog

Kapitel: Einführung in Python

- Variablen
  - Grundtypen
  - Strings
- Bedingungen
  - if-Anweisungen
  - else-Anweisungen
  - geschachtelte Anweisungen
- Schleifen
  - while
  - for
  - for-each

Variablen

- Eigenschaften von Variablen:
  - Eine Variable hat einen **Namen**
  - Eine Variable hat einen **Inhalt**, einen Wert

  - Eine Variable hat einen bestimmten Typ
  - Der Datentyp hat einen **physikalischen** Bereich (Min,Max)
  - Die Variable hat einen **logischen** Bereich (Min,Max)

  - Eine Variable hat einen bestimmten **Gültigkeitsbereich**
  - Dieser Gültigkeitsbereich wird durch einen Block bestimmt.
  - Man kann den Inhalt, den Wert, der Variablen holen
  - Man kann den Inhalt, den Wert, der Variablen setzen

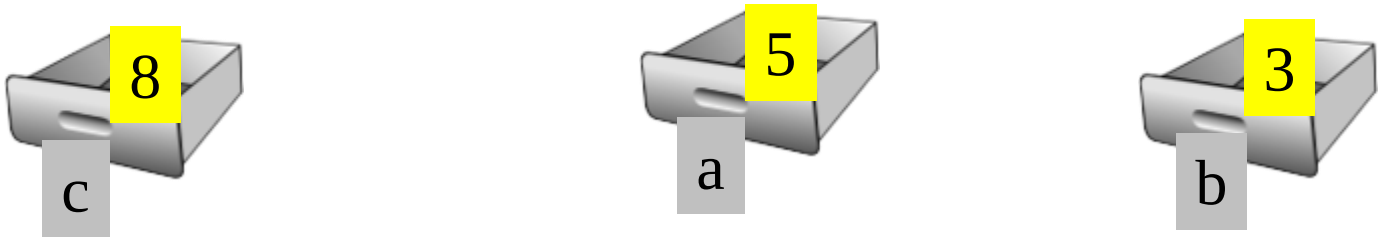
Variable vs. Schublade

Schublade "a"

- **Eigenschaften einer Schublade:**
  - Eine Schublade hat einen Namen (Zettel klebt vorne)
  - In eine Schublade kann man etwas reinlegen (Schublade auf, Gegenstand in die Schublade legen)
  - Eine Schublade hat einen bestimmten Gültigkeitsbereich. Man muss in das Zimmer gehen, in dem der Schrank steht.
  - Ist man in einem anderen Zimmer, kann man diese Schublade nicht öffnen, also auf die Variable nicht zugreifen
  - In einer Schublade kann man unterschiedliche Sachen reinlegen.
    - 5, 7, 1222 oder 55555
  - Man kann den Inhalt sich anschauen: Schublade auf, Gegenstand anschauen

Variable vs. Schublade

- Gegeben zwei Schubladen
  - Schublade "a" hat den Inhalt "5"
  - Schublade "b" hat den Inhalt "3"
- Programmzeilen
  - a, b, c; // irgendwie deklarieren, also Schubladen kaufen
  - a = 5; // Die leere Schublade wird mit der Zahl 5 gefüllt
  - b = 3;
  - c = a + b;



Namensgebung von Variablen

- Variablen müssen bezüglich des gewählten Namens eindeutig sein.
- Grundsätzlich gelten die folgenden Konventionen:
  - Als Zeichen sind Groß- und Kleinbuchstaben, der Unterstrich und die Ziffern 0 bis 9 zulässig.
  - Jeder Bezeichner muss mit einem **Buchstaben** oder einem **Unterstrich** beginnen.
  - Python unterscheidet zwischen **Groß- und Kleinschreibung** (case-sensitive Programmiersprache).
  - Die Bezeichner der Variablen sollten mit einem **Kleinbuchstaben** beginnen.
  - Unterstriche sind zu vermeiden.
  - Falls man Bezeichner aus mehreren Wörtern zusammensetzt, sollte man ab dem zweiten Wort alle Wörter mit einem Großbuchstaben starten (z. B.: **minimumZahl, maxSeitenzahl**).
  - **Die Variablen müssen nicht vorher deklariert werden.**

Beispielnamen von Variablen

- anzahl
- hoeheTisch
- nachName

- # Deklaration mit Initialisierung
- anzahl = 99
- hoeheTisch = 1.34
- hoeheTisch2 = hoeheTisch +1.34
- nachName = "Maier"
- **Wertzuweisung :=**
  - **von rechts nach links**

Reservierte Wörter:

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

https://www.delftstack.com/de/tutorial/python-3-basic-tutorial/keywords-and-identifiers/

Operatoren

Name	Operator	Beispiel	
Addition	+	x+y	
Subtraktion	–	x-y	
Multiplikation	*	x*y	
Division	/	x/y	
Modulo/Rest	%	x%y	
Addition	+	+=	Zuweisungsop.
Subtraktion	–	–=	
Multiplikation	*	*=	
Division	/	/=	
Modulo	%	%=	
Direkte Zuweisung	=		

Vergleichsoperatoren

Name	Operator	Beispiel
Gleicheit	==	x==y
Kleiner	<	x<y
Kleiner Gleich	<=	x<=y
Größer	>	x>y
Größer Gleich	>=	x>=y
Ungleich	!=	x!=y
Und-Bedingung	and	x!=y and x>0
Oder-Bedingung	or	x!=y or x>0

Priorität

Damit ergibt sich folgende Rangfolge der Operatoren:

- Klammern: ()
- Logisches NOT: !
- Multiplikation, Division, Modulo: *, /, %
- Addition und Subtraktion: +, –

- Kleiner als, kleiner gleich als, größer als und größer gleich als: <, <=, >, >=
- gleich, ungleich: ==, !=

Primitive Datentypen (Zahlen und Bool)

- Integer (Ganzzahlig)
  - n = 123
  - monatsgehalt = 792281625142643375935439503336
  - oktalsystem = 0o177
  - hexadecimalsytem = 0x20
  - dualSytem = 0b1010
  - dualSytem = 0b1_0101_1000_1010

- Real (Nachkommazahlen)
  - x = 1.5
  - y = 2*x + 4; # **Wertetabelle**
  - brutto = netto*1.19
  - greatNumber = 1.234e100
  - smallNumber = 1.234E-100
  - pi = 3.141_592_653

Primitive Datentypen (Bool)

- Bool
  - ungerade = True
  - gerade = False
  - isGerade = gerade == True
    - # wenn gerade wahr ist , erhält isGerade den Wert True
    - Sonst den Wert False

Primitive Datentypen (String)

- String (Zeichenketten)
  - nachname = "Meier"
  - vorname = "Kevin"
  - ort = "Hamburg"
  - nachname = 'Meier'
  - vorname = 'Kevin'
  - ort = 'Hamburg'
  - name = nachname + ", " + vorname
  - matrnr = 12345
  - note = 1.3
  - pruefung = **str**(matrnr) + ', ' + **str** (note)

Abfragen

- **if**
  - if note<=2.0:
  - →bem = "Sehr gute Klausur geschrieben"
- **if / else**
  - if note<=2.0:
  - →bem = "Sehr gute Klausur geschrieben"
  - else:
  - →bem = "Nächstes mal wird’s besser"

- **If / else if / else**
  - if note<=2.0:
  - →bem = "Sehr gute Klausur geschrieben"
  - elif note<=3.0:
  - →bem = "Gute Klausur geschrieben"
  - else:
  - →bem = "Nächstes mal wird’s besser"

While-Schleife

```
d:int = 0
e:int = 10
r:int = 2

while d<=e:
    d++
    print(d)
    if d>r:
        break
else:
    print("Die Schleife wurde nicht mit break abgebrochen")

print("nach der Schleife")
```

Foreach-Schleife

```
for c in "Einmal...":
    print("c")


for c in "Einmal...":
    print(c)
else:
    print("Die Schleife wurde nicht mit break abgebrochen")
```

Numerische For-Schleife

Range-Varianten:

• range(n)	i=0	i<n	i++
• range(start,n)	i=start	i<n	i++
• range(start,n,step)	i=start	i<n	i+=step

```
for i in range(1,10,2)
    print(i)
else:
    print("Die Schleife wurde nicht mit break abgebrochen")
```

Ausgabe:

```
0
-2
-4
-6
-8
```

Numerische For-Schleife

Range-Varianten:

• range(n)	i=0	i<n	i++
• range(start,n)	i=start	i<n	i++
• range(start,n,step)	i=start	i<n	i+=step

```
for i in range(1,9,2)
    print(i)
else:
    print("Die Schleife wurde nicht mit break abgebrochen")
```

Ausgabe:

```
0
-2
-4
-6
-8
```

Python-Funktionen

round(x[,n])		
round(12.5)	12	Rundet auf n-Nachkommastellen
math.ceil(x)	Rundet auf die nächsthöhere Ganzzahl	
math.ceil(x)		
math.ceil(12.2)	13	
math.ceil(12.5)	13	
math.ceil(12.8)	13	
math.ceil(-12.2)	-12	
math.ceil(-12.5)	-12	
math.ceil(-12.8)	-12	Rundet auf die nächsthöhere Ganzzahl
math.floor(x)		
math.floor(12.2)	12	
math.floor(12.5)	12	
math.floor(12.8)	12	
math.floor(-12.2)	-13	
math.floor(-12.5)	-13	
math.floor(-12.8)	-13	Rundet auf die nächstniedrigere Ganzzahl

Python-Funktionen

isnan(x)	Gibt True zurück, wenn x nan ist.
ldexp(m,e)	Bestimmt Zahl aus Mantisse und Exponent
math.trunc(x)	Gibt den Vorkommaanteil zurück (à floor) Der Unterschied liegt in den negativen Zahlen.
math.trunc(12.5)	12
math.trunc(12.2)	12
math.trunc(12.8)	12
math.trunc(-12.2)	-12
math.trunc(-12.5)	-12
math.trunc(-12.8)	-12

Python-Funktionen

abs(x)	**Absolutwert bei Integer-Zahlen**
chr(i) **chr(65)** **'A'**	Ausgabe des i-Zeichens (Ascii-Code)
complex([real, [,imag]])	Erzeugt eine komplexe Zahl
float([x])	Erzeugt eine Gleitkommazahl
hex(x) **hex(241)** **0xF1** **hex(241)[2:]** **F1**	Gibt den Hexadezimalwert der ganzen Zahl x in Form eines Strings zurück
min **min([1,2,3]** **1**	Minimum
max **max([1,2,3]** **3**	Maximum
oct(x) **oct(16)** **'0o20'**	Gibt den Oktalwert der ganzen Zahl x in Form eines zurück
ord(c) **ord('a')** **97**	Gibt den Unicodewert des Zeichen c aus.
pow(x[,y[,y]]) **pow(2,3)** **8**	Potenzfunktion
round(x[,n]) **round(12.5)** **12**	Rundet auf n-Nachkommastellen
sum **sum([1,2,3])** **6**	Summierung
print(?)	Ausgabe von String und Variablen

Python-Funktionen

math.exp(x) **math.exp(1)** **2.718281828459045**	**e^x**
math.expm1(x) **math.expm1(1)** **1.718281828459045**	e^x-1. Ist genauer als exp(x)-1.
math.log(x[,base]) **math.log(100,10)** **2.0**	Logarithmus mit Basis
math.log(x) **math.log(11)** **2.3978952727983707**	Logarithmus Naturalis, zur Basis e
math.log10(x) **math.log10(12)** **1.0791812460476249**	Dekadischer Logarithmus
math.log1p(x) **math.log1p(2)** **1.0986122886681098**	Berechnet ln(1+x). Ist genauer als log(x)+1.
math.pow(x,y) **math.pow(2,3)** **8.0**	x^y
math.sqrt(x) **math.sqrt(2)** **1.4142135623730951**	Quadratwurzel

Python-Funktionen

double acos(double x)	**arccos(x) im Bereich [0, p], x ∈ [-1, 1]**
double asin(double x)	arcsin(x) im Bereich [-p/2, p/2], x ∈ [-1, 1]
double atan(double x)	arctan(x) im Bereich [-p/2, p/2]
double atan2(double y, double x)	arctan(y/x) im Bereich [-p, p]
double cos(double x)	Kosinus von x
hypot(x)	Euklidsche Norm
sin(x)	sin(x)
tan(x)	Tangens von x
degrees(x)	Bogenmaß in Grad: *180/π
radians(x)	Grad in Bogenmaß in Grad: *π/180
acosh(x)	ArcCosinus Hyperbolicus von x
asinh(x)	ArcSinus Hyperbolicus von x
atanh(x)	ArcTangens Hyperbolicus von x
cosh(x)	Cosinus Hyperbolicus von x
sinh(x)	Sinus Hyperbolicus von x
tanh(x)	Tangens Hyperbolicus von x

Python-Funktionen

```
import time

now = time.localtime()
print("Tag:", now.tm_mday)
print("Monat:", now.tm_mon)
print("Jahr:", now.tm_year)
print("Stunde:", now.tm_hour)
print("Minute:", now.tm_min)
print("Sekunde:", now.tm_sec)
print("Wochentag:", now.tm_wday)  # Montag = 0
print("Tag des Jahres:", now.tm_yday)
print("Sommerzeit:", now.tm_isdst)  # Sommerzeit: 1; Winterzeit: 0
```

Eigene Funktionen

Es wird nicht definiert, ob eine Funktion einen Rückgabewert hat.

Syntax:

```
def  Funktionsname(Parameter1, Parameter2):
     Anweisung1
     Anweisung2

def  Funktionsname(Parameter1, Parameter2):
     Anweisung1
     Anweisung2
     return x                                  oder None
```

Eigene Funktionen

Beispiele:

```
def  summe1BisN(n):
     summe = 0
     for i in range(1,n+1):       # Ende zählt nicht mit
          summe += i
     return summe
```

```
def  fakultaet(n):
     ergebnis = 1
     for i in range(2,n+1):       # Ende zählt nicht mit
          ergebnis *= i
     return ergebnis
```

Eigene Funktionen: Optionale Parameter

```
def  Funktionsname(Parameter1, Parameter2=2):
     Anweisung1
     Anweisung2
```

```
def  PrintNumbers(n, step=1):
     for i in range(1,n+1,step):      # Ende zählt nicht mit
          print(i)
```

Eigene Funktionen: Beliebige Parameter

```
def  Funktionsname(Parameter1, *weitereParameter):
     print(weitereParameter)   # hier Ausgabe als Liste
```

```
def  summeVonZahlen(init, *steps):
     summe=init
     for item in steps:        # Ende zählt nicht mit
          summe += item
     return summe
```

Listen

- **Merkmale**
  - dynamische Liste (Regal)
  - Speichert beliebige Datentypen
  - Änderbar
- **Erzeugen einer Liste**
  - liste = []
- **Methoden**
  - s.append(x)
  - s.extend(t) — t-Liste an s-Liste
  - s.insert(i,x) — Einfügen von x an i. ter Stelle
  - s.pop() — Letzte Element + löschen
  - s.pop(i) — Ausgabe des i. ten Element + löschen
  - s.remove(x) — x Object
  - s.reverse() — Umdrehen der Liste
  - s.sort() — Sortieren
  - s.sort(key) — s.sort(key=len) nach Länge
  - s.sort(key, reverse)

Listen

- **Operationen mit einer Liste**
  - s[i] = t — Ersetzt das i-te Element
  - s[i:j] = t — Ersetzt die i-te bis j.te Elemente
  - s[i:j:k] =t — Setzen mit von i bis j Schrittweite k
  - del s[i] — Löscht das i-te Element
  - del s[i:j] — Löscht die i-te bis j.te Elemente
  - del s[i:j:k] — Löscht die i-te bis j.te Elemente (Schrittweite)
  - del liste[0:len(liste)] — Löschen aller Elemente in einer Liste:

Beispiele für Listen

```
a = [1337]
b = a                    # hier Referenz
c = a[:]                 # Kopie
b += [2674]
b.append(2675)
b                        # Ausgabe [1337, 2674, 2675]
a                        # Ausgabe [1337, 2674 , 2675]
c                        # Ausgabe [1337]
a is b                   # true

a=[1,2,3,4,5,6,7,8,9]
a[::-1]                  [9, 8, 7, 6, 5, 4, 3, 2, 1]
a[::+1]                  [1, 2, 3, 4, 5, 6, 7, 8, 9]
a[::+2]                  [1, 3, 5, 7, 9]
```

Tupels

- **Ein Tupel ist eine Sequenz, ein Array, mit unterschiedlichen Datentypen und beliebiger Länge.**

- **Eigenschaften**
  - dynamische Liste
  - Speichert beliebige Datentypen
  - **Nicht änderbar**

- **Erzeugen eines Tupels**
  - meinTupel1 = ()
  - meinTupel2 = (123, 1234.567, "Ein Elemente in einem Tupel")

- **Beispiele**
  - a = (1,2,3,4)
  - a[3] # Ausgabe 4
  - b = (1,2) # okay
  - c = (2) # nicht okay, ist nur die Zahl
  - d = (2,) # okay
  - print(len(a)) # 4
  - print(a[0]) # 1

Mengen (Set)

- **Eine Menge ist eine „Array“, indem keine doppelten Elemente vorkommen können.**

- **Unterscheidung zwischen unveränderlichen und veränderlichen Mengen:**
  - set() leere veränderliche Menge
  - frozenset() leere unveränderliche Menge
  - ofs = frozenset(True, 123, "Einmal…")
  - ofs = { True, 123, "Einmal…"}

- **Durchlaufen der Menge**
  - s = set((2,3,5,7)) # Der Parameter ist ein Tupel !
  - for item in s:
  - print(item)

Mengen (Set)

- **Methoden von Mengen**
  - len(menge)
  - x in menge
  - x not in menge
  - menge1 <= menge2
  - menge1 < menge2
  - menge1 >= menge2
  - menge1 > menge2
  - menge1 / menge2 # Vereinigungsmenge
  - menge1 & menge2 # Schnittmenge
  - menge1 - menge2 # Differenzmenge
  - menge1 ^ menge2 # symmetrische Differenzmenge, alle die nur in einer Menge vorhanden sind

Mengen (Set)

- **Methoden für veränderliche Mengen**
  - m1.add(e)
  - m1
  - m1.clear()
  - m1.difference_update(m2) # Äquivalent zu m1 -= m2
  - m1.discard(e) # Löscht das Element
  - m1.intersection_update(m2) # Äquivalent zu m1 &= m2
  - m1.remove(e) # Löscht das Element ev. Exception
  - m1.symmetrisch_update(m2) # Äquivalent zu m1 ^= m2
  - m1.update(m2) # Äquivalent zu m1 /= m2