

# Einführung in die Programmierung

Ronja Düffel  
WS2015/16

05. Oktober 2015

# Rückblick

- Datentypen
  - `bool`
  - Zahlen (`int` und `float`)
  - `string`
- Variablen
- Kontrollstrukturen
  - Verzweigungen (`if...:` und `if...else:`)
  - Schleifen (`while...:` und `for...:`)

# aufsteigend sortierte Zahlenfolge

- überprüfen, ob Zahlenfolge aufsteigend sortiert ist ✓
- Zahlenfolge sortieren ???
- brauchen dynamische Anzahl an Variablen

## Listen

# Listen

- mehrere Werte unter einem Namen zusammengefasst
- Länge der Liste ist nicht festgelegt (dynamisch)
- auf einzelne Werte kann zugegriffen werden
- **Achtung!** In der Informatik fängt man bei 0 an zu zählen!  
<liste>[0] liefert das erste Element der Liste

# Listen

Operator/ Funktion	Beschreibung
<code>&lt;list&gt;[x]</code>	Zugriff auf Element mit Index $x$
<code>&lt;list&gt;[x:y]</code>	Zugriff auf Teilliste von Index $x$ bis $y$
<code>&lt;list&gt; + &lt;list&gt;</code>	zusammenfügen von Listen
<code>&lt;list&gt;.append(x)</code>	hinzufügen von $x$
<code>del &lt;list&gt;[x]</code>	löschen von Element mit Index $x$
<code>&lt;list&gt;.remove(x)</code>	löschen von Element $x$
<code>len(&lt;list&gt;)</code>	Länge der Liste

# Funktionen

für Operationen die immer wieder gebraucht werden

- + Wiederverwertbarkeit
- + leichte Wartbarkeit
- + nur einmal schreiben
- + leicht auszutauschen
- + Übersichtlichkeit

# Funktionen

File Edit Format Run Options Windows Help

```
#Eine Funktion
```

```
def add(a,b):
```

```
    '''Addiere die Zahlen a und b'''
```

```
    return a+b
```

```
|
```

# Gültigkeitsbereiche

File Edit Debug Options Windows Help

```
>>> def sowas():  
    x = 0  
    print('x = 0')  
    return
```

```
>>> x = 5  
>>> sowas()  
x = 0  
>>> 5/x
```



# Gültigkeitsbereiche

File Edit Debug Options Windows Help

```
>>> def sowas():  
    x = 0  
    print('x = 0')  
    return
```

```
>>> x = 5  
>>> sowas()  
x = 0  
>>> 5/x  
1.0  
>>> |
```

# Gültigkeitsbereiche

- Variablenname ist in dem Anweisungsblock gültig, in dem er definiert wird.
- unterscheide zwischen *lokalen* (innerhalb Block/Funktion) und *globalen* (auch außerhalb) Variablen
- Verwendung globaler Variablen innerhalb von Funktionen mit `global`

## Wiederverwendung von Funktionen in anderen Programmen :

- `import`
  - `import <Modulname>` (Dateiname ohne `.py`)  
Verwendung durch `<Modulname>.<Funktionsname>`  
(kein Namenskonflikt)
  - `from <Modulname> import <Funktionsname(n)>`  
Verwendung durch `<Funktionsname>`  
(!gleichnamige Funktionen werden überschrieben!)
  - `from <Modulname> import *`  
Alles wird importiert, gefährlich aber “bequem”

# Dateien lesen und schreiben

`open()` : öffnet eine Datei in angegebenem Modus

- 'r': Lesemodus
- 'w': Schreibmodus !Datei wird überschreiben !
- 'a': Schreibmodus, neue Daten werden am Ende hinzugefügt

`read()` : Lese den Inhalt der Datei; komplett, oder die angegebene Anzahl an Bytes

`write()` : Schreibt Daten in Datei. Zeilenumbruch muss explizit angegeben werden

`close()` : schließt Datei.

# Rekursion

*Um Rekursion zu verstehen,  
muss man erstmal Rekursion verstehen*

- Methode etwas durch sich selbst zu definieren

## Beispiel (Summe)

Die Funktion  $f : \mathbb{N} \rightarrow \mathbb{N}$  sei gegeben durch

$$f(n) := \begin{cases} 0, & \text{falls } n = 0 \\ n + f(n - 1), & \text{sonst.} \end{cases}$$

*Rekursionsanfang  
Rekursionsschritt*

# rekursive Programmierung

- Funktionen die sich selbst aufrufen (auch verschachtelt)
- Abbruchbedingung muss auch erreicht werden (Gefahr der Endlosschleife)

File Edit Format Run Options Windows Help

```
# Summe rekursiv

def sum_rek(n):
    if(n == 0):
        return 0
    else:
        return n + sum_rek(n-1)
```

# ideale Kaninchen

- Ein Kaninchenpaar (m,w)
- ist nach einem Monat geschlechtsreif
- gebären nach einem Monat Tragzeit ein weiteres Kaninchenpaar
- Kaninchen sterben nie
- Wie viele Kaninchenpaare nach  $n$  Monaten ( $KP_n$ )?

$$KP_n = KP_{n-1} + KP_{n-2}$$