PROPÄDEUTIKUM INFORMATIK SOSE 2018

Martin Mehlhose

WIDERHOLUNG

- Variablen und primitive Datentypen in Python
- Grundoperationen auf für Zahlen
- Einlesen der Tastatur

FORMATIERUNG DER AUSGABE

- Funktion print(...) zur Ausgabe auf dem Bildschirm.
- Formatierte Ausgabe mit Platzhaltern %xd für Ganzzahlen und %x.xf für Kommazahlen
- x ist durch die Anzahl der Vor- und Nachkommastellen zu ersetzen
- Eingabe der Werte als Tupel am Ende der print Funktion
- siehe Beispiel im Git

NACHTRAG ZU DEN GANZZAHLOPERATOREN

- / ist die normale Division. Das Ergebnis ist in der Regel ein float
- // ist die ganzzahlige Division. Das Ergebnis ist ein Integer
- % der Modulo Operator ist für alle Zahlentypen gleich

KONTROLLSTRUKTUREN

BOOLESCHE OPERATIONEN

- Negation: **not**
- Logische "und": **and**
- Logische "oder": **or**

VERGLEICHSOPERATOREN

- gleich: ==
- ungleich: !=
- größergleich: >=
- kleinergleich: <=
- strikt größer: >
- strikt kleiner: <

IF ANWEISUNG

- Syntax: **if(logischer Ausdruck): statements**
- Auf eine if Anweisung können mehrere Statements folgen
- Wichtig: Einrückung beachten!

- Syntax: **if(logischer Ausdruck): statements else: statements**

IF ANWEISUNG

- Wichtig: in einem if oder else Block muss immer eine Anweisung stehen!
- soll nichts ausgeführt werden nutze pass als leere Anweisung
- Alle Statements die zu einem Block gehören müssen gleich weit eingerückt sein
- Ein Block ist eine logische Einheit zusammengehörender Anweisungen und kann in Python nur durch die Einrückung erkannt werden.
- Einrückung am besten durch Nutzung der TAB-Taste vornehmen.

BEISPIEL IDENTIFIER

```
# einfache if Anweisung
if(a<b):
    max=b


# if-else Anweisung
if(a<b):
    max=b
else:
    max=a
```

ELIF ANWEISUNG

- Soll nach mehr als zwei Fällen unterschieden werden, dann nutze elif (logischer Ausdruck):
- So können beliebig viele Fälle abgearbeitet werden.
- Es würd von oben nach unten der Erste Block bearbeitet, wo der logische Ausdruck zu **True** validiert.
- Sollen alle positiven Blöcke ausgeführt werden ist if statt elif zu verwenden.

BEISPIEL ELIF

```
if (a % 5 == 0):
    print("a ist durch 5 teilbar")
elif (a % 7 == 0):
    print("a ist durch 7 teilbar")
elif (a % 10 == 0):
    print("a ist durch 10 teilbar")
else:
    pass
```

BEISPIEL IF

```
if (a % 5 == 0):
    print("a ist durch 5 teilbar")
if (a % 7 == 0):
    print("a ist durch 7 teilbar")
if (a % 10 == 0):
    print("a ist durch 10 teilbar")
else:
    pass
```

FUNKTIONEN

FUNKTIONEN

- Ziel der objektorientierten Programmierung ist es widerverwendbaren und gut lesbaren Code zu schrieben
- Einzelne Funktionalitäten eines Programmes werden daher in Funktionen gekapselt
- So können sie an jeder Stelle des Programms widerverwendet werden
- Verbessert außerdem die Möglichkeiten des Testens einzelner Komponenten

FUNKTIONEN

- Funktionen werden wie folgt definiert:
- **def functionName (Variablenliste):**
- Danach kommen eingerückt alle Anweisungen der Funktion
- Hier gelten die selben Regeln zum Einrücken wie bei if Anweisungen

FUNKTIONEN

- Die Variablenliste kann leer sein oder beliebig viele, kommagetrennte Variablen enthalten
- Die Funktion kann auch ein Ergebnis zurück liefern:
- **return ergebnis**
- Für die Funktionsnamen gelten die selben Regeln wie für Variablen

BEISPIEL FUNKTIONEN

```
def beispielFunktion():
    print("TEST")

def addieren(zahl1, zahl2):
    ergebnis = zahl1 + zahl2
    return ergebnis
```

FUNKTIONEN

- Jede Funktion muss mindestens eine Anweisung haben
- Soll die Funktion erstmal leer bleiben ist die **pass** Anweisung zu nutzen

- Zukünftig wollen wir alle Aufgaben in Funktionen kapseln

SCHLEIFEN

FOR-SCHLEIFE

- Syntax: **for var in Iterable: Anweisungen**

- var ist eine selbstgewählte Variable
- Iterable kann jede Collection von Daten sein, die man durchlaufen kann. z.B.: Listen, Tupel, usw.
- Für uns vorerst nur Zahlen

BEISPIEL 10 MAL HELLO WORLD AUSGEBEN

```
for count in range(10):
    print("HelloWorld")
```

FUNKTION RANGE()

- Die Funktion Range gibt grundsätzlich eine Sequenz von Zahlen zurück
- range(n) liefert alle Zahlen von 0 bis n-1
- range(start, ende) liefer alle Zahlen von start bis stop-1
- range(start, stop, step) wie bisher, aber mit step kann die Schrittweite beeinflusst werden

GIB ALLE GERADEN ZAHLEN BIS 100 AUS

```
for count in range (0, 100, 2)
    print(count)
```

ODER MIT IF-ANWEISUNG

```
for count in range(100):
    if(count % 2 == 0):
        print(count)
```