

# 706.088 INFORMATIK 1

## EINFÜHRUNG IN DIE PROGRAMMIERUNG

# WIEDERHOLUNG

1 . 2

# ZUSAMMENFASSUNG - ÜBERBLICK

- › Werkzeuge zur Programmierung
  - » Compiler, Interpreter
- › Python3
  - » Strukturen, Variablen, Schleifen etc.

Wie wird aus "Text" ein ausführbares Programm?

2.1

# PROGRAMMIER WERKZEUGE

Bei der Übersetzung von **Quellcode (Source-Code)**,  
geschriebenem Programmcode, übersetzt der Computer  
das Programm in Maschinensprache.

2 Arten der Übersetzung...

- › **Compiler**
  - » einmalige Übersetzung in eine Binärdatei
- › **Interpreter**
  - » Interpretation des Quellcodes bei der Ausführung

# COMPILER

- › übersetzt Befehle in Maschinensprache
- › Das entstandene Programm enthält nur mehr Maschinenbefehle
  - » ist nicht mehr für den Menschen lesbar
  - » als Binärdatei gespeichert
- › Übersetzung findet **einmalig** statt
- › Vorgang wird "Compilierung" genannt
- › Computer führt Programm direkt aus

# INTERPRETER

- › ähnlich zum Compiler wird das Programm in Maschinensprache ausgeführt, allerdings
  - » wird jeder Befehl erst bei seiner Ausführung übersetzt
  - » entsteht kein zusätzliches, übersetztes Programm
- › Programme werden **bei jeder Ausführung erneut übersetzt**
- › Dieser Vorgang wird als "Interpretation" bezeichnet
- › Je nach Art des Programms kann die Verwendung eines Compilers Geschwindigkeitsvorteile bringen

# VORAUSSETZUNGEN FÜR DIE ÜBUNG

- › Computer (Linux, Mac Os X, Windows)
  - » kein Tablet
  - » kein Handy
- › Texteditor (Atom, vim, Emacs, Notepad, etc)
- › Konsole (xterm, Terminal.app, etc.)
- › Python3 Interpreter (CPython)



PYTHON

4 . 1

# PYTHON3

- › für Übung: Python 3.5
- › gut lesbar, klar strukturiert
- › interpretiert
- › seit 1991, Python3 seit 2008
- › läuft auf allen gängigen Betriebssystemen

# PYTHON

- › Informationen zu Python: [www.python.org](http://www.python.org)
- › Python Dokumentation: [docs.python.org](http://docs.python.org)
- › Python Tutorial: [docs.python.org/3/tutorial](http://docs.python.org/3/tutorial)

# PYTHON CHEATSHEETS

Google: Python Cheat Sheet

- › [www.cogsci.rpi.edu\[...\]pdf](http://www.cogsci.rpi.edu/.../pdf)
- › [www.astro.up.pt\[...\]pdf](http://www.astro.up.pt/.../pdf)
- › Such Link...

# HELLO WORLD!



# KONSOLE

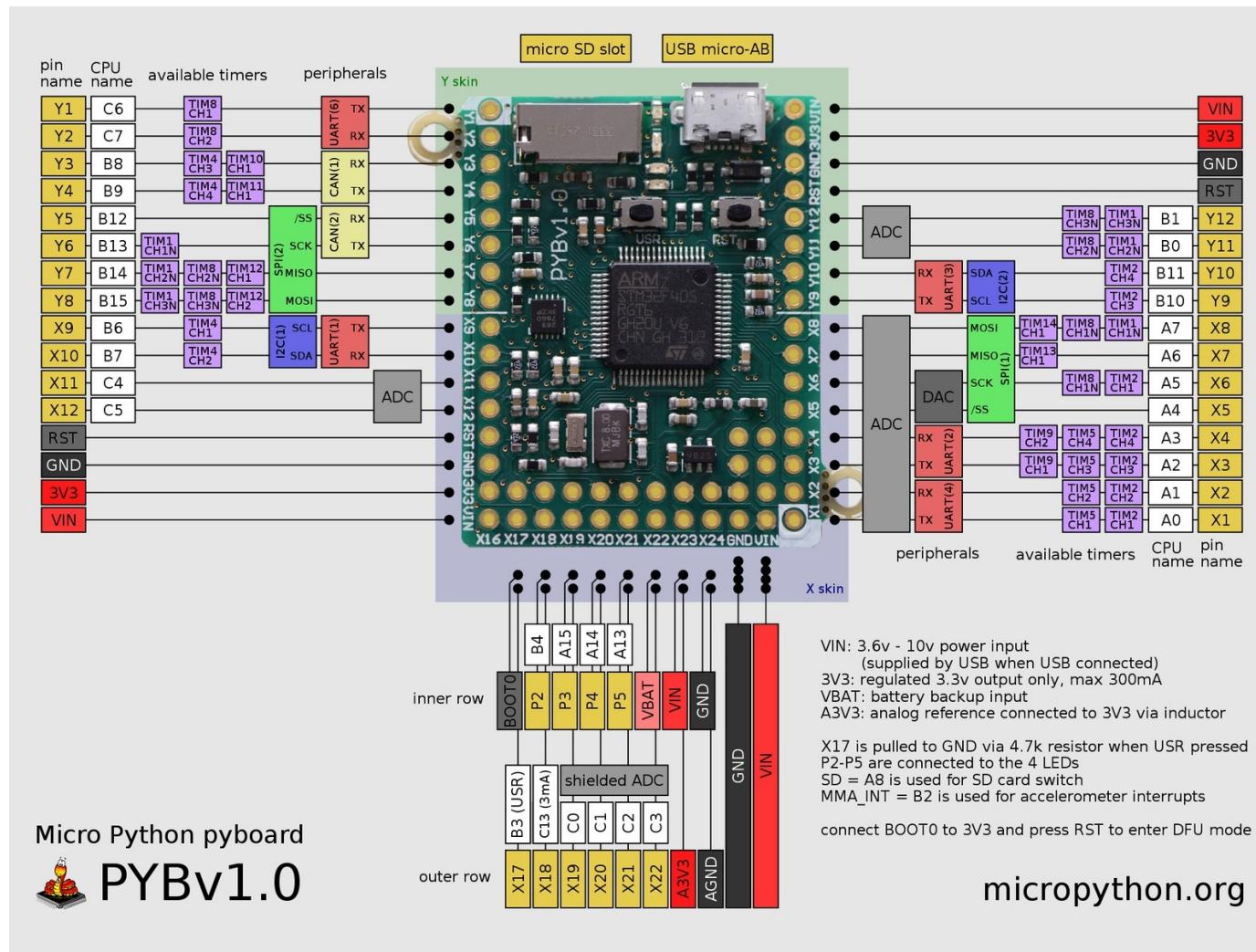
- › Navigation
- › Basiskommandos (cd, cd .., mkdir, man)
- › **Linuxtage Merkzettel für Fortgeschrittene**

# LERNZIELE

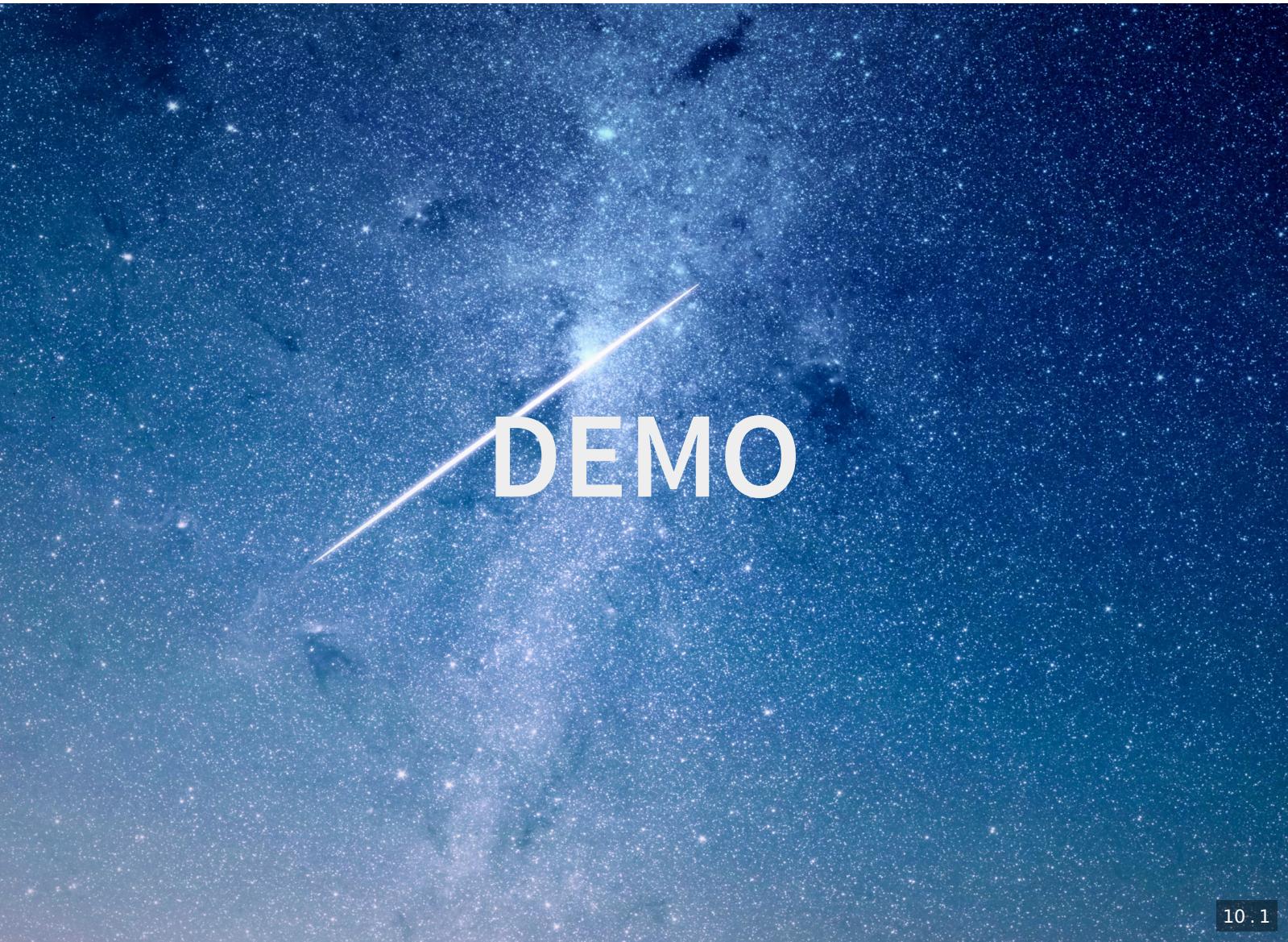
- › Erstellen eines Programms
  - » Arbeitsschritte, Tools
- › Programmablauf steuern
  - » Variablen, Datentypen
  - » Kontrollstrukturen
  - » Schleifen
- › Programm strukturieren
  - » Funktionen
  - » Klassen

# WARUM PYTHON

- › **Scriptsprache**
  - » Abstrahiert komplexe Konzepte
  - » Einfacher Einstieg
    - › Texteditor, Programm & Konsole
  - » Gute Dokumentation
  - » Zwingt zu klarer Code-Struktur
  - » Für alle grossen Plattformen verfügbar
  - » sogar auf Micro-Controllern → **µPython**







DEMO

10.1

# VIDEO



 go there

# AUFBAU EINES PROGRAMMS

```
import sys

# Kommentar
#main() function
def main():
    # Hier startet die Abarbeitung
    return 0

# pythonic way um main zu definieren
if __name__ == "__main__":
    main()
```

# SYNTAX - EINRÜCKUNG

```
import sys

def main():
    a = 10
    b = 20
    print(a)
    print(b)

c = 30
print(c)

if __name__ == "__main__":
    main()
```

```
$ python3 test.py
30
10
20
```

# EINRÜCKUNGSREGELN - PEP8

## PEP8 Style Guide

- › 4 Spaces (Leerzeichen) pro Level bei Einrückung
- › max. 79 Zeichen pro Zeile
- › Im Detail studieren und für Übung beachten!
- ›  Beispiele der Vorlesung halten sich nicht unbedingt an diese Vorgaben (Platzbedarf)

```
# Aligned with opening delimiter.  
foo = long_function_name(var_one, var_two,  
                         var_three, var_four)  
  
# More indentation included to distinguish this from the rest.  
def long_function_name(  
    var_one, var_two, var_three,  
    var_four):  
    print(var_one)  
  
# Hanging indents should add a level.  
foo = long_function_name(  
    var_one, var_two,  
    var_three, var_four)
```

# BESTANDTEILE VON PROGRAMMEN

## BEGRIFFE

strukturierte Programme bestehen aus verschiedenen Elementen:

- › Variablen & Datentypen
- › Operatoren
- › Kontrollstrukturen
- › Schleifen
- › Funktionen, Signaturen & Parameter
- › Eingabe-/Ausgabe Parameter

# **VARIABLEN & DATENTYPEN**

# VARIABLEN & DATENTYPEN

- › **Variablen** sind Platzhalter
- › **Datentypen** definieren wofür eine Variable Platz hält.

```
kilometers = 42  
car_name = "Volkswagen"
```

| Name       | Typ                                |
|------------|------------------------------------|
| kilometers | <i>natürliche Zahl mit Wert 42</i> |
| car_name   | <i>Text mit Wert Volkswagen</i>    |

# DEKLARATION VON VARIABLEN

Variablen sind Platzhalter für Werte.

Zuweisung eines Werts heisst **deklarieren**.

# VARIABLEN NAMEN

Namen sollen englisch, **kurz** und **eindeutig** sein, keine Leerzeichen und Sonderzeichen (außer '\_') enthalten.

Ausserdem sind folgende Worte in Python **reserviert**:

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

# EINFACHE WERTE

```
# Wahrheitswerte (Boolean)
var = True
# Natürliche Zahl (Integer)
var = 1234
# Reelle Zahl (Float)
var = 3.1415
# Text (String)
var = "Ein Text hat viele Zeichen"
# Kein Wert
var = None
```

# SEQUENZIELLE DATENTYPEN

```
liste = [1, 2, 3] # Liste aus Integern
liste[0]

string = "Das ist ein Text" # String
string[1]
string[-2]
string

dic = {"element1": 5, "element2": 6}
dic["element1"]
```

```
>>>
1
'a'
'x'
'Das ist ein Text'
5
```

# WAHRHEITSWERTE

| Datentyp | False-Wert |                     |
|----------|------------|---------------------|
| bool     | False      | Wahrheitswert False |
| int      | 0          | numerisch Null      |
| float    | 0.0        | numerisch Null      |
| str      | ""         | leerer String       |
| list     | []         | leere Liste         |
| dict     | {}         | leeres Dictionary   |

# GÜLTIGKEIT VON VARIABLEN (SCOPE)

```
import sys

GLOBAL1 = 1

def main():
    a = 10
    print(a)
    print(GLOBAL1)

#print(a)
if __name__ == "__main__":
    main()
```

```
$ python3 scope.py
10
1
```

```
import sys

GLOBAL1 = 1

def main():
    a = 10
    print(a)
    print(GLOBAL1)

print(a)
if __name__ == "__main__":
    main()
```

```
$ python3 scope.py
Traceback (most recent call last):
  File "scope.py", line 10, in <module>
    print(a)
NameError: name 'a' is not defined
```

# OPERATOREN

Operatoren definieren die Interaktion von Variablen und (deren) Werten

$$a = 3, b = 4, c = 2$$

| Operator | Beispiel      | Ausgabe       |
|----------|---------------|---------------|
| +x, -x   | -c            | -2            |
| +, -     | a + b         | 7             |
| *, /     | a * b, b / a  | 12, 1.3333333 |
| //, %    | 13//3, 13 % 3 | 4, 1          |
| **       | a**c          | 9             |

# LOGISCHE OPERATOREN

Boolsche Operatoren

| Operator | Beispiel       | Ausgabe |
|----------|----------------|---------|
| not      | not True       | False   |
| or       | True or False  | True    |
| and      | True and False | False   |

# KOMPERATOREN

| Komperator | Beispiel     | Ausgabe |
|------------|--------------|---------|
| <          | $1 < 2.1$    | True    |
| >          | $1 > 1$      | False   |
| $\leq$     | $1 \leq 1.0$ | True    |
| $\geq$     | $2 \geq 3$   | False   |
| $\equiv$   | $2 \equiv 2$ | True    |
| $\neq$     | $2 \neq 2$   | False   |

# DIVISION

```
a = 9  
b = 3  
c = 10  
  
print(c//b) # Ganzzahldivision  
print(c/b) # Division (Datentyp wird angepasst: Ergebnis ist Float)
```

```
3  
3.333333333333335
```

# DIVISION

```
a = 9.0  
b = 3.0  
c = 10.0  
  
print(a/b)  
print(c/b)
```

```
3.0  
3.3333333333
```

# KONTROLLSTRUKTUREN

- › Werden benötigt um den Ablauf eines Programms zu steuern
  - » if
  - » else
  - » elif

# IF ELSE

```
a = 10  
b = 12  
  
if (a > b):  
    print("a > b")  
else:  
    print("b >= a!")
```

```
b >= a!
```

# IF ELIF ELSE

```
a = 10
b = 2

if a < 10 and b > 0:
    print(a/b)
elif a > 10 or b > 2:
    print("a > b")
else:
    print("neither, ...")
```

```
neither, ...
```

# IF VERSCHACHTELT

```
a = 10
b = 2

if a < 15:
    if b > 0:
        print("a = {}".format(a))
    else:
        print("b = {}".format(b))
else:
    print("a >= 15!")
```

```
a = 10
```



# SCHLEIFEN

Ermöglichen mehrere Durchläufe von Codestellen

- › while
  - » solange bis ...
- › for
  - » durch iterierbare Objekte durchlaufen (zB Listen)

# WHILE

```
a = 1
while a < 10:
    a += 1 # same as: a = a + 1
    #print(a)
print(a)
```

```
10
```

# WHILE

kann mit *continue* von oben fortgesetzt werden:

```
a = 1
while a < 10:
    a += 1
    if not a % 2:
        print(a)
        continue
    a += 2
print(a)
```

```
2
6
10
10
```

# WHILE

kann mit *break* abgebrochen werden:

```
a = 1
while a < 10:
    if a > 3:
        break
    a += 1
print(a)
```

```
4
```

# FOR

Schleife um durch iterierbare Objekte zu laufen

```
ark = ["Camel", "Cat", "Dog", "Snake", "Cow"]

for animal in ark:
    print(animal)
```

```
Camel
Cat
Dog
Snake
Cow
```

# FOR ALS ZÄHLSCHLEIFE

mit Hilfe der Funktion *range()*

```
for count in range(0,5): # von 0, bis exklusive 5
    print(count)
```

```
0
1
2
3
4
```

```
for count in range(0,5,2): # mit Schrittgröße 2
    print(count)
```

```
0
2
4
```

# DEFINITION EINES GUTEN PROGRAMMS

- › Funktionalität
- › Lesbarkeit
- › Wartbarkeit
- › Robustheit
- › Korrektheit
- › Effizienz
- › Portierbarkeit

# FUNKTIONEN

# FUNKTIONEN

dienen der Kapselung von Code nach logischen Einheiten

```
pets = ["Hamster", "Cat", "Dog", "Canary"]

def not_a_dog_person(animal):
    return animal.replace("Dog", "Fish")

for pet in pets:
    print("Tom: {:10} Tim: {}".format(pet, not_a_dog_person(pet)))
```

```
Tom: Hamster      Tim: Hamster
Tom: Cat          Tim: Cat
Tom: Dog          Tim: Fish
Tom: Canary       Tim: Canary
```

# FUNKTIONEN

```
def add(x, y):
    return x + y

def print_result(result):
    print("Result: {}".format(result))

def main():
    a = 10
    b = 20
    c = add(a, b)
    print_result(c)

if __name__ == "__main__":
    main()
```

```
Result: 30
```

# FRAGEN?

27