Programmierung mit Python

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2026-09 Dozent: Karsten Flügge

Themen des Tages

- Variablen und einfache Datentypen
- Strings / Texte
- Bibliotheken (import) "module"
- Datum und Zeit

Zahlen in python

```
# Integer (≈ entire Ganzzahlen) <class 'int'>
x = 5

# Float (Fließkommazahlen) <class 'float'>
y = 3.14

Zahl aus Text extrahieren:

z = int('7')
z == 7  # True

Zahl in Text umwandeln:
t = str(7)  # string
t == '7'    # True
```

Scientific Notation

```
Statt 300_000 3 * 10**5 darf man einfach schreiben 3e5
heist: 3 mit 5 Nullen

Statt 3 * 10**-5 darf man einfach schreiben 3e-5
heist: 3 mit 5 Nullen davor: 0.00003
```

Zahlen in python

```
Achtung: Buchstabe (character) Zeichen 'Ziffer' vs Zahl

>>> '7'+'7'
'77'
>>> int('7')+int('7')
14
```

Zahlen in python

Brüche sind nur per Erweiterung vorhanden
bruch = a/b # macht in python **float**
a nennt Zähler (numerator)
b nennt man Nenner (divisor or **denominator**)

```
>>> x.denominator
1  # Konstant 1 für int (ganze Zahlen)
>>> x.numerator
78
>>> 78 == 78/1
```

```
from fractions import Fraction

# Example rational number
r = Fraction(22, 7)

# Get the denominator
denominator = r.denominator
print(denominator) # 7
```

Zahlen in python

```python
my_number_as_text = input('gib mir Zahl!') # immer als Text
my_number = int(my_number_as_text)  # str als int Zahl umwandeln
print("Quadrat", my_number* my_number)
```

```python
#For floating-point numbers, this truncates towards zero.
pi = 3.1415
pi_abgerundet = int(pi) # 3 == 3.0
```

Kommentare

In Python und anderen Sprachen sind Kommentare Texte im Code, die nicht ausgeführt werden. Sie dienen dazu, den Code verständlicher zu machen, indem sie Erklärungen oder Anmerkungen zu bestimmten Teilen des Codes bieten.

```
# Kommentar ≈ Dokumentation
print("Hallo 🌍") # Ausgabe von Text auf der Konsole kann auch Sonderzeichen enthalten
```

Kommentare sind auch nützlich zum 'deaktivieren' von Kommandos und Code :

```
# print("Kein Hallo 🌎")  # Zeile 'auskommentiert' mit # => nichts geschieht ;)

Um zu verdeutlichen dass kommentierter Code keine Funktion hat wird dieser
typischerweise in Editoren in einer hellen Farbe zum Beispiel grau dargestellt.
```

Variablen

Variablen sind ein grundlegendes Konzept in jeder Programmiersprache, einschließlich Python.
Sie sind wie Behälter, die Daten speichern, die das Programm verwenden kann. "Kurzzeitgedächtnis"

Synonyme: Platzhalter, Referenz

```
gruss = "Hallo Welt" # Zuweisung
print(gruss)
```

```
ergebnis = 3 * 2 # ⚠️ Achtung, Formeln werden direkt und automatisch ausgerechnet
print(ergebnis) # Ausgabe: 6  (nicht : 3 * 2)
```

```
>>> ergebnis = 3 * 2  # Zuweisung
>>> ergebnis == 3 * 2 # Vergleich
True
>>> ergebnis == 3 * 3
False
>>> ergebnis = 3 * 3
>>> ergebnis
9
```

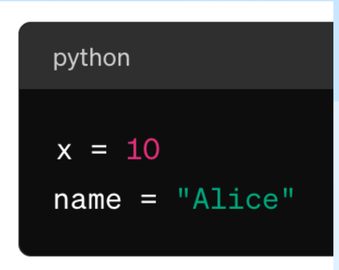
Variablen sind flüchtig, das heißt sie gehen am Ende des Programmes verloren. Ausweg: **Speichern** in Dateien

Variablen Deklaration

Eine Variable ist ein Name / Bezeichner in dem man Werte / Objekte abspeichern kann

x = 2
y = 3

In Python musst du eine Variable nicht explizit deklarieren, bevor du sie verwendest. Du weist ihr einfach einen Wert zu, und Python erkennt automatisch den Typ.
Die Zuweisung erfolgt mit dem Gleichheitszeichen (=). "sei, das soll so sein!"
Nicht zu verwechseln mit dem Vergleich (==) "ist das so? ist das gleich**?**"




```
>>> type(x)  # wird direkt aufgelöst zu dem Objekt welches 'in x steckt'
<class 'int'>
```

Variable = veränderlich

```
"Variable" heisst 'veränderlich'

# Variablen können Werte speichern, die sich verändern können
alter = 25
print(alter)
# Geburtstag
alter = 26
print(alter)
```

Python ist eine dynamisch typisierte Sprache, was u.a. bedeutet, dass der Typ einer Variablen geändert werden kann, wenn ihr ein neuer Wert zugewiesen wird.

```
# Variablen in Python können Werte von verschiedenen Typen speichern
alter = 25 # Zahl
print(alter)
alter = "fünfundzwanzig" # Text
print(alter)


Achtung: in Mathematik bleiben Variablen fest, wenn einmal 'gefunden' x^2 = 4    x=2 / x=-2
```

Selbstveränderlich

```
"Variable" heisst 'veränderlich'

# Variablen können Werte speichern, die sich verändern können
alter = 25
print(alter)

# Geburtstag
# alter = 26
alter = alter + 1  # alter Wert in neuer Zuweisung (nicht in Mathe!)
print(alter)
```

Konstante Variablen ;)

In Python gibt es **keine eingebauten Konstanten.** Es ist jedoch eine **Konvention**, Namen von Variablen, die konstant sein sollen, in **Großbuchstaben** zu schreiben.

```
PI = 3.14159
```

```
Auswege: class __PI
Ausweg: Methode/Funktion  math.PI()
Ausweg: pi: Final[float]
```

Variablennamen

Variablennamen in Python können Buchstaben (a-z, A-Z), Ziffern (0-9) oder Unterstriche (_) enthalten, dürfen jedoch nicht mit einer Ziffer beginnen. Außerdem sind Variablennamen in Python case-sensitive, d.h., name, Name und NAME sind drei verschiedene Variablen.

```
hallö = "Hallo Erde" # Variablennamen können (seit ___) auch Umlaute enthalten

hallo🌎 = "Hallo Erde" # ⚠️ Variablennamen dürfen (noch) keine allgemeinen Sonderzeichen enthalten
```

Namen von Variablen sind als Konvention meist *Englisch* und starten mit kleinem Buchstaben

Namen von Variablen die sich nie ändern "Konstanten" werden als Konvention meist in GROSS-Buchstaben geschreiben

```
CONSTANT_PI = 3.1415 # Konstanten werden in Großbuchstaben geschrieben

舌 = 3 # heute erlaubt!
print(舌)
```

Variablennamen

Variablennamen sollten schön leserlich sein (Abkürzungen vermeiden)

Variablennamen in Python können Buchstaben (a-z, A-Z), Ziffern (0-9) oder Unterstriche (_) enthalten, dürfen jedoch nicht mit einer Ziffer beginnen. Außerdem sind Variablennamen in Python case-sensitive, d.h., name, Name und NAME sind drei verschiedene Variablen.

```
myage   unreadable case  (VERMEIDEN)
myAge   camel case 🐫 (jedes neue Wort im Namen wird Groß geschrieben)
my_age  snake case 🐍 (jedes neue Wort im Namen wird mit Unterstrich verbunden) << PEP Konvention
my-age  kebab case (in python … verboten)  Subraction Minus??
my age  space case (in python … verboten)
```

```
Typischerweise vom variablen Namen mit einem kleine Buchstaben an:  alter = 37

Großbuchstaben am Anfang, wie 'Alter' etc ist 'reserviert' für Klassen und Typen Namen

int => Integer
math.pi => math.PI
```

Variablen vs Text

a Buchstaben(folgen) ohne Anführungszeichen sind **Variablen Namen**

'a' Buchstaben(folgen) mit Anführungszeichen sind **Text (str)**

1 Zahlen(folgen) ohne Anführungszeichen sind **Zahlen**

'1' Zahlen(folgen) mit Anführungszeichen sind **Text (str)**

Syntax

Syntax beschreibt die Regeln, welche Reihenfolgen von Worten und Zeichen in einem Programm erlaubt sind.

```
Wie Grammatik in Sprachen.
```

```
ich esse Toast
ich essen Toast
```

```
# my age = "space case (in python … verboten)" # SyntaxError: invalid syntax
```

Datentypen

Python unterstützt verschiedene Datentypen, die als Variablen gespeichert werden können, darunter:

```
# Integer (Ganzzahlen) <class 'int'>
x = 5

# Float (Fließkommazahlen) <class 'float'>
y = 3.14

# Strings (Zeichenketten, Text)  <class 'str'>
name = "Bob"
```

x = None # "nichts", zum Beispiel das Resultat von print() in anderen Sprachen null oder nil

```
# Boolean (Wahrheitswerte Wahr, Falsch) <class 'bool'>
is_valid = True # or False

# Listen <class 'list'>
fruits = ["apple", "banana", "cherry"]

# Maps Dictionaries (Wörterbücher Abbildungen)  <class 'dict'>
person = {"name":"Anna", "age": 28}
translations = {"dog":"Hund", "cat": "Katze"}
```

Datentypen

Python unterstützt verschiedene Datentypen, die als Variablen gespeichert werden können:

```
# Integer (Ganzzahlen) <class 'int'>
x = 5
```

Objekt vom **Typ** / von der **Klasse** (class) int
Objekte sind **Instanzen (Individuen)** von Klassen

Klasse = Hund
Objekt = **Bello**

Objekte haben meistens einen individuellen Namen,
in python **Variablen** Namen.

Synonyme für Instanzen in diesem Kontext:

•Exemplare •Entitäten •Ausprägungen •Konkrete Objekte •**Individuen** •Kreationen

Daten Typen

Zeichen versus Zahlen

'3' + '3' = '33'

3 + 3 = 6

Der Typ kann bei Bedarf mit der type() Funktion abgefragt werden

```
x = 3
type(x)    # Funktion die den DatenTyp eines Objektes liefert
type(3)    # <class 'int'>
type('3')  # <class 'str'>

help(...)    Funktion erklärt Variablen, Objekte und Typen
dir(...)     Funktion gibt eine Liste von Funktionen von einem Objekt
```

Aufgaben

- **a)** Speichert Euren Vornamen und Euren Nachnamen jeweils in einer eigenen Variable und gebt den ganzen Namen dann zusammen aus mit einem Space dazwischen **print(...)**

```
vorname="Max"
...
```

b) Lese einen Namen mit **input(**"wie heißt Du?"**)** ein und begrüße dann diese Person.

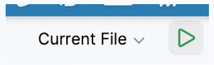
c) Lese ein Alter mit Input ein und gebe dann aus, herzlichen Glückwunsch, du bist jetzt ...

d) Erhöhe dieses Geburtsdatum um eins und sage: Du bist nächstes Jahr ... Tip: **int**(jahr)+1

e) speichert zwei Zahlen und dann das Ergebnis der Multiplikation in einer neuen Variable

Ausgabe stets mit **print(...)**

Vor dem Starten mit grünem Pfeil immer auf "aktuelle Datei" achten:



Strings

"Texte" ≈ Strings (str) ZeichenKetten
'a b c' Jeden Buchstaben kann man sich vorstellen wie ein Glied auf einer Kette.

Strings sind in Python und vielen anderen Programmiersprachen eine wesentliche **Datenstruktur / Typ**, die für die Verarbeitung von Text verwendet wird. In Python sind Strings eine Folge von Zeichen, die in einfache (') oder doppelte (") Anführungszeichen eingeschlossen werden.

💡

```
# ⚠️ Achtung, in Python gibt es keine 'Char'-Datentypen (character/Buchstaben)
'a' == "a" !!
```

*Stattdessen werden einzelne Zeichen in Python als Strings mit der **Länge 1** behandelt.*

Strings

Texte ≈ Strings (str) ZeichenKetten

Was darf alles in einem Text stehen?

Alles ausser dem passenden Anführungszeichen "

Buchstaben, Zahlen, Sonderzeichen... Unicode! (Tag 13)

Ausweg um " zu verwenden: Anderes ' verwenden:

```
'Er sagt "Hallo"'
"Mike's cat"
```

Was ist wenn man beides verwenden möchte? Backslash \
Backslash \ ist ein besonderes Zeichen mit dem man besondere Dinge im Text einfügen kann

```
"Er sagt \"Hallo\""
```

Backslash \

Was ist wenn man beide Tüttel in einem String verwenden möchte? Backslash \" \'
Backslash \ ist ein besonderes Zeichen mit dem man besondere Dinge im Text einfügen kann
"Er sagt \"Hallo\""

```
\n    # newline
\t    # tab
\\    # literal backslash
\'    # single quote
\"    # double quote
\r    # carriage return (altes Windows Zeugs)
\b    # backspace (löschen im Text??)
\0    # null character
\xNN  # character by 2-digit hex value
\uNNNN       # Unicode, 4 hex digits
\UNNNNNNNN   # Unicode, 8 hex digits
```

```
print("C:\\Users\\name")
print("\u03C0")       # π
print("\U0001F600")   # 😀
```

Strings

```
# Länge

len("Hallo") # Ausgabe: 5    exteren 'Funktion' wie print()
# Spezieller Befehl welcher auf __len__ abgebildet wird!

len("😺") # Ausgabe der Anzahl Zeichen (Buchstaben, Zahlen,
Sonderzeichen, Leerzeichen, emojis...) : 1
# Länge eines Strings in python ist die Anzahl der Zeichen im String,
⚠️  nicht die Anzahl der Bytes wie in anderen Programmiersprachen
⚠️  help("abc".count) nicht was man erwarten mag

"Hallo".len()   # geht nicht!
```

Emojis macOS: ^ Control + ⌘ Command + Space
Windows: Win + .

Strings

concatenate ≈ aneinanderhängen

Strings Kombinieren einfach mit +
Achtung: kein automatisches Space

```
>>> "Hallo" + "Welt"
'HalloWelt'
>>> "Hallo " + "Welt"
'Hallo Welt'
```

```
    "Hallo " + 3
    ~~~~~~~~~^~~
TypeError: can only concatenate str (not "int") to str
>>> "Hallo " + str(3)
'Hallo 3'
```

```
>>> print("Hallo", 3)  # ist etwas anderes
Hallo 3
```

Strings

Zugriff auf einzelne Zeichen in einem String über eckige klammern [] **index offset (Abstand / Versatz)**

```
name = "Alice"
print(name[0])  # Ausgabe: A
```

⚠️ Achtung, Indizes beginnen bei 0, also n-tes Zeichen == [**n-1**] !

```
print("Hi😺"[2])
print("Hi😺"[-1]) # Letztes Zeichen ausgeben
(Index Trick statt len(…)-1 )
😺
```

Teilstring extrahieren (Slicing):

```
name = "Alice"
Spezielle Index Syntax um Teil Bereich 0 bis 3 auszugeben
print(name[0:3])  # Ausgabe: Ali
```

⚠️ *Achtung* [0:3] Zeichen 0 inklusive bis 3 exklusive [0,3)

String Formatierung

Strings können in der Regel mit '+' Zeichen verbunden werden:

```
first_name = "ada"
last_name = "lovelace"
print(first_name + " " + last_name)
print(first_name , " " , last_name)
```

zum Verbinden mit anderen Typen muss man diese aber erst mit **str()** umwandeln:

```
print("Hallo "+str(2024))
```

oder man kann sich Text mit **spezieller f"{}" Syntax formatieren** lassen:

```
print(f"Hello, {2024}!")

year = 2024
print(f"Hello, {year}!")


path = r"C:\Users\Example\Documents"  Backslash wird NICHT gesondert behandelt
path_alternativ = "C:\\Users\\Example\\Documents"  Backslash ist sonderzeichen, muss doppelt sein \\
```

String Iteration

Man kann die einzelnen Zeichen in Strings mit "for-Schleife" durchgehen:

```
for char in "Python":
  print("Gib mir ein", char)
```

```
# mit/für jeden Buchstabe im Text "Python" : tue etwas (print)
```

wird abgebildet auf spezielle Funktion __iter__ für Iteration

```
P
y
t
h
o
n
```

Teil-String Prüfung

'__contains__' beinhaltet ein Text ein Wort / Zeichen ?

Man kann mit **'in'** Befehl sehen ob ein String einen anderen enthält:

```
print("P" in "Python") # Ausgabe: True
print("p" in "Python") # Ausgabe: False
print("tho" in "Python") # Ausgabe: True
print("Welt" in "Hallo Welt") # Ausgabe: True
print(phrase in text)
```

'in' Operator / Keyword => darf man nicht als Variable verwenden

Ersetzen

'**replace**'

```
original = "Hallo Welt! Hallo Python!"
ersetzt = original.replace( was, womit) # ALLE vorkommen
ersetzt = original.replace("Hallo", "Guten Tag") # neues Ergebnis
print(ersetzt)
```

Achtung falls jemand Regex möchte, dafür gibt es eigene Bibliothek!

z.B.

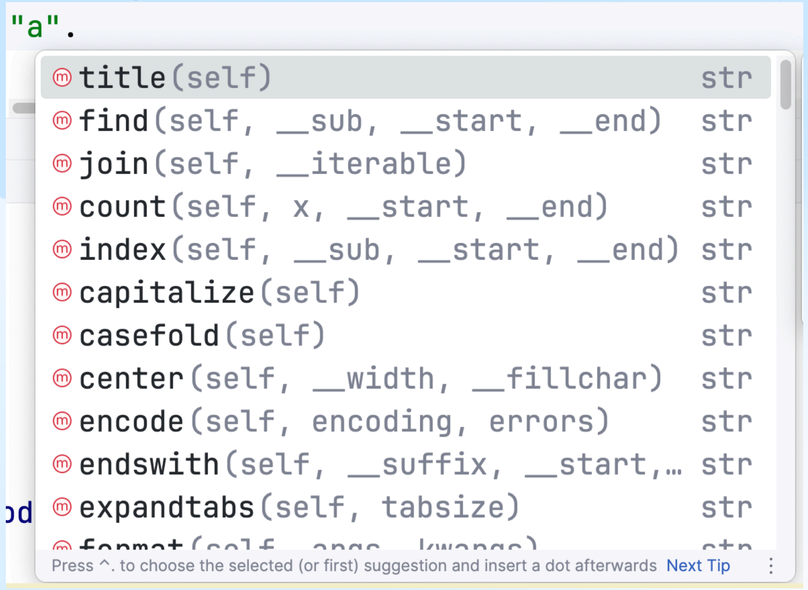
```
original.replace(password,"*****")
```

Strings

*. / help / **dir** / print() / len()* ***Funktionen ohne Punkt "global" gehören zu keiner Klasse/Objekt***

*Es gibt fast **100 Methoden** von String, die kann keiner alle kennen, daher bei Bedarf **GPT** Fragen*

z.B. "max".title() => "Max"
Anzeige in IDE mit `.` oder dir()



Strings

Spezielle Funktionen mit Unterstrichen entsprechen Operatoren, z.B. "a" + "b" == "a".__add__("b") als lesbare Funktion …

```
>>> dir(str)  # directory: zeigt alle Methoden an
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__getstate__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', f'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip' (trim), 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

PS: die **fett** markierten nutze ich oft genug, dass ich sie auswendig kenne

Aufgaben

Aufgabe aktuell: kurz mit String Funktionen **experimentieren**

```
"abc".endswith("c") # True
"a b c".split(" ")  # ['a', 'b', 'c']
" mein Name ".strip() # "mein Name"  entferne Spaces, Tabs, Newlines...
"a".upper()  # 'A'
```

Aufgaben:

a) ersetze in "Hallo Welt" die Welt durch ein Emoji (via python;)
b) Speichere den ersetzten String in einer Variablen und prüfe, ob wirklich das Emoji drin vorkommt. (via python;)
c) Welche der String-Methoden könnte eventuell noch interessant sein?
   dir(str) oder help(str)
d) Lese deinen Namen mit Input ein und schaue nach wie lang der Name ist.

String Aufgaben

- leicht
- a) Schreibe Programm welches nach Namen Fragt und Länge ausgibt

mittel
- b) gebt das erste Zeichen aus ^^
- b1) gib Namen GROSS aus
- b2) prüfe ob der Name ein "a" enthält

schwerer
- c) gebt das letzte Zeichen aus

extra
- d) lies aktuelles Jahr ein und gib nächstes Jahr aus

- e) gib aus ob ein Name mit 'B' beginnt

Strings

Lange Texte lassen sich mit dreifachem " abbilden:

```
"""
Dies ist ein mehrzeiliger String,
bla
"""
```

Keywords / Schlüsselworte

Keywords / Schlüsselworte sind Worte die in der Sprache fest eingebaut sind, nicht als Variablenname benutzen.

z.B.
`in` : "allo" in "Hallo" # True
`True` `False` `None`
`class`
`import`

Keywords / Schlüsselworte

Bibliotheken / module

In Python gibt es quasi zu jedem Problem schon eine 'vorverpackte' Lösung.

Diese Lösungen sind in sogenannten **Modulen** oder **Bibliotheken** verpackt, welche man mit dem **import** Befehl hinzuladen kann.

Beispiel: **import math** (Fest eingebaut)

⚠️ Einige Bibliotheken müssen erst im System installiert werden bevor sie importiert werden können! Das geht über Bibliotheken manager **pip** oder **conda (bei Anaconda).**

Sehr spezielle andere Bibliotheken stehen nicht mit PIP zur Verfügung, sondern müssen von GitHub oder ähnliches heruntergeladen werden.

Bibliotheken / module

Viele (mathematische) Funktionen stehen bereits im "**default namespace**"* zur Verfügung, aber in der Regel muss man sich **Zusatzfunktionalität** aus Bibliotheken hinzuladen.

Dies geht über den **import Befehl.**

⚠️ Einige Bibliotheken müssen erst im System installiert werden bevor sie importiert werden können! Das geht über Bibliotheken manager **pip** oder **conda (bei Anaconda).**

* In Python bezieht sich der Begriff "default namespace" auf den Namensraum, der standardmäßig verwendet wird, wenn ein Programm ausgeführt wird. Ein Namespace in Python ist eine Art Wörterbuch oder Karte, die Namen (zum Beispiel von Variablen oder Funktionen) auf Objekte abbildet.

```
print(dir())
print(dir(__builtins__)) # Anzeigen was im default namespace schon definiert ist
```

Bibliotheken / module

```
# import MODULNAME # Lade ALLE Klassen
import datetime # eingebautes python Modul wird komplett geladen

datetime.datetime # Modul datetime mit Typ datetime

# from MODUL import CLASS # importiert eine Klasse aus einem Modul
from datetime import datetime, date # Klasse Datetime selektiv

datetime direkt
```

Wichtige Bibliotheken / module

```
import datetime
import math
import sys
import os

print(sys.argv) # Argumente Beim Aufruf des Programms

os.system("bash command!")
```

Datum und Zeit

Beispiel für importierte Modul-Funktionalität:

```
from datetime import datetime
jetzt = datetime.now()
print(jetzt) # 2024-04-29 14:48:52.923587

formatiert = jetzt.strftime("%d.%m.%Y") # oder selbst:
formatiert = f"{jetzt.day}.{jetzt.month}.{jetzt.year}"
print(formatiert) # 29-04-2024
```

Aufgabe jetzt: Experimentieren,

dir() help() . tab-tab

⚠️ *Bei den abertausenden Bibliotheken macht es wenig Sinn viel auswendig zu lernen, es geht nur um das Verständnis der Prinzipen*

Datum und Zeit

```
Beispiel Anwendung: Performance Messung

from datetime import datetime

start = datetime.now()
print("Zeitmessung") # lange Routine
ende = datetime.now()

print(ende - start,"ms")
```

Aufgaben datetime

a) Was ist aktuelle Stunde
b) Zeit bis zur Pause
c) Zeitdifferenz zu NYC
d) Gib das Datum von Weihnachten formatiert aus (mit python)

help, dir
Chatty fragen
KEIN copy paste, selber tippen!

Datum und Zeit

```
>>> nun = datetime.datetime.now()
>>> dir(nun)
['__add__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__',
'__radd__', '__reduce__', '__reduce_ex__', '__repr__', '__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', 'astimezone', 'combine', 'ctime', 'date', 'day', 'dst', 'fold', 'fromisocalendar', 'fromisoformat',
'fromordinal', 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday', 'max', 'microsecond', 'min', 'minute',
'month', 'now', 'replace', 'resolution', 'second', 'strftime', 'strptime', 'time', 'timestamp', 'timetuple', 'timetz',
'today', 'toordinal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow', 'utcoffset', 'utctimetuple', 'weekday', 'year']

>>> nun.    # tab tab !
nun.astimezone(       nun.fromisoformat(    nun.microsecond       nun.strftime(         nun.tzinfo
nun.combine(          nun.fromordinal(      nun.min               nun.strptime(         nun.tzname(
nun.ctime(            nun.fromtimestamp(    nun.minute            nun.time(             nun.utcfromtimestamp(
nun.date(             nun.hour              nun.month             nun.timestamp(        nun.utcnow(
nun.day               nun.isocalendar(      nun.now(              nun.timetuple(        nun.utcoffset(
nun.dst(              nun.isoformat(        nun.replace(          nun.timetz(           nun.utctimetuple(
nun.fold              nun.isoweekday(       nun.resolution        nun.today(            nun.weekday(
nun.fromisocalendar(  nun.max               nun.second            nun.toordinal(        nun.year
>>> nun.
```

⚠️ *Bei den abertausenden Bibliotheken macht es wenig Sinn viel auswendig zu lernen, es geht nur um das Verständnis der Prinzipen*

Vorschau Funktionen

```python
# in Mathe: f(x) = 2x
# in Python:
def f(x):
  return 2 * x
#  ^^^ return ist zum Zurückgeben von Werten immer nötig!
# def ist kurz für 'definiere'

def hoch(x, y):
  return x ** y

# x, y heissen Parameter / Argumente
```

🥺 eingebaute Typen (int, string, list) lassen sich leider NICHT erweitern (nur eigen Klassen)

also 2.hoch(3) geht nicht, nur hoch(2,3) oder myInt(2).hoch(3)

Vorschau Kontrollstrukturen

Kontrollstrukturen sind wichtige Bausteine in der Programmierung, insbesondere in Python. Sie ermöglichen es dem Programm, Entscheidungen zu treffen, Wiederholungen auszuführen und somit auf verschiedene Eingaben und Situationen flexibel zu reagieren. Hier sind die grundlegenden Kontrollstrukturen in Python:

Bedingte Ausführung "IF"

```
temperatur = 22
if temperatur > 25:
   print("Es ist warm")
```

"Wenn Temperatur größer 25 ist, sag es ist warm"

```
if Bedingung :
   Dann-führe-aus
```

Bedingte Ausführung "IF ELSE"

```
temperatur = get_sensor_data()
if temperatur > 25:
   print("Es ist warm")
else:
   print("Es ist kühl")
```

"Wenn Temperatur größer 25 ist, sag es ist warm, ANSONSTEN sag es ist kühl" **otherwise**

Vorschau Kontrollstrukturen

Kontrollstrukturen und Funktionen haben Programmcode in einem sogenannten Block

```
temperatur = 22
if temperatur > 25:
   print("Es ist warm")           # Block Befehl 1
   print("Es ist wirklich warm")  # Block Befehl 2
```

Dieser wird durch **Einrückung** von anderem Code abgegrenzt

```
temperatur = 22
if temperatur > 25:
   print("Es ist warm")           # Block Befehl 1
   print("Es ist wirklich warm")  # Block Befehl 2
print("nicht Teil des if Befehles")
```

Space oder Tab \t **Einrückung**

Funktionen Typensicherheit

Typensicherheit

```
from typeguard import typechecked

@typechecked
def test_int(value: int) -> int:
  return value

a = test_int(5)  # OK
b = test_int('5')  # TypeError
```

Variablen Typensicherheit

```
Geht in Python 3.12 leider noch nicht automatisch

man kann dynamisch prüfen

def typecheck(varible,typ):
  #  isinstance(varible,typ) ...
```

```
from typeguard import check_type

a:int = 5  # Initial assignment with an integer
check_type('a', a, int)  # Checks if 'a' is of type int
```

```
ODER per IDE plugin / console Befehl
```

Gültigkeitsbereich von Variablen

```
Kleine Vorschau, gehen wir später im Detail ein

global_var = 10 # Globale Variable, überall im Code gültig

# Funktionen sind Code-Blöcke, die wiederverwendet werden können
# Variablen können auch in Funktionen definiert werden,
# sind (in der Regel) dann aber nur in dieser Funktion gültig
def my_function():
  local_var = 5  # Lokale Variable, nur in dieser Funktion gültig
  print(global_var)  # Ausgabe: 10

if True:
 x = 8


print(local_var)  # ⚠️ Fehler: NameError: name 'local_var' is not defined
```

Tests

```
def f(x):
    return 2*x

def teste_die_f_funktion():
  assert f(3) == 6
  assert f(4) == 8
  # assert f(4) == 9
```

Operatoren Vorschau

Taschenrechner Nachtrag

Mathematisch: + - / * // % (modulo rest)

💡 Für Potenzrechnung "hoch Rechnung" muss man ** verwenden und nicht wie vielleicht erwartet `^`
(später: ^ sind bit Operatoren)

Vergleich: == > < <= >=

⚠️ Vorsicht, NIE Vergleich == mit Zuweisung = verwechseln!

```
>>> x = 1
>>> y = 2
>>> x == y
False / True  "Boolean" Wahrheitswert
>>> x = y
>>> x == y
True
```

`nil` ist null aus anderen Sprachen

gibt keine Pointer

Logisch: and, or, not, ^ (xor)

*= /= %= += -= # in place arithmetic / compound arithmetic zB compound addition

Operatoren Ausdruck

Die Anwendung von Operatoren sind ein Beispiel für einen sogenannten Ausdruck (expression)

"Alles was bei einer Variablen Zuweisung rechts stehen kann"

```
x = 3 * 3
x = print('hallo')
```

Ein Ausdruck welcher nicht ausgegeben oder zugewiesen wird macht normalerweise wenig Sinn!

```
len('hallo') # ⚠️ use it!?
print(len('hallo')) # ok, verwendet
laenge = len('hallo') # ok, verwendet (später…)
```

Operatoren

Taschenrechner Nachtrag

Mathematisch: + - / * % (modulo rest) ** (hoch, potenz, ^, ^^)
werden abgebildet auf Funktionen mit speziellem Namen
__add__

Vergleich: == , >, <
__eq__ __gt__ __lt__ equal greater / less than

⚠️ **Vorsicht, NIE Vergleich == mit Zuweisung = verwechseln!**

```
>>> x = 1
>>> y = 2
>>> x == y
False
>>> x = y
>>> x == y
True
```

Logisch: and, or, not, ^ (xor)

IDE

PyCharm:
Wichtigster Keyboard-shortcut: Strg + Shift + A : Search everywhere
Wichtigster Keyboard-shortcut: Strg + Shift Voreingestellt: SUCHE

Wichtigster Keyboard-shortcut : VS-Code: Strg + Shift + P
Select Interpreter: Python …

```
print("Hallo 🌍")
```

Vorschau Funktionen vs Methoden

Führen Aktionen aus / holen Informationen
Rückgabe **Typ** kann unterschiedlich sein

Funktionen
stehen unabhängig von Objekten
print()
input() # str
help()
dir(x) **#** listet alle Methoden vom Objekt x
type(x) # liefert Typ von x
len(x) # int

Methoden
hängen an Objekten (Dinge mit Eigenschaften), werden mit '.' Punkt aufgerufen, weiter mit **()**
z.B.
"HELLO"**.**lower() => "hello"

Kommentare Teil 2

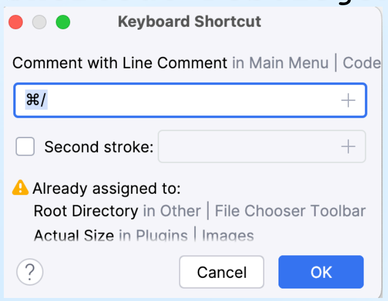
```
# Dies ist der Anfang eines mehrzeiligen Kommentars
# Jede Zeile wird mit einem # Zeichen begonnen
# Das endet hier


"""
Dies ist ein mehrzeiliger String,
der als Kommentar verwendet werden kann.
Python wird diesen String ignorieren, da er keiner Variablen zugewiesen ist.
"""

" ein Kommentar als String der nicht ausgegeben wird"
```

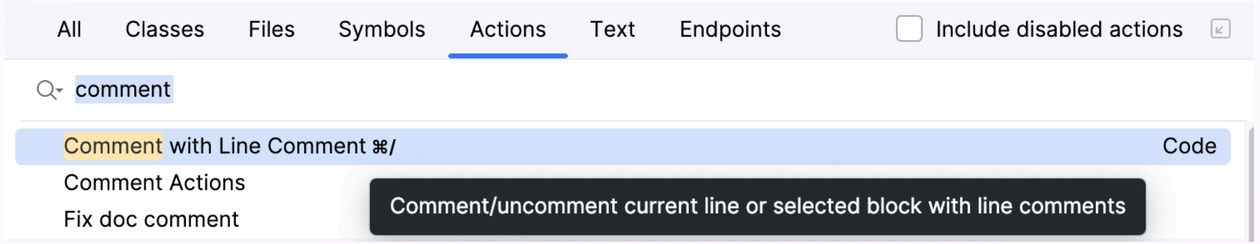
In PyCharm:

Strg + Shift + A : Search everywhere

Suche: comment

Shortcut Festlegen mit **Alt+Enter**





String Aufgaben

- leicht
- a) Schreibe Programm welches nach Namen Fragt und Länge ausgibt

mittel
- b) gebt das erste Zeichen aus ^^
- b1) gib Namen GROSS aus
- b2) prüfe ob der Name ein "a" enthält

schwerer
- c) gebt das letzte Zeichen aus

extra
- d) lies aktuelles Jahr ein und gib nächstes Jahr aus

- e) gib aus ob ein Name mit 'B' beginnt

Aufgaben Tag 2

- **prüfe ob eine Zahl gerade ist**
- **Berechne 100 Milliarden mal 100 Milliarden Ohne eine Null zu verwenden.**
- **Formatiere Name und Vorname und Titel => mit f"{title} …" => "Herr Ferdinant Mustermann"**
- **Berechnung des Alters anhand des Geburtsjahres**
- **Umrechnung von Temperaturen C/F:**
- **Schreibe ein Programm, welches die eine Temperatur in Celsius entgegennimmt und diese in Fahrenheit umrechnet.**
- **Umwandlung von Minuten in Stunden und Minuten**
- **formatiere Millisekunden als Tag (ohne und mit datetime)**

Extra (Für Überflieger, die vorzeitig fertig sind oder zu viel Energie haben:)

- **prüfe ob ein Jahr ein Schaltjahr ist**
- **(noch nicht lösbar)**
- **Schreibe eine Funktion, die eine Anzahl von Minuten entgegennimmt und diese in Stunden und Minuten umwandelt.**
- **Trivialer Taschenrechner mit Funktionen**
- **Schreibe eine Python-Anwendung, die Grundoperationen (Addition, Subtraktion, Multiplikation, Division) über Funktionen ermöglicht.**
- **def addiere(x,y): return x+y**
- **print(dividiere(10, 5))**

Extra Aufgaben Tag 2

⚠️ für Denkaufgaben (erstmal) kein GPT benutzen, nur für technische Fragen

Extra Aufgaben (noch nicht lösbar)

- Schreibe Programm welches den Namen (o.ä.) umdreht Lars => sraL (via for loop)
- Schreibe Programm welches vom Namen (o.ä.) jedes a in e (…) umwandelt (if letter == 'a': …)
- Schreibe Programm welches vom Namen (o.ä.) die Vokale (aeiou) zählt
- Schreibe Programm welches im Namen (o.ä.) Leerzeichen einfügt: "Lars" => "L a r s"

```
Tipp: man kann das Ergebnis stückweise konstruieren:
ergebnis = ""
if …:
  ergebnis = ergebnis + …
```

- **Palindrom-Prüfung**
  - **prüfe, ob ein gegebener String ein Palindrom ist (ein Wort, das vorwärts und rückwärts gelesen gleich bleibt)**
  - **z.B. abacaba**

Vokabular Tag 2

Variablen
Deklaration / Initiation (Zuweisung eines Wertes mit '=')
Vergleich (==)
Keywords / Schlüsselworte

Datentypen
int
float
str
bool (True,False)

Klasse (Typ)
Objekt, Instanz

type(…) Funktion
help(…) Funktion
dir(…) Funktion
Funktion() / .Methode()

>>> ctrl+l zum reinigen der Console (geht auch in System Console)