# Von C zu Python

>> Python ist eine **High-Level-Sprache**, die einfacher zu verwenden ist als die Sprache C.

Python ermöglicht die Nutzung verschiedener Programmierparadigmen wie **objektorientiertes** und **funktionales** Programmieren. In späteren Einheiten werden wir uns diese anhand praktischer Beispiele ansehen.

Dazu müssen wir aber erst die Syntax von Python kennen und verstehen, wie in Python programmiert wird.

## Von C zu Python

– C (1972) vs. Python (1991)

– Deutlich einfachere Syntax, mächtigere Konstrukte.

– Python verwaltet den Speicher automatisch.

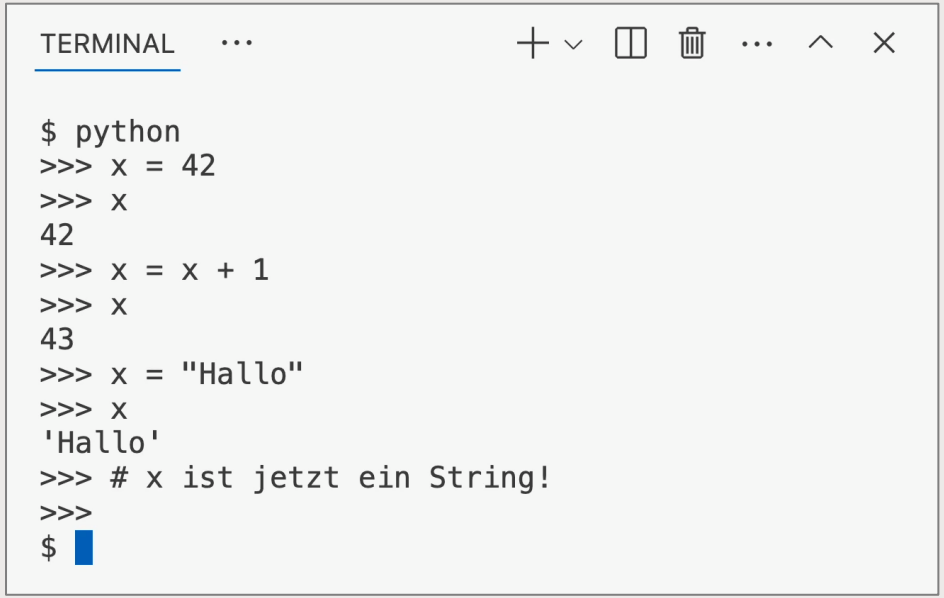
– Python-Code muss nicht explizit kompiliert werden.

```c
// C: Speicher manuell verwalten
int numbers[100];  // Feste Größe
char name[20];     // Fester Speicher
```

```python
# Python: Fokus auf Funktionalität
numbers = [42, 17, 23]  # Flexible Liste
name = "Alice"          # Einfacher Text
```

## Der Python Interpreter

- Interaktiver Modus für schnelles Experimentieren
- Direktes Feedback
- Ideal zum Lernen und Testen



## Syntax: Grundlagen

– Keine Typ-Deklarationen

– Keine Semikolons

– Einrückung statt geschweifter Klammern

```c
// C-Style
if (x < 0) {
    printf("negativ\n");
}
printf("fertig\n");
```

```python
# Python-Style
if x < 0:
    print("negativ")  # Einrückung wichtig!
print("fertig")
```

## Kommentare und Strings

– Einzeilige und mehrzeilige Kommentare

– Flexible String-Notation

```
# Einzeiliger Kommentar
# Jede Zeile braucht ein #

"""
Mehrzeiliger Kommentar
mit dreifachen Anführungszeichen
"""

name = "Alice"   # Doppelte Anführungszeichen
name = 'Alice'   # Einfache gehen auch

text = "It's a nice day"    # Praktisch für Apostrophe
text = 'Say "hello"'        # Oder für Zitate
```

## Typumwandlungen

– Explizite Konvertierung mit `str()`, `int()`, `float()`

– Nicht alle Umwandlungen sind möglich

```
# Zahlen zu Text
alter = 42
alter_text = str(alter)      # "42"

# Text zu Zahlen
preis = "19.99"
preis_num = float(preis)    # 19.99

# Vorsicht bei Umwandlungen!
text = "hallo"
# zahl = int(text)  # Würde fehlschlagen!
```

**PEP 8 für guten Stil** (Python Enhancement Proposal 8)

– Einrückungen mit 4 Leerzeichen (keine Tabs).

– Leerzeichen vor und nach Operatoren (z. B. `a = b + c`).

– Wähle aussagekräftige Namen (z. B. `anzahl_schleifen` statt `x`).

```
# Gut:
student_name = "Max"
durchschnitt = (note1 + note2) / 2

# Weniger gut:
x="Max"
y=(a+b)/2
```

Automatische Formatierung z.B. mit *autopep8:*

```
autopep8 --in-place meine_datei.py
```

**PEP 8 für guten Stil** (Python Enhancement Proposal 8)

– Einrückungen mit 4 Leerzeichen (keine Tabs).

– Leerzeichen vor und nach Operatoren (z. B. `a = b + c`).

– Wähle aussagekräftige Namen (z. B. `anzahl_schleifen` statt `x`).

– **Maximale Zeilenlänge: 79 Zeichen**

```
# Nicht so gut:
long_string = "Das ist ein sehr langer String, der weit über die empfohlene Zeilenlänge
hinausgeht"


# Besser:
long_string = (
    "Das ist ein sehr langer String, "
    "der weit über die empfohlene "
    "Zeilenlänge hinausgeht"
)
```

## Variablen und Datentypen

– C: statische Typisierung; Python: dynamisch zur Laufzeit

– Variablen können zur Laufzeit ihren Typ ändern

– Werte haben weiterhin feste Typen

```
x = 42           # Zahl
x = "Hallo"      # Jetzt Text - kein Problem!

# Typkompatibilität beachten
text = "Hallo"   # String
zahl = 42        # Integer
pi = 3.14        # Float

# Nur kompatible Typen kombinieren
ergebnis = "Hallo" + " Welt"    # Geht
# falsch = "Hallo" + 42         # Geht nicht!
```

## Ein- und Ausgabe

– `print()` statt `printf()`

– Hängt automatisch Zeilenumbruch `\n` an

– Eingabe: get_string aus cs50-Modul (oder `input`)

```
# Ausgabe
print("Hallo")            # Mit Zeilenumbruch
print("Hallo", "Welt")    # Automatisch getrennt


# Eingabe mit CS50-Library
from cs50 import get_string, get_int

name = get_string("Name: ")    # Wie in C!
alter = get_int("Alter: ")     # Gibt Integer zurück
```

## Module importieren

– Ähnlich wie #include in C

– Flexibel: ganzes Modul oder nur Teile

```
# Python Varianten:
import math                    # Ganzes Modul
from random import randint     # Einzelne Funktion
from cs50 import get_int       # CS50-Funktion


# Verwendung:

# Modul.Element (da ganzes Modul importiert)
print(math.pi)

# Direkt nutzbar (da einzelne Funktion importiert)
zahl = randint(1, 6)
```

## String-Formatierung

– **f-Strings:** bevorzugte Methode – geschweifte Klammern im String

– **.format():** Alternative mit benannten oder positionierten Platzhaltern.

– **Formatierungsoptionen:** z. B. Festlegen von Nachkommastellen ({:.2f}).

```
name = "Alice"
alter = 20

print(f"Hallo, {name}! Alter: {alter}")

print("Hallo, {}! Alter: {}".format(name, alter))
print("Hallo, {n}! Alter: {a}".format(n=name, a=age))

zahl = 42.12345
print(f"Zahl: {zahl:.2f}")         # Zwei Nachkommastellen
print("Zahl: {:.2f}".format(zahl)) # Gleiche Funktionalität

print(f"Name: {name.upper()}, halbes Alter: {alter/2}")
# => Name: ALICE, halbes Alter: 10.0
```

## Python vs. C: Ausführungsmodell

– **C:** Kompilierung → ausführbare Datei (Executable)

– **Python:** Interpretation des Quellcodes durch den Interpreter

– Manche Fehler können daher erst zur Laufzeit bemerkt werden.

– Python erstellt intern *Bytecode*, aber das ist für Entwickler nicht sichtbar.

```
from cs50 import get_string

def greet(name):
    print("Hallo " + name + extra)  # extra nicht definiert!

user_input = get_string("Name: ")
greet(user_input)

# Programm wird vom Interpreter ohne Kommentar ausgeführt,
# bricht aber dann beim Aufruf der Funktion mit Fehler ab.
```

## Arithmetische Operatoren und einfache String-Operationen

– Arithmetische Operatoren wie in C, aber Abweichungen (z.B. Integer-Division)

– Grundlegende String-Operationen

```
# Arithmetik
x = 10 + 5    # Addition
y = 10 - 5    # Subtraktion
z = 10 * 5    # Multiplikation
w = 10 / 5    # Division (ergibt Float)

# Anders als in C
x += 1        # Kein x++ Operator
y = 10 // 3   # Ganzzahldivision (ergibt 3)
z = 10 % 3    # Modulo (ergibt 1)

# String-Operationen
text = "Hallo"
print(len(text))              # Länge: 5
nachricht = text + " Welt"    # Konkatenation
```

# EXTRAS IN 3 MINUTEN

FRAGEN – ANTWORTEN – RÄTSEL
UND KURZE ZUSAMMENFASSUNG

## Aufgabe 1

Schreiben Sie ein Python-Programm, das den Benutzer nach seinem Namen fragt und dann *Hallo Name!* ausgibt. Nutzen Sie "get_string" und f-Strings.

```
# Ihre Lösung?
```

## Aufgabe 1

Schreiben Sie ein Python-Programm, das den Benutzer nach seinem Namen fragt und dann *Hallo Name!* ausgibt. Nutzen Sie "get_string" und f-Strings.

```
from cs50 import get_string

name = get_string("Name: ")
print(f"Hallo {name}!")
```

## Aufgabe 2

Wandeln Sie den String "42.5" in eine Fließkommazahl um und multiplizieren Sie diese mit 2. Geben Sie das Ergebnis mit genau einer Nachkommastelle aus.

```
# Ihre Lösung?
```

## Aufgabe 2

Wandeln Sie den String "42.5" in eine Fließkommazahl um und multiplizieren Sie diese mit 2. Geben Sie das Ergebnis mit genau einer Nachkommastelle aus.

```
text = "42.5”

zahl = float(text)

ergebnis = zahl * 2
print(f"Ergebnis: {ergebnis:.1f}")
```

## Aufgabe 3

Berechnen Sie die Fläche eines Kreises! Der Radius soll als Eingabe abgefragt werden. Importieren Sie *pi* aus dem *math*-Modul und geben Sie die Fläche mit zwei Nachkommastellen aus.

```
# Ihre Lösung?
```

## Aufgabe 3

Berechnen Sie die Fläche eines Kreises! Der Radius soll als Eingabe abgefragt werden. Importieren Sie *pi* aus dem *math*-Modul und geben Sie die Fläche mit zwei Nachkommastellen aus.

```
from cs50 import get_int
from math import pi


radius = get_int("Radius: ")
flaeche = pi * radius * radius
print(f"Die Kreisfläche beträgt {flaeche:.2f}.")
```

## Aufgabe 4

Was gibt dieser Code aus?

```
x = 5
x = x / 2
x = x * 2
print(f"{x}")
```

Direkte Ausführung ohne explizites Kompilieren.

Python-Interpreter auf der Kommandozeile zum Experimentieren.

Ein-/Ausgabe

print() und f-Strings

get_string() und get_int() aus CS50

Vereinfachte Syntax:

Keine Typen

keine Semikolons

Einrückung statt Klammern

Modulsystem

import statt #include

Teilimporte möglich

Dynamische Typisierung ermöglicht flexiblen Typ-Wechsel zur Laufzeit

Explizite Konvertierung mit str(), int(), float(), …

PEP 8: Stil-Richtlinien

Einheitliche Darstellung

autopep8 zur automatischen Formatierung von Code