FHV
Vorarlberg University
of Applied Sciences

Einführung Programmierung

Nico Mangeng

Rep Programmiertechniken
Wintersemester 2024

Was versteht man unter Programmierung?

„Die Programmierung bezeichnet die Tätigkeit, Computerprogramme zu erstellen. Dies ist ein Teilbereich der Softwareentwicklung. Computerprogramme werden mit Hilfe einer Programmiersprache formuliert. Der Programmierer übersetzt dabei die vorgegebenen Anforderungen und Algorithmen in eine gewünschte Programmiersprache.“

- Geklaut von Wikipedia

Funktionsweise eines Codes

- Programmcode läuft sequentiell ab
- Nutzt Kontrollstrukturen zur Steuerung des Programmflusses
- Beinhaltet Variablen, Funktionen, opt. externe Bibliotheken, ….

Schreiben von Code:
Lexikalik, Syntax und Semmantik

- Lexikalik: „Bekanntheit“ der Zeichen im Code
- Syntax: „Rechtschreibung“ des Codes
- Semantik: „Grammatik“ des Codes

Lexikalik, Syntax und Semmantik

- Lexikalik: A-Z, 0-9, !?., …

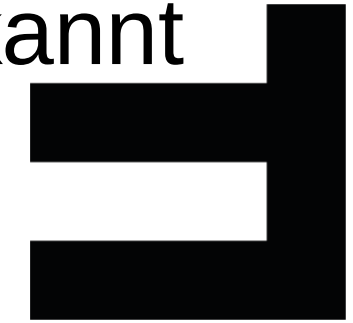
Kennen wir die verwendeten Zeichen?

- Syntax: Worte, sind, uns, diese, allen, bekannt

Sind die Worte richtig geschrieben?

- Semantik: Diese Worte sind uns allen bekannt

Macht der Einsatz der Worte Sinn?



Lexikalik, Syntax und Semmantik

- Lexikalik: Wir lernen **Py1h0n**

Unbekannte Zeichen werden verwendet!

- Syntax: **Wiehr** lernen **Paithon**

Die Worte sind falsch geschrieben!

- Semantik: Wir lernen **Einkaufswagen**

Die Sinnhaftigkeit ist nicht gegeben!

Lexikalik, Syntax und Semmantik

- Lexikalik:

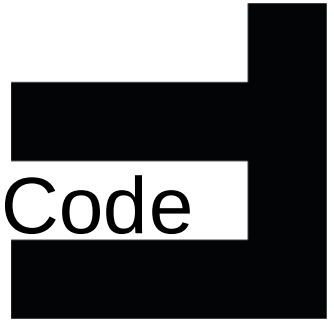
Verwendung von Zeichen die dem Code bekannt sind:
Kontrollstrukturen (if-else, while), Datentypen (int, float), …

- Syntax:

Werden keine Programmierregeln verletzt:
Keine Klammern vergessen, Funktionswerte übergeben, …

- Semantik:

Macht der geschriebene Code Sinn:
Keine Endlosschleifen, Divisionen durch 0, unerreichbarer Code



Lexikalik, Syntax und Semmantik

- Lexikalik:

```
switch(variable):
        case 1: print(„Switch case?“)
        case 2: print(„What is that?“)
        default: print(„Never heared of…“)
```

Python kennt Switch-Case so nicht!
Update: Mit Version 3.10 → match/case

- Semantik:

```
while(True):
        print(„This is a never ending story!“)
```

Logikfehler, der Code läuft so ewig!

- Syntax:

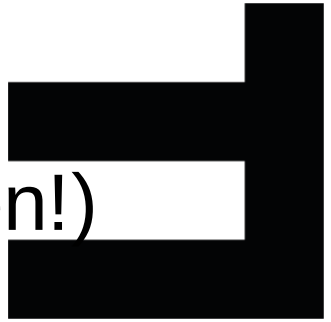
```
var1 = 25
var2 = 15
var3 = 17
print(((var1 + var2) – 4 * var1 + (1 / var3))
```

Es fehlt eine Klammer!
Typischer Syntaxfehler

Variablen: Grundlegendes

- Bestimmter Bereich im Speicher
- Müssen eindeutig benannt werden
- Können beliebige Werte annehmen
- Sind veränderbar → variabel
- Sollten immer initialisiert werden!!!
- Unterscheidung Deklaration/Definition

  (In Python nicht so wichtig, sollte aber verstanden werden!)



Variablen: Deklaration & Definition in C

- Deklaration: Was für ein Datentyp?
  - Erstellung einer Variablen
  - Speicherplatz wird nicht reserviert
  - **Ohne** Zuweisung eines Wertes
  - Wird oft bei Funktionen verwendet
- Definition: Was für ein Inhalt?
  - Erstellung einer Variablen
  - Speicherplatz wird reserviert
  - **Mit** Zuweisung eines Wertes
  - Inhalt der Variable = vom Programmierer zugewiesener Wert

```c
int main() {
    int i;          // Declaration
    int j = 10;     // Definition
    printf(i+j);  // Error: variable "i" not initialised!
}
```

Variablen: Deklaration & Definition in Python

- Definition:
  - Erstellung einer Variable
  - Speicherplatz wird reserviert
  - **Mit** Zuweisung eines Wertes
  - Es wird kein Datentyp angegeben
  - Funktionen können ebenfalls als Variable gesehen werden!

Es nicht möglich eine Deklaration für eine Variable zu erstellen! Variablen werden bei der ersten Zuweisung eines Wertes erstellet!

```
very_varying_variable = "I am a string"
print(type(very_varying_variable))
very_varying_variable = 12345
print(type(very_varying_variable))
very_varying_variable = ["List", "of", "strings"]
print(type(very_varying_variable))
very_varying_variable = 3.1415
print(type(very_varying_variable))
very_varying_variable = "Back to string"
print(type(very_varying_variable))
```

```
Output:

<class 'str’>
<class 'int’>
<class 'list’>
<class 'float’>
<class 'str’>
```

Datentypen: Grundlagen

- Zahlenwerte:
  - int Ganzzahl 1, 2, 3, 400, 123456789
  - float Dezimalzahl 3.1415, 1.2, 0.99, 17e-2
- Zeichenwerte:
  - char Zeichen A-Z, a-z, 0-9, !?ß, …
  - str String/Zeichenkette „Hallo“, „Baum“, „Sieben“, …
- Wahrheitswerte:
  - bool Boolean True, False (Erste Buchstaben groß!)
- Listen:
  - div.* Liste/Array [1,2,3,4,5], [„Eins“, „Zwei“, „Drei“], …
- Tuple:
  - div.* Unveränderbare Liste (1,2,3,4,5), („Eins“, „Zwei“, „Drei“), …

* Datentyp Abhängig vom Inhalt der Liste/ des Tuples

Datentypen: Spezielle Datentypen

- Dictionaries (kurz dicts)
- Pandas Dataframe
- Numpy-Arrays

- Spezielle Datentypen, welche fortgeschritten sind. Werden in späteren Reps genauer behandelt.

Datentypen:

- Abfrage eines Datentyps: type()

```
name = "Max Mustermann"
age = 25
weight = 80.3

print(type(name), type(age), type(weight))
print(type(name) == str)
print(type(age) == str)
print(isinstance(name, str))
print(isinstance(age, str))

<class 'str'> <class 'int'> <class 'float’>
True
False
True
False
```

Datentypen:

- Konvertierung eines Datentyps: explizit/implizit

```
name = "Max Mustermann"
age = 25
weight = 80.3

# Explicit conversion. Done by yourself!
weight = int(weight)
print(weight, type(weight))
# Implicit conversion. Done by Python!
age = age * 1.0
print(age, type(age))

Output:

80 <class 'int’>
25.0 <class 'float'>
```

```
# Implicit casting by Python
var1 = 10   # int
var2 = 7    # int
result = 0  # int

result = var1 / var2 # cast to float
print(result, type(result))

Output:

1.4285714285714286 <class 'float'>
```

Kontrollstrukturen

- Was sind Kontrollstrukturen?
  - Steuern den Ablauf des Programmes
  - Stellen Logik für den Code bereit
  - Ermöglichen komplexe Abläufe zu realisieren
- Welche Kontrollstrukturen stehen zur Verfügung?
  - Verzweigungen: if, if-else, match-case (ab Python Version 3.10)
  - Schleifen: for, while, do-while
  - Try-Except (Fehlerbehandlung)

do-while Schleifen sind in Python nicht standardmäßig enthalten, können aber selbst relativ einfach gebastelt werden

Kontrollstrukturen: if, if-else, elif

- Verzweigungen sind Fallunterscheidungen:
  - Wenn Bedingung erfüllt ist mache was
  - Wenn Bedingung erfüllt ist mach das, sonst jenes
  - Wenn Bedingung 1 und Bedingung 2 erfüllt ist, mache was
  - Wenn Bedingung 1 oder Bedingung 2 erfüllt ist, mache was
  - …
- Jede Verzweigung basiert auf einem logischen True oder False

Kontrollstrukturen: if, if-else, elif

```
temperature = 95.3

if (temperature < 100):       # Simple if-statement
    print("Temperature ok")

Output:
Temperature ok
```

```
temperature = 105.4

if (temperature < 100):       # if-else statement
    print("Temperature ok")
else:
    print("Temperature critical!")

Output:
Temperature critical!
```

Kontrollstrukturen: if, if-else, elif

```
temperature = 105.4
cooling_on = True

# elif-statement with multiple conditions
if (temperature > 100 and cooling_on == True):
    print("Temperature critical! Cooling active")
elif (temperature > 100 and cooling_on == False):
    print("Temperature critical! Cooling not active")
elif (temperature < 100):
    print("Temperature ok")
    cooling_on = False

Output:
Temperature critical! Cooling active
```

Kontrollstrukturen: for Schleife

- Meist verwendet, um durch Listen zu iterieren:

```
my_list = ["Every", "dog", "has", "its", "day"]

for name_me_as_you_want in my_list:
    print("List content is: ", name_me_as_you_want)

Output:
List content is: Every
List content is: dog
List content is: has
List content is: its
List content is: day
```

Kontrollstrukturen: for Schleife

```python
int_list = [1,5,7,34,5,7,4,3,235,7,78]
counter_odd = 0
counter_even = 0

for number in int_list:
    # % - modulo: Returns the rest of the division
    # 7 % 4: 7/4 = 1 with 3 rest --> returns 3
    # 9 % 3: 9/3 = 3 with 0 rest --> returns 0
    # Everything else as 0 is seen as True in if
    if number % 2:
        counter_odd += 1
    else:
        counter_even += 1

print("Even numbers found: ", counter_even)
print("Odd  numbers found: ", counter_odd)

Output:
Even numbers found: 3
Odd numbers found: 8
```

```python
alphabet_list = [
    "a", "b", "c", "d", "e", "f", "g",
    "h", "i", "j", "k", "l", "m", "n",
    "o", "p", "p", "q", "r", "s", "t",
    "u", "v", "w", "x", "y", "z"
]

for idx in range(0, 25, 2): # range:
startindex, stopindex, index steps
    print(alphabet_list[idx])

Output:
a c e g i k m o p r t v x
```

Kontrollstrukturen: break

```
counter_variable = 1
max_iteration = 10
break_condition = 11

while (counter_variable <= max_iteration):
    print("We are at iteration: ", counting_variable )
    counter_variable += 1
    if (counter_variable == break_condition):
        print("Stopped at iteration", counter_variable, "because of break condition!")
        break

# Syntax error --> Code doesn´t work, why?
# Logical error --> "Stopped at iteration 11",
# but we don’t want more than 10 iterations???
```

Kontrollstrukturen: break & continue

```
counter_variable = 1
max_iteration = 10
continue_condition = 5

while (counter_variable <= max_iteration):
    if (counter_variable == continue_condition):
        print("I really HATE the number:", counter_variable)
        continue
    print("I really like the number:", counter_variable )
    counter_variable += 1

# Why does the code not work???
```

Einstieg in die Übungen

Übung: Vielfaches finden

- Erstellt eine Liste mit Ganzzahlen
- Definiert eine Variable, welche als Divisor verwendet wird
- Iteriert durch die Liste und findet heraus welche Ganzzahl ein Vielfaches des Divisors ist.
- Schreibt auf die Konsole ob die aktuelle Zahl der Liste ein Vielfaches ist, oder nicht.

- Hilfreiche Funktionen:
  - range(0, 10, 1) → Erstellt eine Liste beginnend bei 0 bis 10 mit Schritten von 1 [0, 1, … 9, 10]
  - print() → Ausgabe auf die Konsole
  - Modulo Operator (%) 8 % 3 → 2 Rest, 8 % 4 → 0 Rest

Übung: Würfelsimulator

- Gebt an, wie oft ihr spielen wollt und schreibt es in eine Variable
- Euer Tipp soll in einer Variable gespeichert werden
- Schreibt eine zufällig generierte Ganzzahl im Bereich von 1 – 6 in eine Variable
- Überprüft, ob ihr richtig liegt oder nicht und schreibt es auf die Konsole
- Zusätzlich soll in eine Liste „Win“ bzw. „Lose“ geschrieben werden
- Gebt nach allen Spielzügen das Ergebnis der Runden in der Konsole aus

- Hilfreiche Funktionen:
  - randint(1,10) → Gibt eine Ganzzahl von 1 bis 10 aus
  - .append() → Hängt einen Wert an eine (leere) Liste an # my_list.append(„Wert“)
  - print() → Ausgabe auf die Konsole.
  - input(() → Variable von außen ins Jupyter Notebook übergeben

Happy Coding!