Programmieren mit Python

Teil 2: Conditionals und Schleifen

Dr. Aaron Kunert
aaron.kunert@salemkolleg.de

22. April 2021

Script Mode

Script Mode

Sobald man mehrere zusammenhängende Zeilen hat, wird die REPL sehr unübersichtlich. Daher gibt es auch die Möglichkeit, alle Programmzeilen zunächst aufzuschreiben und diese dann gebündelt von Python ausführen zu lassen. Im Gegensatz zum REPL bzw. interactive Mode von Python wird dies *Script Mode* genannt.

Beispiel

```
1 name = "Max"
2 age = 20
3 print(f"Hello, I'm {name} and I'm {age} years old")
```

Beispiel

```
1 name = "Max"
2 age = 20
3 print(f"Hello, I'm {name} and I'm {age} years old")
```

Problem:

Wie kann man Python erklären, diese 3 Zeilen auf einmal auszuführen?

Old-School-Lösung

- Erstelle eine neue Datei (z.B. `my_script.py`)
- Öffne die Datei mit einem Texteditor und speichere den Beispiel-Code darin ab.
- Öffne den Ordner mit der Datei `my_script.by` mit dem Terminal bzw. der Eingabeaufforderung
- Führe das Kommando `python my_script.py` aus.

Optimallösung: Verwende eine IDE

Eine IDE (integrierte Entwicklungsumgebung) hilft Dir beim Programmieren und unterstüzt Dich wo immer möglich. Dadurch lassen sich auch große Projekte schnell umsetzen.

Optimallösung: Verwende eine IDE

Eine IDE (integrierte Entwicklungsumgebung) hilft Dir beim Programmieren und unterstüzt Dich wo immer möglich. Dadurch lassen sich auch große Projekte schnell umsetzen.

Nachteile

Die anfängliche Einrichtung kann schnell kompliziert werden. Aufgrund der vielen Features fühlt man sich schnell mal überfordert.

Optimallösung: Verwende eine IDE

Eine IDE (integrierte Entwicklungsumgebung) hilft Dir beim Programmieren und unterstüzt Dich wo immer möglich. Dadurch lassen sich auch große Projekte schnell umsetzen.

Nachteile

Die anfängliche Einrichtung kann schnell kompliziert werden. Aufgrund der vielen Features fühlt man sich schnell mal überfordert.

→ Das machen wir etwas später.

Kompromiss für den Anfang: Browserbasierte Editoren

Um schnell einzusteigen, kann zu Beginn auch ein browsergestützter Editor/Interpreter verwendet werden. Zum Beispiel:

Kompromiss für den Anfang: Browserbasierte Editoren

Um schnell einzusteigen, kann zu Beginn auch ein browsergestützter Editor/Interpreter verwendet werden. Zum Beispiel:

- Programiz (`https://www.programiz.com/python-programming/online-compiler`)
  - einfacher Einstieg
  - schnell und unkompliziert
  - geringer Funktionsumfang

Kompromiss für den Anfang: Browserbasierte Editoren

Um schnell einzusteigen, kann zu Beginn auch ein browsergestützter Editor/Interpreter verwendet werden. Zum Beispiel:

- Programiz (`https://www.programiz.com/python-programming/online-compiler`)
  - einfacher Einstieg
  - schnell und unkompliziert
  - geringer Funktionsumfang
- Repl.it (`https://replit.com/languages/python3`)
  - Auch für viele andere Sprachen geeignet
  - Manchmal etwas langsam
  - Man kann mehrere Dateien und Projekte verwalten (braucht Account)
  - Hat fast alle IDE-Features (braucht Account)

Input/Output

Kommunikation über die Konsole

Die Konsole

Da wir zu Beginn noch über keine grafische Benutzeroberfläche verfügen, verwenden wir für die Kommunikation mit unserem Programm die *Konsole*. Dabei handelt es sich um ein einfaches Textfenster, auf dem Dein Programm Informationen ausgeben kann (*Output*) und Text einlesen kann (*Input*).

Output

Um einen String auf der *Konsole* auszugeben, verwende die Funktion `print()`.

Zum Beispiel: `print("Hello there")`.

Output

Um einen String auf der *Konsole* auszugeben, verwende die Funktion `print()`.

Zum Beispiel: `print("Hello there")`.

Es können auch Variablen eingesetzt werden:

```
1 message = "Hello there"
2 print(message) # Hello there
```

String Interpolation

Um Variablenwerte innerhalb eines Strings auszugeben, verwenden wir die String-Interpolation-Syntax:

```
my_value = 5
print(f"The variable my_value has the value {my_value}")
# The variable my_value has the value 5
```

String Interpolation

Um Variablenwerte innerhalb eines Strings auszugeben, verwenden wir die String-Interpolation-Syntax:

```
my_value = 5
print(f"The variable my_value has the value {my_value}")
# The variable my_value has the value 5
```

Das geht auch als *inline expression*:

```
print(f"The sum of 1 and 2 is {1+2}")
# The sum of 1 and 2 is 3
```

Input

Um einen String vom User einzulesen, verwende die Funktion `input()`:

```
1 age = input("How old are you?")
2 print(f"I am {age} years old")
```

Input

Um einen String vom User einzulesen, verwende die Funktion `input()`:

```
1 age = input("How old are you?")
2 print(f"I am {age} years old")
```

Achtung

Das Ergebnis von `input` hat stets den Datentyp `string` auch wenn Zahlen eingelesen werden. Gegebenenfalls muss das Ergebnis mittels `int()` oder `float()` in den gewünschten Typ umgewandelt werden.

Beispiel: Input und Output kombiniert

```
1 name = input("What is your name?")
2 age = input("What is your age?")
3 print(f"Hello {name}, you are {age} years old")
```

Übung

Adressabfrage

Schreibe ein kurzes Skript, dass Dich nach Deinem Namen, Alter und Adresse fragt. Wenn es alles eingelesen hat, soll es diese Infos in folgender Form auf der Konsole ausgeben:

```
Hallo Max, schön dass Du da bist. Du bist 21 Jahre alt und wohnst in der Bismarckstraße 12 in Glücksstadt.
```

Übung

Blick in die Zukunft

Schreibe ein kurzes Skript, dass Dich nach Deinem Alter fragt. Daraufhin soll es auf der Konsole ausgeben, wie alt Du in 15 Jahren sein wirst.

Kommentare

Kommentare

Alle Zeichen einer Zeile, die hinter einem # (Hashtag) kommen, werden von Python ignoriert. So lassen sich Kommentare im Quellcode platzieren.

Kommentare

Alle Zeichen einer Zeile, die hinter einem # (Hashtag) kommen, werden von Python ignoriert. So lassen sich Kommentare im Quellcode platzieren.

Beispiel

```
1 print("This line will be printed")
2 # print("This line won't")
```

Conditionals

Ein Programm verzweigen

Problemstellung

Lies eine Zahl `x` ein. In Abhängigkeit von `x` soll Folgendes ausgegeben werden:

```
Die Zahl x ist größer als 0
```

bzw.

```
Die Zahl x ist kleiner 0
```

Wie macht man das?

Lösung (fast)

```
x = input("Gib eine Zahl x an")
x = int(x)

if x < 0:
  print("x ist größer 0")
else:
  print("x ist kleiner 0")
```

Struktur `if-else` Statement

Struktur `if-else` Statement

`if`

Struktur `if-else` Statement

`if` *Bedingung*

Struktur `if-else` Statement

`if` *Bedingung* :

Struktur `if-else` Statement

`if` *Bedingung* :

␣␣

Struktur `if-else` Statement

`if` *Bedingung* :

␣␣ *Codezeile A1*

Struktur `if-else` Statement

`if` *Bedingung* :

␣␣ *Codezeile A1*

␣␣ *Codezeile A2*

␣␣ ⋮

Struktur `if-else` Statement

`if` *Bedingung* `:`

␣␣ *Codezeile A1*

␣␣ *Codezeile A2*

␣␣ ⋮

`else:`

Struktur `if-else` Statement

`if` *Bedingung*`:`

␣␣ *Codezeile A1*

␣␣ *Codezeile A2*

␣␣ ⋮

`else:`

␣␣ *Codezeile B1*

␣␣ *Codezeile B2*

␣␣ ⋮

Struktur `if-else` Statement

`if` *Bedingung*`:`

␣␣ *Codezeile A1*

␣␣ *Codezeile A2*

␣␣ ⋮

`else:`

␣␣ *Codezeile B1*

␣␣ *Codezeile B2*

␣␣ ⋮

Codezeile C1

⋮

Wie funktioniert's?

Ist die `if`-Bedingung `True`, so wird der `if`-*Block* ausgeführt. Ist sie `False` wird der `else`-*Block* ausgeführt.

Wie funktioniert's?

Ist die `if`-Bedingung `True`, so wird der `if`-*Block* ausgeführt. Ist sie `False` wird der `else`-*Block* ausgeführt.

Definition: Block

Aufeinanderfolgende Codezeilen, die alle die gleiche Einrückung besitzen, nennt man *Block*. D.h. Leerzeichen am Zeilenanfang haben in Python eine syntaktische Bedeutung.

Wie funktioniert's?

Ist die `if`-Bedingung `True`, so wird der `if`-*Block* ausgeführt. Ist sie `False` wird der `else`-*Block* ausgeführt.

Definition: Block

Aufeinanderfolgende Codezeilen, die alle die gleiche Einrückung besitzen, nennt man *Block*. D.h. Leerzeichen am Zeilenanfang haben in Python eine syntaktische Bedeutung.

Good to know

- Der `else`-Block ist optional.
- Falls die Bedingung nicht vom Typ `bool` ist, so wird sie implizit umgewandelt.

Volljährigkeit prüfen/Zutrittskontrolle

Schreibe ein Skript, dass nach dem Alter eines Users fragt und überprüft, ob der User schon volljährig ist. Dementsprechend soll auf der Konsole entweder

```
Willkommen
```

oder

```
Du darfst hier nicht rein
```

erscheinen.

Übungen

Volljährigkeit prüfen/Zutrittskontrolle

Schreibe ein Skript, dass nach dem Alter eines Users fragt und überprüft, ob der User schon volljährig ist. Dementsprechend soll auf der Konsole entweder

```
Willkommen
```

oder

```
Du darfst hier nicht rein
```

erscheinen.

Teilbarkeit bestimmen

Schreibe ein Skript, dass eine ganze Zahl einliest. Daraufhin soll auf der Konsole ausgegeben werden, ob die Zahl durch 7 teilbar ist. Beispiel: Ist die Eingabe 12, so ist die Ausgabe:

```
Die Zahl 12 ist nicht durch 7 teilbar.
```

Logische Operatoren

Booleans können mittels folgender Operatoren miteinander verknüpft werden:

Logische Operatoren

Booleans können mittels folgender Operatoren miteinander verknüpft werden:

`and`	Ist genau dann `True`, wenn beide Operanden `True` sind.
`or`	Ist genau dann `True`, wenn mindestens ein Operand `True` ist.
`not`	Kehrt den nachfolgenden Wahrheitswert um.

Logische Operatoren

Booleans können mittels folgender Operatoren miteinander verknüpft werden:

`and`	Ist genau dann `True`, wenn beide Operanden `True` sind.
`or`	Ist genau dann `True`, wenn mindestens ein Operand `True` ist.
`not`	Kehrt den nachfolgenden Wahrheitswert um.

Beispiel

- `2 > 0 and 3 > 4` ist `False`
- `1 > 0 or 6 > 1` ist `True`
- `not 2 < 1` ist `True`

Übung

Was ergeben die folgenden Ausdrücke?

- `not 2 < 3 and 4 < 7`
- `4 not == 8`
- `3 != 4 and not 4 == 8`
- `7 <= 7.0 and not 7 != 7.0`
- `7 > 5 or 4 < 5 and not 9 > 6`
- `not 3 < 6 > 8`
- `not 3`

Übung

Was ergeben die folgenden Ausdrücke?

- `not 2 < 3 and 4 < 7`
- `4 not == 8`
- `3 != 4 and not 4 == 8`
- `7 <= 7.0 and not 7 != 7.0`
- `7 > 5 or 4 < 5 and not 9 > 6`
- `not 3 < 6 > 8`
- `not 3`

Präzedenz beachten!

1. `==`, `!=`, `<=`, `<`, `>`, `>=`
2. `not`
3. `and`
4. `or`

Das `elif`-Statement

Mit der reinen `if-else`-Syntax können nur *binäre* Verzweigungen dargestellt werden. Um mehrer, gleichrangige Verzweigungsäste zu realisieren kann man das `elif`-Conditional verwenden.

Das `elif`-Statement

Mit der reinen `if-else`-Syntax können nur *binäre* Verzweigungen dargestellt werden. Um mehrer, gleichrangige Verzweigungsäste zu realisieren kann man das `elif`-Conditional verwenden.

Beispiel

```
if x < 0:
    print("x is < 0")
elif x == 0:
    print("x is 0")
elif x == 1:
    print("x is 1")
else:
    print("x is not negative but neither 0 nor 1")
```

Das `elif`-Statement

Mit der reinen `if-else`-Syntax können nur *binäre* Verzweigungen dargestellt werden. Um mehrer, gleichrangige Verzweigungsäste zu realisieren kann man das `elif`-Conditional verwenden.

Beispiel

```
1 if x < 0:
2     print("x is < 0")
3 elif x == 0:
4     print("x is 0")
5 elif x == 1:
6     print("x is 1")
7 else:
8     print("x is not negative but neither 0 nor 1")
```

Die Anzahl der `elif`-Blöcke ist beliebig. Der `else`-Block ist wie immer optional.

Worin unterscheiden sich die beiden Abschnitte?

Abschnitt 1:

```
if x % 2 == 0:
    # some Code here
if x % 3 == 0:
    # some Code here
else:
    # some Code here
```

Abschnitt 2:

```
if x % 2 == 0:
# some Code here
elif x % 3 == 0:
# some Code here
else:
# some Code here
```

Übung

Baue einen Bestätigungsdialog

Schreibe ein Skript was einen typischen Bestätigungsdialog simuliert. Zum Beispiel:

`Are you sure to continue? (y)es/(n)o.`

Mögliche Antworten sind `yes`, `no` bzw. `y`, `n`. Daraufhin soll auf der Konsole `confirmed` oder `aborted` erscheinen.

Komplexere Übung

Lies eine Zahl zwischen 1 und 9 ein und gib auf der Konsole *deinen nächsten Urlaubsort* aus.

Der *Ternary Operator*

Oftmals möchte man eine Variable in Abhängigkeit eines Wahrheitswertes definieren. Für diesen einfachen Fall, ist das `if-else`-Konstrukt sehr umständlich. Stattdessen kann man für die Kürze den *ternary operator* verwenden.

Der *Ternary Operator*

Oftmals möchte man eine Variable in Abhängigkeit eines Wahrheitswertes definieren. Für diesen einfachen Fall, ist das `if-else`-Konstrukt sehr umständlich. Stattdessen kann man für die Kürze den *ternary operator* verwenden.

Beispiel

```
1  if x < 0:
2    sign = "sign"
3  else:
4    sign = "positive"
```

Der *Ternary Operator*

Oftmals möchte man eine Variable in Abhängigkeit eines Wahrheitswertes definieren. Für diesen einfachen Fall, ist das `if-else`-Konstrukt sehr umständlich. Stattdessen kann man für die Kürze den *ternary operator* verwenden.

Beispiel

```
1 if x < 0:
2   sign = "sign"
3 else:
4   sign = "positive"
```

Stattdessen mit Ternary Operator

```
sign = "negative" if x < 0 else "positive"
```

Übung

Lies eine ganze Zahl ein und gib ihren Betrag auf der Konsole aus. Schaffst Du es, das Ganze mit weniger als 5 Zeilen Code zu programmieren?

Die For-Schleife

Einen Programmabschnitt x-mal ausführen

Problemstellung

Lies eine ganze Zahl `x` ein. Gib dann folgende Zeilen auf der Konsole aus

```
1
2
3
4
⋮
x
```

Wie macht man das?

Lösung

```
x = input("Enter a number")

for k in range(1, x + 1):
  print(k)
```

Struktur der `for...in` Schleife

Struktur der `for...in` Schleife

```
for
```

Struktur der `for...in` Schleife

`for` *Variable*

Struktur der `for...in` Schleife

```
for Variable in
```

Struktur der `for...in` Schleife

```
for Variable in range(min, max)
```

Struktur der `for...in` Schleife

```
for Variable in range(min, max):
```

Struktur der `for...in` Schleife

```
for Variable in range(min, max):
␣␣Codezeile 1
```

Struktur der `for...in` Schleife

```
for Variable in range(min, max):
␣␣ Codezeile 1
␣␣ Codezeile 2
```

Struktur der `for...in` Schleife

```
for Variable in range(min, max):
␣␣ Codezeile 1
␣␣ Codezeile 2
␣␣      ⋮
```

Struktur der `for...in` Schleife

```
for Variable in range(min, max):
␣␣ Codezeile 1
␣␣ Codezeile 2
␣␣      ⋮
Code, der nicht mehr Teil der Schleife ist
```

Struktur der `for...in` Schleife

```
for Variable in range(min, max):
␣␣ Codezeile 1
␣␣ Codezeile 2
␣␣      ⋮
Code, der nicht mehr Teil der Schleife ist
```

Wie funktioniert's?

Die Schleifenvariable wird zunächst gleich dem unteren Wert in `range` gesetzt. Dann wird der `for`-Block wiederholt ausgeführt. Bei jedem Durchgang wird die Schleifenvariable um `1` vergrößert und zwar so lange, wie der Wert der Schleifenvariable kleiner als der obere Wert in `range` ist.

Good to know

Good to know

- Achtung: Die Schleifenvariable erreicht nie das obere Ende der `range`-Funktion, sondern bleibt immer 1 drunter.

Good to know

- Achtung: Die Schleifenvariable erreicht nie das obere Ende der `range`-Funktion, sondern bleibt immer `1` drunter.
- Die `range`-Funktion ist nicht auf 1er-Schrittweite beschränkt. Mit folgendem Ausdruck werden die Zahlen von `0` bis `9` z.B. in 3er-Schritten durchlaufen: `range(0, 10, 3)`.

Good to know

- Achtung: Die Schleifenvariable erreicht nie das obere Ende der `range`-Funktion, sondern bleibt immer `1` drunter.
- Die `range`-Funktion ist nicht auf 1er-Schrittweite beschränkt. Mit folgendem Ausdruck werden die Zahlen von `0` bis `9` z.B. in 3er-Schritten durchlaufen: `range(0, 10, 3)`.
- `For`-Schleifen sind flexibel und können alles mögliche durchlaufen, z.B. auch die einzelnen Buchstaben eines Strings (dazu später mehr).

Übung

Einmaleins: Die 7er-Reihe

Schreibe ein kleines Skript, was die 7er-Reihe (bis 70) auf der Konsole ausgibt.

Übung

Einmaleins: Die 7er-Reihe

Schreibe ein kleines Skript, was die 7er-Reihe (bis 70) auf der Konsole ausgibt.

7er-Reihe mit beliebigem oberen Ende

Lies eine positive ganze Zahl `x` ein, gib die 7er-Reihe von 7 bis `x` auf der Konsole aus.

Übung

Einmaleins: Die 7er-Reihe

Schreibe ein kleines Skript, was die 7er-Reihe (bis 70) auf der Konsole ausgibt.

7er-Reihe mit beliebigem oberen Ende

Lies eine positive ganze Zahl `x` ein, gib die 7er-Reihe von 7 bis `x` auf der Konsole aus.

7er-Reihe mit ganzen Sätzen

Lies wie eben eine obere Grenze für die 7-er Reihe ein. Gib dann die 7-er Reihe wie folgt auf der Konsole aus:

```
1 mal 7 ist 7
2 mal 7 ist 14
     ⋮
```

Das Gauss-Problem

Berechne die Summe der Zahlen 1 bis 100.

Schwierigere Übungen

Das Gauss-Problem

Berechne die Summe der Zahlen 1 bis 100.

Zahlenmuster

Gib folgendes Muster auf der Konsole aus:

```
1
1 2
1 2 3
1 2 3 4
    ⋮
1 2 ··· 20
```

Quersumme

Lies eine ganze Zahl `x` ein und bestimme ihre Quersumme.

Tipp 1: Die Anzahl der Stellen einer Zahl bekommt man mittels `len(str(x))` heraus.
Tipp 2: Man benötigt Tipp 1 gar nicht.

Harte Übungen

Quersumme

Lies eine ganze Zahl `x` ein und bestimme ihre Quersumme.

Tipp 1: Die Anzahl der Stellen einer Zahl bekommt man mittels `len(str(x))` heraus.
Tipp 2: Man benötigt Tipp 1 gar nicht.

Primzahltest

Lies eine ganze Zahl `x` ein und überprüfe, ob diese Zahl eine Primzahl ist. Das Programm soll etwa folgende Ausgabe liefern

```
Die Zahl 28061983 ist eine Primzahl.
```

Die While-Schleife

Wie die For-Schleife nur abstrakter und open-end

Problemstellung

Lies immer wieder eine Zahl von der Konsole ein. Höre auf, wenn diese Zahl 7 ist.

Wie macht man das?

Lösung

```
x = 0

while x != 7:
  x = input("Enter a number")
  x = int(x)

print("Yeah, you picked the right number.")
```

Struktur der `While`-Schleife

Struktur der `While`-Schleife

```
while
```

Struktur der `While`-Schleife

`while` *Bedingung*

Struktur der `While`-Schleife

`while` *Bedingung* :

Struktur der `While`-Schleife

`while` *Bedingung* :

␣␣ Codezeile 1

Struktur der `While`-Schleife

`while` *Bedingung* :

␣␣ Codezeile 1

␣␣ Codezeile 2

Struktur der `While`-Schleife

`while` *Bedingung* :

␣␣ Codezeile 1

␣␣ Codezeile 2

␣␣ ⋮

Struktur der `While`-Schleife

`while` *Bedingung* :

␣␣ Codezeile 1

␣␣ Codezeile 2

␣␣ ⋮

Code, der nicht mehr Teil der Schleife ist

Struktur der `While`-Schleife

```
while Bedingung:
␣␣ Codezeile 1
␣␣ Codezeile 2
␣␣      ⋮
Code, der nicht mehr Teil der Schleife ist
```

Wie funktioniert's?

Die Schleife wird solange ausgeführt, wie die *Bedingung* `True` ergibt. Nach jedem Durchgang wird der Ausdruck der *Bedingung* neu ausgewertet. Ist die Bedingung `False` wird der Code unterhalb des Schleifenblocks ausgeführt.

Achtung Endlosschleife

Man sollte immer darauf achten, dass die Bedingung in der `while`-Schleife auch wirklich irgendwannmal `False` wird. Ansonsten bleibt das Programm in einer *Endlosschleife* gefangen.

Übung

Ersetze eine `for`-Schleife durch eine `while`-Schleife

Schreib ein Programm, dass alle 7er-Zahlen von 7 bis 700 auf der Konsole ausgibt.

Übung

Ersetze eine `for`-Schleife durch eine `while`-Schleife

Schreib ein Programm, dass alle 7er-Zahlen von 7 bis 700 auf der Konsole ausgibt.

Bestätigungsdialog *deluxe*

Verbessere den Bestätigungsdialog:

`Are you sure to continue? (y)es/(n)o.`

Mögliche Antworten sind `yes`, `no` bzw. `y`, `n`.

Daraufhin soll auf der Konsole `continued` oder `aborted` erscheinen. Falls die Eingabe nicht klar erkannt wird, soll die Frage nochmal auf der Konsole gestellt werden.

Zum Beispiel: `Please enter either "yes" or "no"`

Übung

Notenrechner

Schreib ein Programm, dass wiederholt nach einer Note von Dir fragt und Dir dann jeweils die aktuelle Durchschnittsnote auf der Konsole ausgibt. Das Programm soll durch die Eingabe vom Buchstaben **q** beendet werden können.

Beispielausgabe:

```
Bitte gib eine Note oder q zum Beenden ein: 1
Deine Durchschnittsnote ist 1.0
Bitte gib eine Note oder q zum Beenden ein: 2
Deine Durschnittsnote ist 1.5
    ⋮
```

Übung

Ratespiel

Definiere eine positive ganze Zahl `number_to_guess`. Der User kann nun wiederholt eine Zahl eingeben. Das Spiel endet, wenn die eingegebene Zahl mit `number_to_guess` übereinstimmt. Andernfalls wird auf der Konsole beispielsweise ausgegeben:

```
Sorry, Deine eingegebene Zahl war zu klein, versuche es nochmal:
```

Übung

Ratespiel

Definiere eine positive ganze Zahl `number_to_guess`. Der User kann nun wiederholt eine Zahl eingeben. Das Spiel endet, wenn die eingegebene Zahl mit `number_to_guess` übereinstimmt. Andernfalls wird auf der Konsole beispielsweise ausgegeben:

```
Sorry, Deine eingegebene Zahl war zu klein, versuche es nochmal:
```

Zusatz 1:
Am Ende soll die Anzahl der Versuche angegeben werden.

Übung

Ratespiel

Definiere eine positive ganze Zahl `number_to_guess`. Der User kann nun wiederholt eine Zahl eingeben. Das Spiel endet, wenn die eingegebene Zahl mit `number_to_guess` übereinstimmt. Andernfalls wird auf der Konsole beispielsweise ausgegeben:

```
Sorry, Deine eingegebene Zahl war zu klein, versuche es nochmal:
```

Zusatz 1:
Am Ende soll die Anzahl der Versuche angegeben werden.

Zusatz 2:
Das Spiel soll mit der Eingabe von q abgebrochen werden können.

Übung

Ratespiel

Definiere eine positive ganze Zahl `number_to_guess`. Der User kann nun wiederholt eine Zahl eingeben. Das Spiel endet, wenn die eingegebene Zahl mit `number_to_guess` übereinstimmt. Andernfalls wird auf der Konsole beispielsweise ausgegeben:

```
Sorry, Deine eingegebene Zahl war zu klein, versuche es nochmal:
```

Zusatz 1:
Am Ende soll die Anzahl der Versuche angegeben werden.

Zusatz 2:
Das Spiel soll mit der Eingabe von q abgebrochen werden können.

Zusatz 3:
Google, wie Python die Zahl `number_to_guess` zufällig erzeugen kann (das verbessert das Gameplay).

Arbeit mit einer IDE

Integrierte Entwicklungsumgebungen (IDE)

Die Arbeit mit gewöhnlichen Texteditoren ist auf Dauer sehr mühsam. Daher empfiehlt es sich eine IDE zu verwenden. Das bringt zum u.a. folgende Vorteile:

Integrierte Entwicklungsumgebungen (IDE)

Die Arbeit mit gewöhnlichen Texteditoren ist auf Dauer sehr mühsam. Daher empfiehlt es sich eine IDE zu verwenden. Das bringt zum u.a. folgende Vorteile:

- Syntax-Highlighting

Integrierte Entwicklungsumgebungen (IDE)

Die Arbeit mit gewöhnlichen Texteditoren ist auf Dauer sehr mühsam. Daher empfiehlt es sich eine IDE zu verwenden. Das bringt zum u.a. folgende Vorteile:

- Syntax-Highlighting
- Code-Inspection

Integrierte Entwicklungsumgebungen (IDE)

Die Arbeit mit gewöhnlichen Texteditoren ist auf Dauer sehr mühsam. Daher empfiehlt es sich eine IDE zu verwenden. Das bringt zum u.a. folgende Vorteile:

- Syntax-Highlighting
- Code-Inspection
- Autocomplete

Integrierte Entwicklungsumgebungen (IDE)

Die Arbeit mit gewöhnlichen Texteditoren ist auf Dauer sehr mühsam. Daher empfiehlt es sich eine IDE zu verwenden. Das bringt zum u.a. folgende Vorteile:

- Syntax-Highlighting
- Code-Inspection
- Autocomplete
- Geile Shortcuts

Integrierte Entwicklungsumgebungen (IDE)

Die Arbeit mit gewöhnlichen Texteditoren ist auf Dauer sehr mühsam. Daher empfiehlt es sich eine IDE zu verwenden. Das bringt zum u.a. folgende Vorteile:

- Syntax-Highlighting
- Code-Inspection
- Autocomplete
- Geile Shortcuts
- Code direkt ausführen

Integrierte Entwicklungsumgebungen (IDE)

Die Arbeit mit gewöhnlichen Texteditoren ist auf Dauer sehr mühsam. Daher empfiehlt es sich eine IDE zu verwenden. Das bringt zum u.a. folgende Vorteile:

- Syntax-Highlighting
- Code-Inspection
- Autocomplete
- Geile Shortcuts
- Code direkt ausführen
- Hilfe bei der Fehlersuche (*Debugging*)

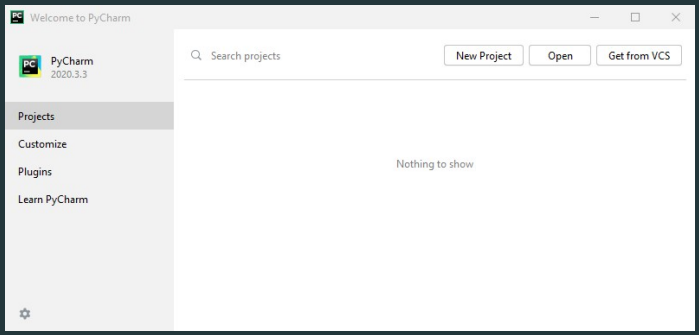
Installation von PyCharm

1. Gehe auf `https://www.jetbrains.com/pycharm/download`
2. Lade die kostenlose *Community Edition* herunter
3. Führe den Installer aus
4. Öffne PyCharm

Installation von PyCharm

1. Gehe auf `https://www.jetbrains.com/pycharm/download`
2. Lade die kostenlose *Community Edition* herunter
3. Führe den Installer aus
4. Öffne PyCharm

Wenn alles passt, sollte es etwa so aussehen:



Installation PyCharm

1. Gehe auf `Customize > All Settings...`
2. Einstellungen synchronisieren
   2.1 `Tools > Settings Repository`
   2.2 Unter *Read-only Sources* auf +
   2.3 `https://github.com/a-kunert/ide-settings.git` eingeben
3. Verknüpfe den Interpreter
   3.1 In den Settings auf `Python Interpreter`
   3.2 Falls möglich unter `Python Interpreter` einen Interpreter wählen. Ansonsten wie folgt:
   3.3 `Zahnrad > Add`
   3.4 `System Interpreter`
   3.5 Dort den Pfad zu Python angeben
4. Mit dem Button *Apply* alles bestätigen

Konfiguration abschließen

1. Lege eine Ordner für den Kurs an
2. Öffne diesen Ordner mit `Projects > Open`
3. `File > Manage IDE Settings > Sync with Settings Repository > Merge` ausführen
4. Bei `File > Settings` unter `Keymap` die Keymap *Salem-Win/Mac* auswählen.
5. Code in die Datei `main.py` schreiben
6. Mittels grünem Pfeil (oben rechts) Code ausführen