geekIT

Python Komplett

5-Tages Seminar

Tag 1+2

Timm Gieger

GFU Cyrus AG

Agenda – Tag 1+2

Vorstellungsrunde

Einführung Python & Entwicklungsumgebung

Variablen & Datentypen

Bedingungen & Schleifen

Funktionen / Methoden

Exceptions

Feedback & Fragerunde

Agenda – Tag 3-5

Ausstehende Themen von Tag 1+2

Klassen / Objektorientierte Programmierung

Dateien einlesen / OS-Befehle

SQL Grundlagen / SqlAlchemy / Pandas / numpy

Testing

Ausblick / eigene Themen

Feedback & Fragerunde

Timm Gieger

Timm Gieger ist seit 2019 in den Bereichen Data Science & Machine Learning tätig. Seine bisherigen Projekte beschäftigten sich hauptsächlich mit der Konzeption & Implementierung intelligenter Systeme zur Unterstützung der geschäftlichen Entscheidungsfindung oder Prozessautomatisierung. Sein gesammeltes Wissen & seine Erfahrungen gibt er als Trainer & Dozent mit Leidenschaft weiter. Er zeichnet sich durch seine Empathie, Anpassungsfähigkeit und hohe Eigenverantwortung aus.

Vorstellungsrunde

- Hintergrund – was mache ich in meinem Beruf / Tätigkeit?
- Habe ich bereits Erfahrungen mit Python / SQL gemacht?
- Warum besuche ich diese Schulung?
- Welche Vorkenntnisse habe ich? (Allgemein Programmierung etc.)
- Was erwarte ich mir von dieser Schulung?

Lernziele

- Umgang und Bedienung der wichtigsten Tools bei der Entwicklung mit Python
- Beherrschung der allgemeinen Grundlagen der Programmiersprache Python (Variablen, Datentypen, Sequentielle Datentypen…)
- Objektorientierte Programmierung mit Python (abstrakte Klassen)
- Erste Einblicke in das Gebiet Data Science mit Pandas,Numpy,SQL
- Möglichkeit weitere Lernquellen auszuprobieren

Literatur

Buchquelle für diese Schulung:

- Klein, Bernd, Einführung in Python3: Für Ein- und Umsteiger, Hanser Verlag, 2021

Onlinequellen:

Learnpython.org: https://www.learnpython.org/

W3schools: https://www.w3schools.com/

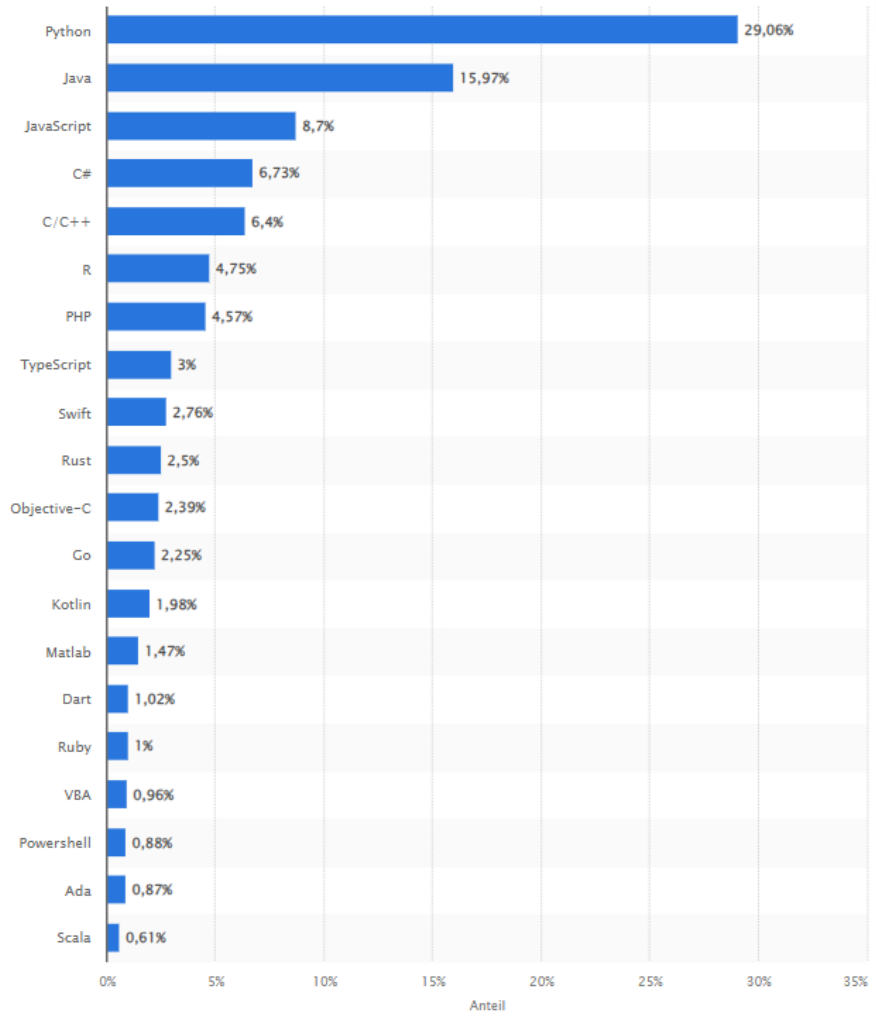
Einführung Python
&
Entwicklungsumgebung

Was ist Python?

- Wurde in den 1990er Jahren von Guido van Rossum entwickelt
- Als Nachfolger der Programmiersprache ABC angedacht
- Name Inspiriert von Monty Python (Flying Circus) → nicht von Python als Schlange
- Verbindung mit Schlange trotzdem vorhanden (z.B.: Toolkit Boa oder Logo)
- Python 1.0 – 1994 ; Python 3.0 - 2008

Source: Klein, B., Einführung in Python 3, 2021, Seite 3 f.

Beliebteste Programmiersprachen weltweit laut PYPL-Index [Juni 2024]

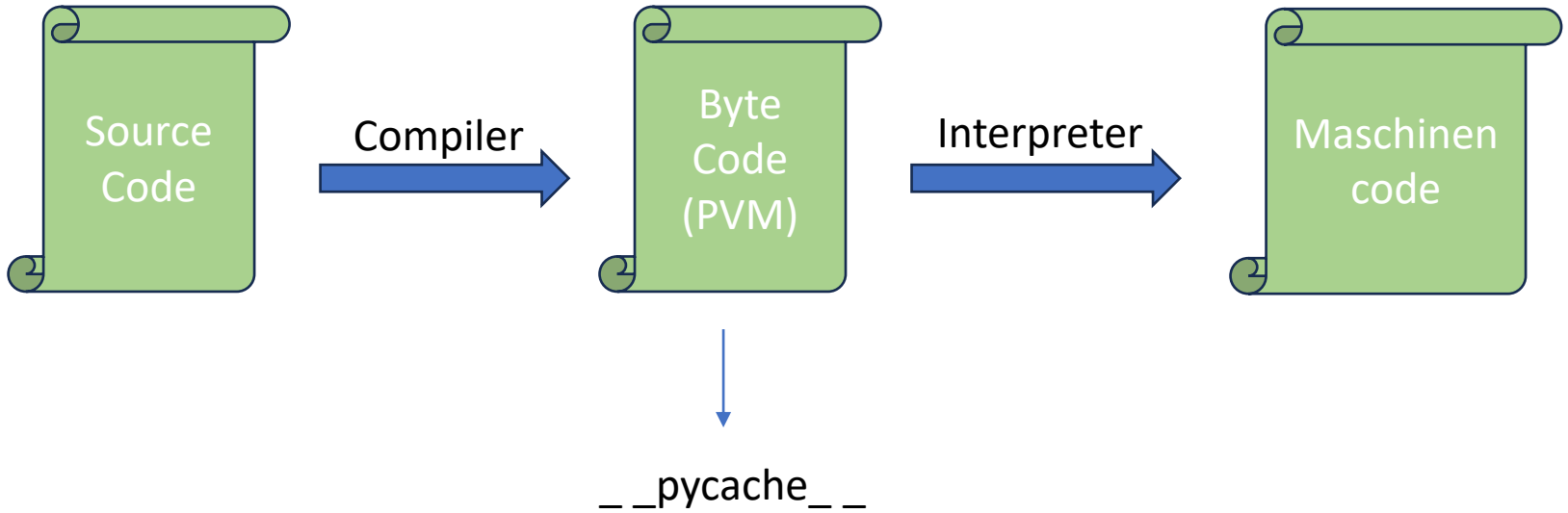


Source: https://de.statista.com/statistik/daten/studie/678732/umfrage/beliebteste-programmiersprachen-weltweit-laut-pypl-index/

Was ist Python?

Interpreter oder Compilersprache?

Sowohl als auch



Source: Klein, B., Einführung in Python 3, 2021, Seite 21 f.

Entwicklungsumgebungen

- PyCharm – Python IDE (Entwicklungsumgebung)

- Anaconda - Data Science Plattform für Python

- Jupyter Notebook

Importieren der Schulungsunterlagen aus GitHub

1. Rechtsklick in gewünschten Dateipfad → „Git Bash Here“

2. „git clone https://github.com/timmg-cyber/python_komplett.git “

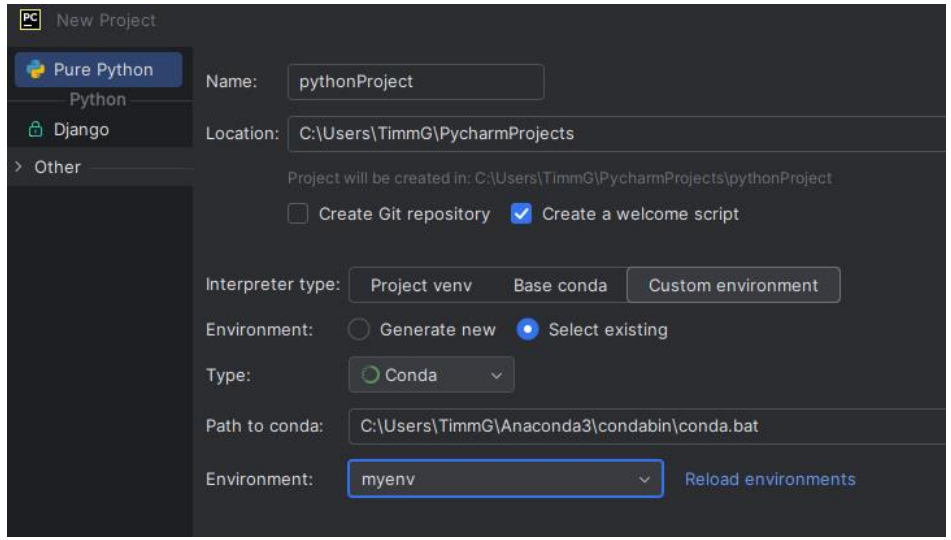
Anlegen eines venv in Anaconda

1. Anaconda Navigator öffnen (oder direkt Anaconda Prompt über Windows-Suche)
2. Anaconda Prompt installieren und starten (falls noch nicht installiert)
3. „conda create –n myenv python=3.11“
4. Navigieren in Verzeichnis von requirements.txt (Schulungsunterlagen)
5. „pip install –r requirements.txt“
6. Aktivieren der venv mit „conda activate myenv“

My First Python Project

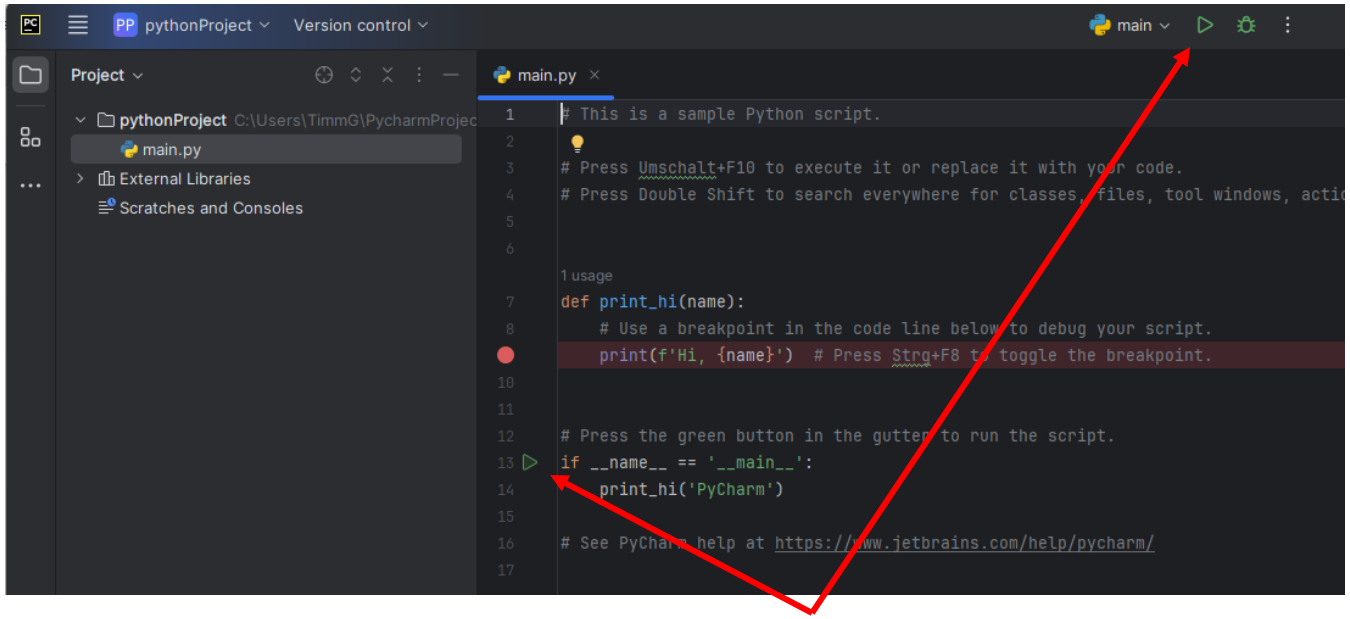
1. PyCharm öffnen
2. New Project klicken
3. „Create a welcome script“ auswählen

   Bei Interpreter type: Custom environment wählen, dann Environment: Select existing → Type: Conda aus Dropdown Environment das zuvor erstellte Environment auswählen



My First Python Project

4. Create klicken (unten rechts)



5. Ausführen des Welcome Scripts mit Play-Symbol oder Umschalt+F10

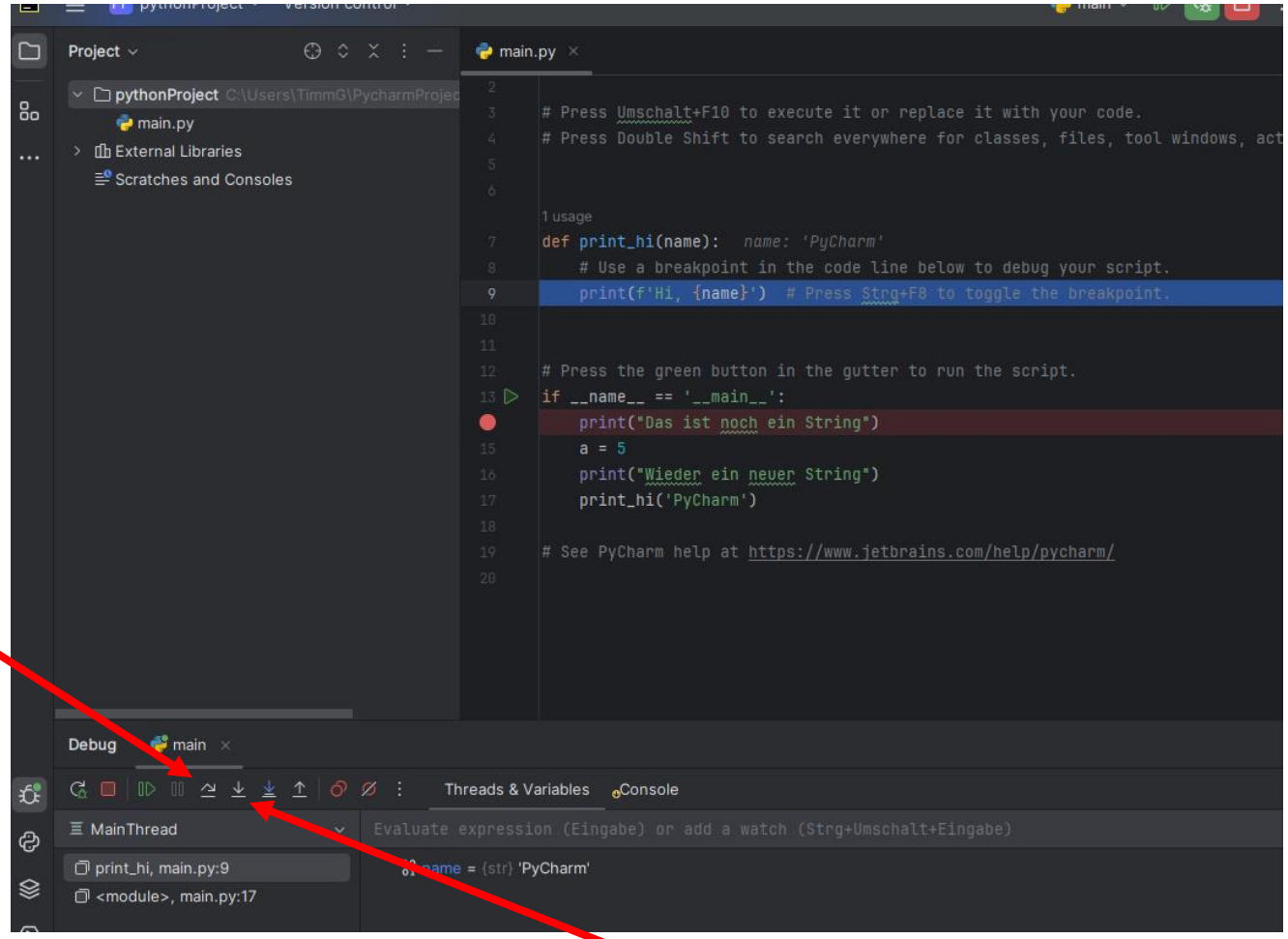
My First Python Project

Debug Mode

1. Fügen Sie print(“Das ist noch ein String“) und a=5 in zwei separaten Zeilen in der main()-Methode hinzu
2. Klicken Sie neben die erste Codezeile, damit ein roter Button erscheint (Breakpoint)
3. Klicken Sie auf das „Bug“-Symbol oder Umschalt+F9
4. Es sollte nun das Debug-Fenster erscheinen. Klicken Sie links unten auf den Pfeil mit „Step Over“ oder F8 bis Sie die Variable a in Ihrem Debug-Fenster sehen.
5. Wenn Sie beim Methodenaufruf „print_hi“ angekommen sind drücken Sie statts „Step Over“, „Step Into“ oder F7
6. Sie sollten sich nun in der Methode befinden.

My First Python Project

Debug Mode



Step Over (F8)

Step Into (F7)

01 Variablen & Datentypen

Variablen

Referenzierung in Python

Code:

```
x = 38    ## Variablenzuweisung
y = x
```

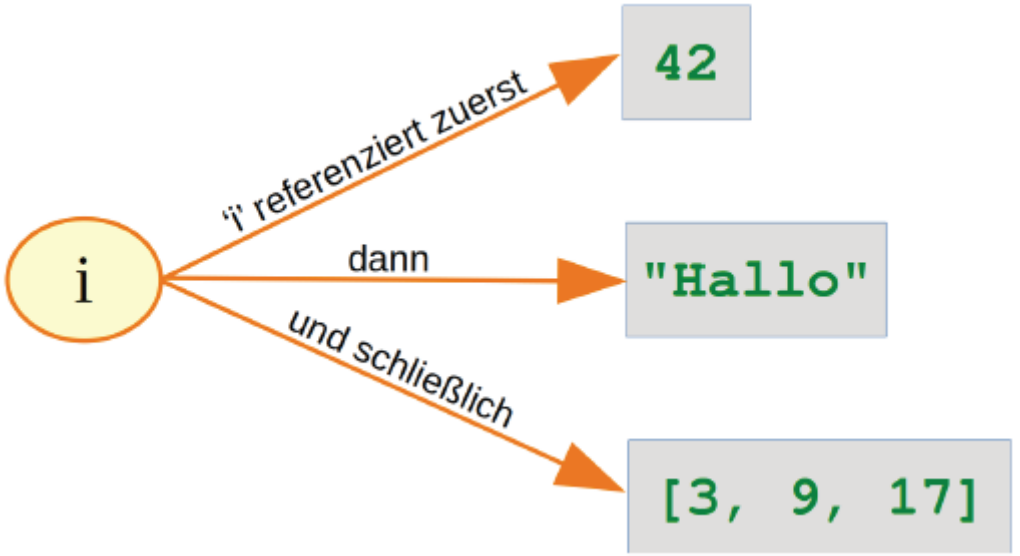
Interne Repräsentation

Variablen

- Typendeklaration in Python ist dynamisch
- Java, C und C++ haben statische Typendeklaration

→ Datentyp einer Variable kann sich während der Laufzeit mehrmals ändern

Achtung! Dies kann auch zu ein paar ungewollten Type Exceptions führen



Datentypen

Datentyp	Bezeichnung	Beschreibung	Beispiel
int	Integer	Ganzzahliger Wert	2
float	Float	Fließkommazahl	2.0
list	Liste	Sequentielle Liste mit einer Sammlung von Objekten	[1,2.0,“abc“]
dict	Dictionary	Sammlung von Key/Value Pairs, ähnlich wie JSON aufgebaut	{“key“: “value“}
str / object	String	Zeichenkette	“Hallo Welt“
bool	Boolean	Boolscher Wert	True / False
complex	Complex	Komplexe Zahlen	4 + 4.5j

Variablen & Datentypen mit Python

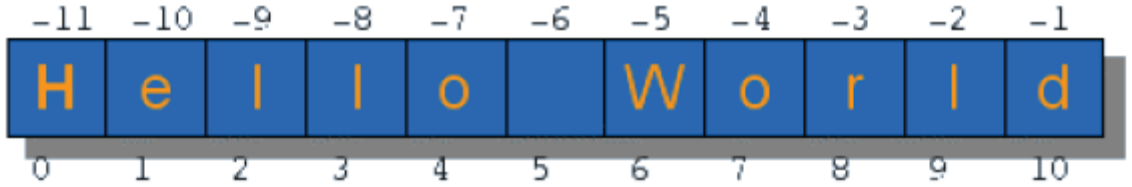
Sequentielle Datentypen

- Umfasst Listen, Tupel und sogar Strings
- Objekte, deren Inhalt sequentiell dargestellt ist
- Teilen sich grundlegende Methoden und Funktionen (z.B.: len())

Sequentielle Datentypen

Indizierung

- Sequentielle Datentypen können über Index indiziert werden
- Jeder Index ist unique
- Aufrufen eines nicht vorhandenen Indizes führt zu einer Out of Bound Exception



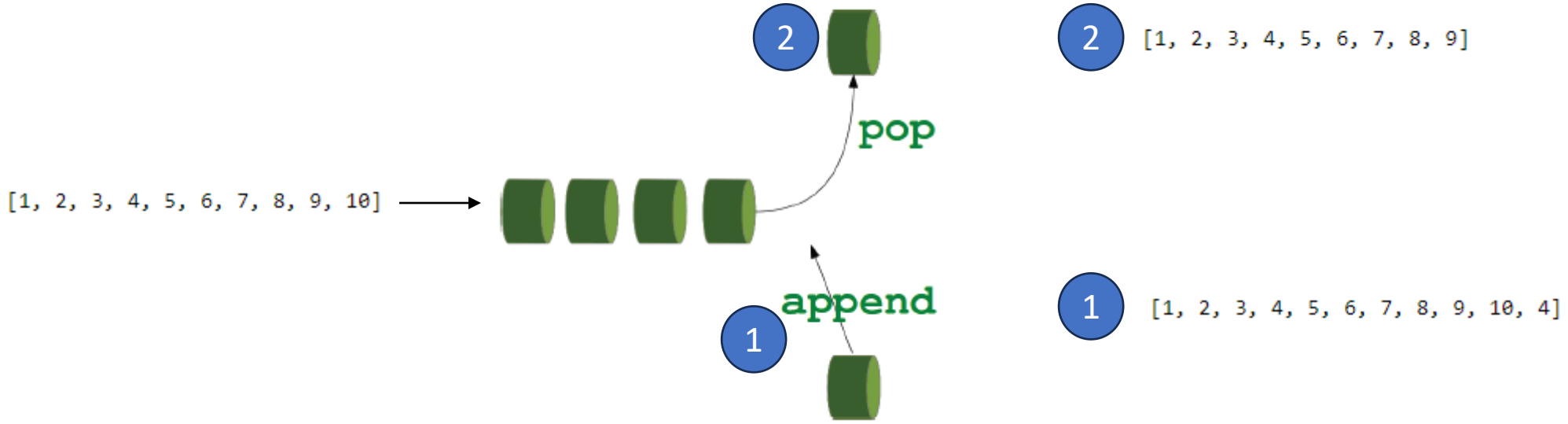

```
In [14]: "Hello World"[2]
Out[14]: 'l'

In [17]: "Hello World"[6:]
Out[17]: 'World'
```


Source: Klein, B., Einführung in Python3, 2021, S. 43

Sequentielle Datentypen

Elemente hinzufügen/entfernen



Source: Klein, B., Einführung in Python3, 2021, S. 52 f.

Sequentielle Datentypen mit Python

Sequentielle Datentypen

Dictionaries

- Key/Value pairs
- Ähnlich wie JSON-Datei
- Dient als Datenspeicher/Zwischenspeicher während der Laufzeit eines Programms

Beispiel:

```
In [1]: my_dict = {
            "key": "value",
            "key2": 1,
            "key3": ["value1",2,(1,2,3)]
        }
```

Dictionaries mit Python

02 Bedingungen & Schleifen

Bedingungen

- Python benutzt Einrückungen und Spaces um Codeabschnitte voneinander zu trennen

Bsp.:

Java

```
Anweisungskopf:  {
Anweisung;
Anweisung;
}
```

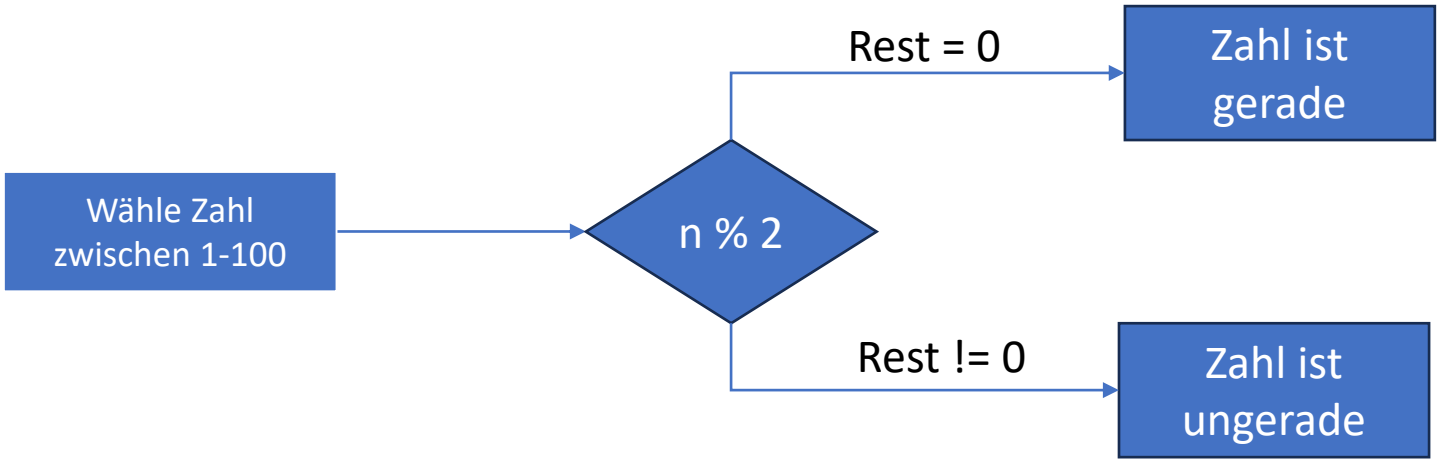
Python:

```
Anweisungskopf:
        Anweisung
        Anweisung
```

Problem: Falscher Tab oder Space zwischen Code kann das ganze Programm ändern und kann schwer zu finden sein

Bedingungen

Gerade vs Ungerade Zahl



Bedingungen

If-/else Anweisung

Normale if-Anweisung

```
If bedingung:
        anweisungen
```

If/Else

```
If bedingung:
        anweisungen
else:
        anweisungen
```

If/Elif/Else

```
If bedingung:
        anweisungen
elif bedingung:
        anweisungen
elif bedingung:
        amweisungen
else:
        anweisungen
```

Bedingungen

If-/else Anweisung

Gerade vs. Ungerade Zahl

Richtig

```
import random

zahl = random.randint(0,100)

result = zahl % 2

if result != 0:
    print(f"Die Zahl {zahl} ist ungerade")
else:
    print(f"Die Zahl {zahl} ist gerade")
```

```
Die Zahl 39 ist ungerade
```

Falsch

```
import random

zahl = random.randint(0,100)

result = zahl % 2

if result != 0:
    print(f"Die Zahl {zahl} ist ungerade")
else:
    print("Hallo")
print(f"Die Zahl {zahl} ist gerade")
```

```
Die Zahl 53 ist ungerade
Die Zahl 53 ist gerade
```

Bedingungen

Boolsche Operatoren

Operator	Erklärung	Beispiel	Wahrheitswert
==	Prüfung auf Gleichheit	`42 == 42` oder `[3, 4, 8] == [3, 4, 8]`	True
!=	Prüfung auf Ungleichheit	`42 != 43` oder `[3, 4, 8] != 17`	True
<	Prüfung auf „kleiner"	`4 < 12` oder `"Tisch" < "Tischbein"`	True
<=	Prüfung auf „kleiner gleich"	`4 <= 12` oder `"Tischbein" <= "Tischbein"`	True
>	Prüfung auf „größer"	`4 > 12` oder `"Tisch" > "Tischbein"`	False
>=	Prüfung auf „größer gleich"	`40 >= 12` oder `"Tischbein" >= "Tischbein"`	True

Source: Klein, B., Einführung in Python3, 2021, S. 68 f.

Bedingungen

Ausnahme

```
x = int(input("Zahl: "))

if x:
    print("Die eingegebene Zahl ist ungleich Null")
else:
    print("Die eingegebene Zahl ist gleich Null")
```

„False“ wenn:

- Zahl numerischer Null-Wert
- Leere Zeichenkette
- Leere Liste/Tupel
- Leeres Dictionary
- None → Fehlwert

Source: Klein, B., Einführung in Python3, 2021, S. 69

Bedingungen mit Python

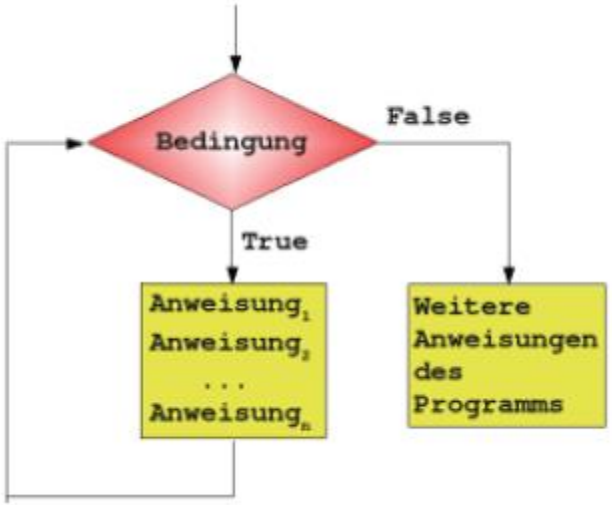
Schleifen

- Werden benötigt um sequentielle oder iterierbare Objekte zu durchlaufen
- Erlauben mehr Flexibilität und Übersichtlichkeit im Code
- Erhöhen die Ausführungszeit enorm (so viel wie nötig, so wenig wie möglich)
- Die Gefahr von Endlosschleifen besteht
- Schleifen können mit „break“ zwangsweise beendet werden
- „continue“ sorgt für das Springen in den nächsten Schleifendurchlauf

Schleifen

While-Schleife

- Bedingung wird vor erster Ausführung geprüft (kopfgesteuerte Schleife)
- Solange die Bedingung zutrifft, werden alle Anweisungen in der Schleife ausgeführt
- Schleife muss 1x durchlaufen werden bevor Bedingung erneut überprüft wird

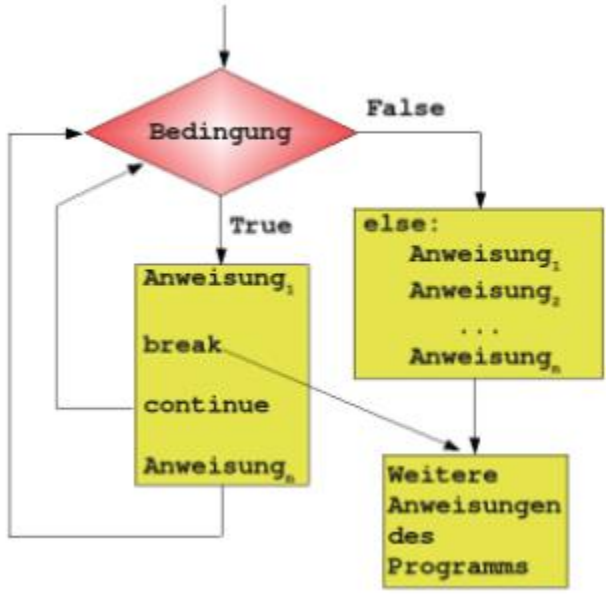


Source: Klein, B., Einführung in Python3, 2021, S. 72 f.

Schleifen

While-else

- Wenn Bedingung False wird, wird Else-Zweig ausgeführt
- Nicht unbedingt notwendig
- Wird nicht bei unnatürlicher Beendigung des loops ausgeführt



Source: Klein, B., Einführung in Python3, 2021, S. 74 f.

Schleifen

For-Schleife

- sehr sprechend umgesetzt in Python

- Gibt nicht wie in Java den Schleifenindex zurück

Syntax:

```
for elem in elements:
        Anweisung
else:
        Anweisung
```

Schleifen

For-Schleife

Enumerator:

- Wird automatisch hochgezählt

- Kann als Schleifenindex benutzt werden

```
for i,elem in enumerate(elements):
        Anweisung
```

Schleifen mit Python

03 Funktion / Methoden

Funktionen / Methoden

Syntax

- Kann von Programm aufgerufen werden
- Mehr Flexibilität durch Parameter
- Rückgabewerte möglich

Syntax:

Definition:

```
def funktionsbezeichnung(Parameter):
        Anweisungen
        return Rückgabewert
```

Aufruf:

```
funktionsbezeichnung(Parameter)
```

Funktionen / Methoden

Docstring

- Ermöglicht es Funktionen/Methoden direkt zu dokumentieren

Syntax:

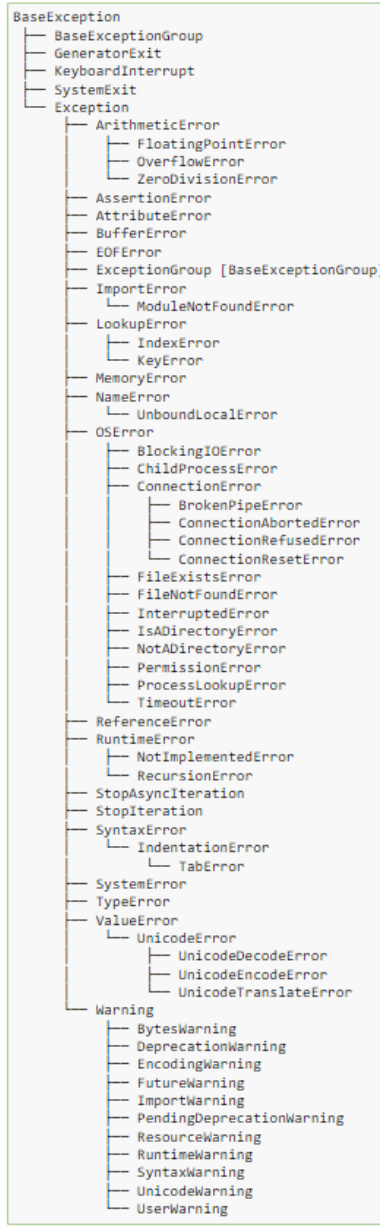
```
>>> def fahrenheit(T_in_celsius):
...     """ returns the temperature in degrees Fahrenheit """
...     return (T_in_celsius * 9 / 5) + 32
...
```

Funktionen / Methoden mit Python

04 Exceptions

Exceptions

- Fehlerbehandlung in Python-Programmen
- Ermöglicht es unbehandelte Fehler aufzufangen ohne Programmabbruch
- Es gibt verschiedene Arten von Exceptions die aufgefangen werden können



Built-In Exceptions

Exceptions

Basic Syntax:

```
try:
  Anweisung
except:
  Anweisung wenn Fehler in try-Block
```

Exceptions mit Python

Feedback & Fragerunde

- Haben Sie Fragen zu den behandelten Themen?
- Welche Themen haben Sie vermisst?
- Feedback an mich?

Ihr Ansprechpartner

Timm Gieger

mobile: +49176 40566154

mail: timm.gieger@geekit-ds.de