



**Ostfalia**  
Hochschule für angewandte  
Wissenschaften

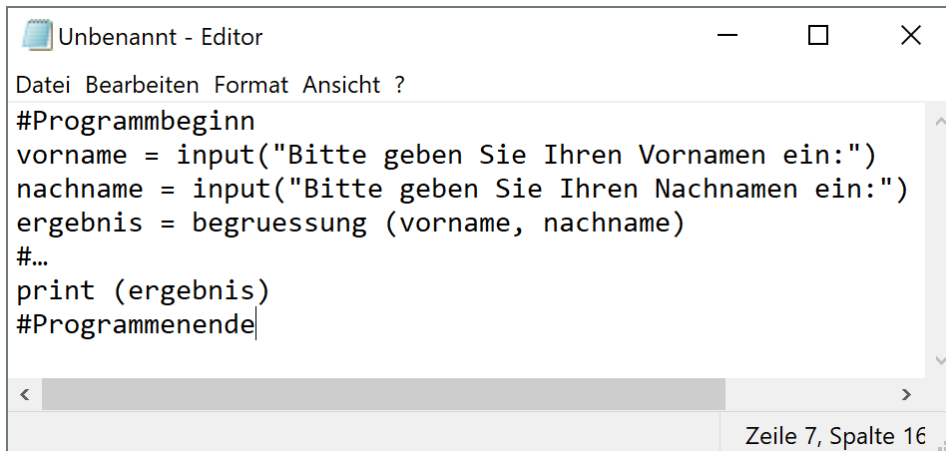
## Kapitel 4      Werkzeuge der Softwareentwicklung

# Inhalt

- Programme erstellen
  - Sie kennen die Standardwerkzeuge des Programmierers und können benennen wie diese eingesetzt werden.
  - Sie wissen, was Versionsmanagement ist, kennen wichtige Begriffe aus dem Bereich des Versionsmanagements und können am Beispiel von Git einen typischen Workflow erklären.
- Standardbibliotheken (API) und Wiederverwendung
  - Sie kennen den Python Package Index und können das zugehörige pip-Tool anwenden.
  - Am Beispiel häufig genutzter Bibliotheken (math, random, time, matplotlib) lernen Sie die Einbindung und Benutzung von Standardbibliotheken in Ihren eigenen Projekten.
- Virtualisierung mit Containern
  - Sie kennen die grundlegenden Technologie der Virtualisierung mit Containern für die Bereitstellung von Software und können Vorteile und konkrete Umsetzungsmöglichkeiten benennen.
- Fehlerarten und Fehlersuche
  - Sie können die drei wesentlichen Fehlerkategorien und deren Eigenschaften benennen.
  - Sie können den Debugger für die Fehlersuche einsetzen.

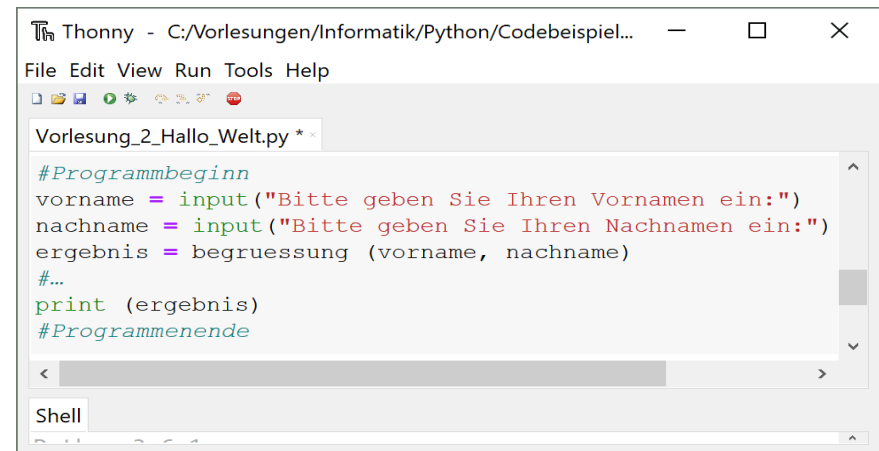
# Standardwerkzeuge des Programmierers – 1. Editor

- Jedes Programm muss zunächst geschrieben werden. Dazu können verschiedene Programmiersprachen zum Einsatz kommen.
- Im einfachsten Fall kann ein Programm dabei in einem einfachen Texteditor geschrieben werden.
- Üblicherweise werden heute aber spezielle Editoren verwendet, welche den Programmcode farblich hervorheben (Syntax-Highlighting) und meist auch Teile der Syntax automatisch ergänzen (Autovervollständigung).



```
#Programmbeginn
vorname = input("Bitte geben Sie Ihren Vornamen ein:")
nachname = input("Bitte geben Sie Ihren Nachnamen ein:")
ergebnis = begruessung (vorname, nachname)
#...
print (ergebnis)
#Programmenende
```

Zeile 7, Spalte 16



```
#Programmbeginn
vorname = input("Bitte geben Sie Ihren Vornamen ein:")
nachname = input("Bitte geben Sie Ihren Nachnamen ein:")
ergebnis = begruessung (vorname, nachname)
#...
print (ergebnis)
#Programmenende
```

Shell

## Standardwerkzeuge des Programmierers – 2. Präprozessor

- In vielen Programmiersprachen wird das geschriebene Programm in einem ersten Schritt mit einem Präprozessor vorverarbeitet.
- Dieses Werkzeug sucht in der Regel nach bestimmten Schlüsselwörtern oder definierten Konstanten und ersetzt diese im Programmcode.
- In der Programmiersprache C werden über einen solchen Präprozessor auch externe Bibliotheken eingebunden.

### C++ Programm:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

//Konstanten
#define PI 3.14159

//Makros
#define UMFANG_AUS_RADIUS(r) (2 * PI * r)
```

### Mikrocontroller: Portdefinition

```
...
/* PORTC */
#define PC7      7
#define PC6      6
#define PC5      5
#define PC4      4
#define PC3      3
#define PC2      2
#define PC1      1
#define PC0      0
```

## Standardwerkzeuge des Programmierers – 3. Compiler / Interpreter

- Damit Programme auf dem jeweiligen Prozessor lauffähig sind, müssen sie in Maschinsprache übersetzt werden.
- Diese Übersetzung für das jeweilige Prozessorsystem übernimmt der Compiler bzw. der Interpreter.
- Dabei ist grundsätzlich zwischen drei Möglichkeiten zu unterscheiden:

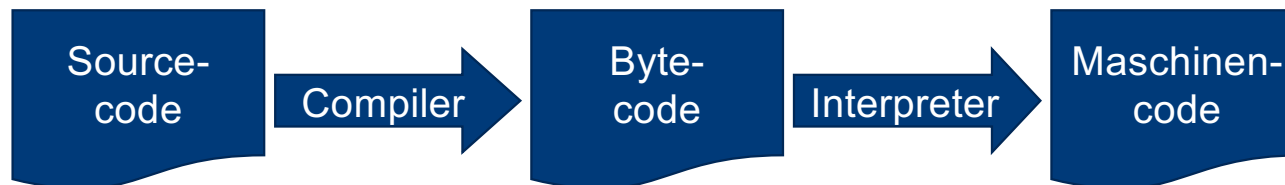
Reine Interpretersprachen  
(z.B. Basic)



Reine Compilersprachen  
(z.B. C++, C#)



Kombination  
(z.B. Java, Python)



## Standardwerkzeuge des Programmierers – 4. Debugger

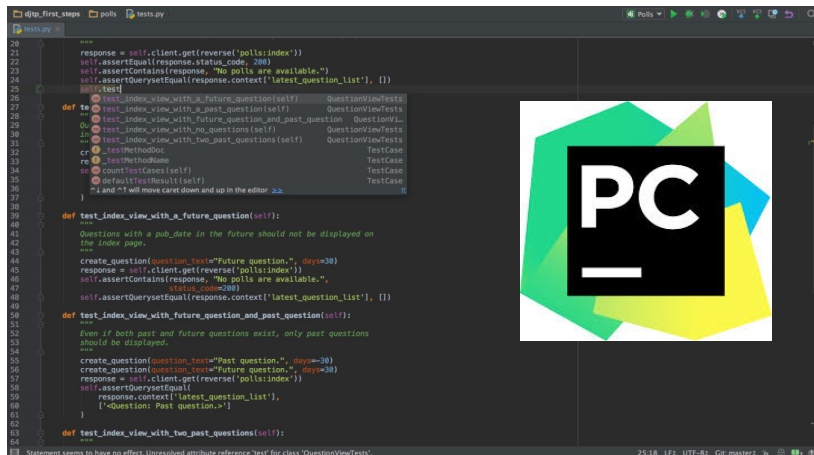
- Ein Debugger ist ein spezielles Programm bzw. Tool zum Auffinden von Fehlern.
- Debugger gibt es für nahezu alle Programmiersprachen..
- Die Programmausführung kann z.B. an einer definierten Stelle unterbrochen werden, indem man sogenannte **Haltepunkte** (Breakpoints) im Programm setzt.
- An den Haltepunkten können z.B. Werte von Variablen beobachtet und geändert werden oder der Quellcode kann Zeile für Zeile ausgeführt werden, um das Verhalten zu beobachten.
- Wichtigste Möglichkeiten für die Fehlersuche in Programmen:
  - schrittweises Verfolgen (sogenanntes **Steppen**),
  - gezieltes Anhalten (mit **Haltepunkten** beziehungsweise **Breakpoints**) oder
  - Auswertung einzelner Variablen (**Watch**) des Programms
  - gezielte Veränderung von Variablen während der Laufzeit



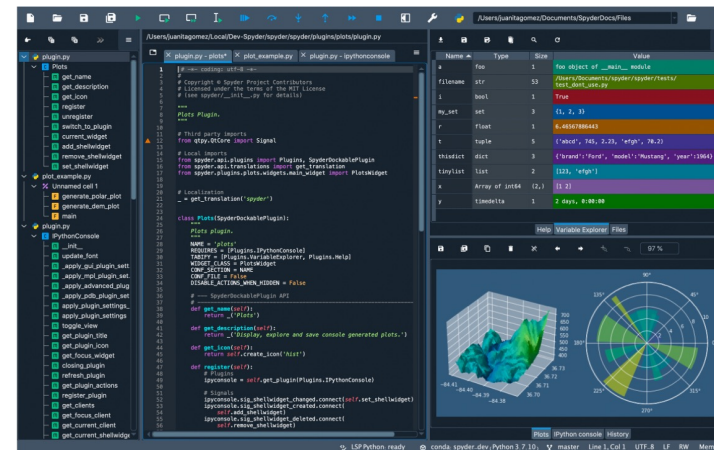
Quelle: www.keluro.com

# Standardwerkzeuge des Programmierers – 5. Entwicklungsumgebung / IDE

- Als Entwicklungsumgebung oder IDE (Integrated Development Environment) werden speziell für Programmierer gestaltete Programme bezeichnet, die die vorher genannten Werkzeuge in einer geschlossenen Umgebung zusammenführen.
- Für Python gibt es z.B. folgende IDE's:
  - **IDLE** (Integrated Development and Learning Environment): Standard IDE, wird mit dem Python Installer ausgeliefert
  - **Spyder**: Professionelle Python Entwicklungsumgebung (<https://www.spyder-ide.org>)
  - **PyCharm**: Professionelle Python Entwicklungsumgebung (<https://www.jetbrains.com/pycharm>)



Quelle: [www.jetbrains.com/pycharm/](https://www.jetbrains.com/pycharm/)



Quelle: <https://www.spyder-ide.org>

## die IDE am Beispiel von Spyder

- Die Spyder IDE besteht im Kern aus folgenden 6 Funktionalitäten und lässt durch verschieden Plugins funktional erweitern.



Editor

Work efficiently in a multi-language editor with a function/class browser, code analysis tools, automatic code completion, horizontal/vertical splitting, and go-to-definition.



IPython Console

Harness the power of as many IPython consoles as you like in one GUI. Run code by line, cell or file; or work interactively with debugging, plots and magic commands.



Variable Explorer

Interact with and modify variables on the fly: plot a histogram or timeseries, edit a dataframe or Numpy array, sort a collection, dig into nested objects, and more!



Plots

Browse, zoom, copy and save the figures and images you create.



Debugger

Trace each step of your code's execution interactively.



Help

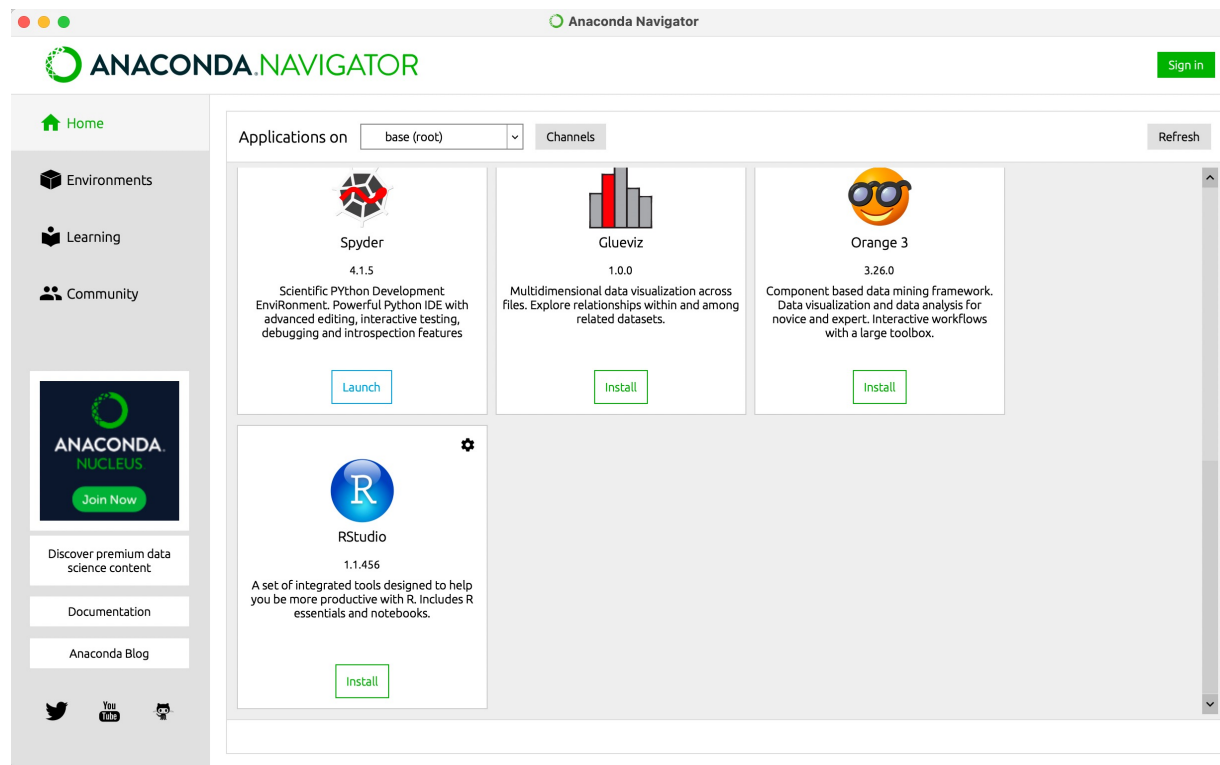
Instantly view any object's docs, and render your own.

Quelle: <https://www.spyder-ide.org>



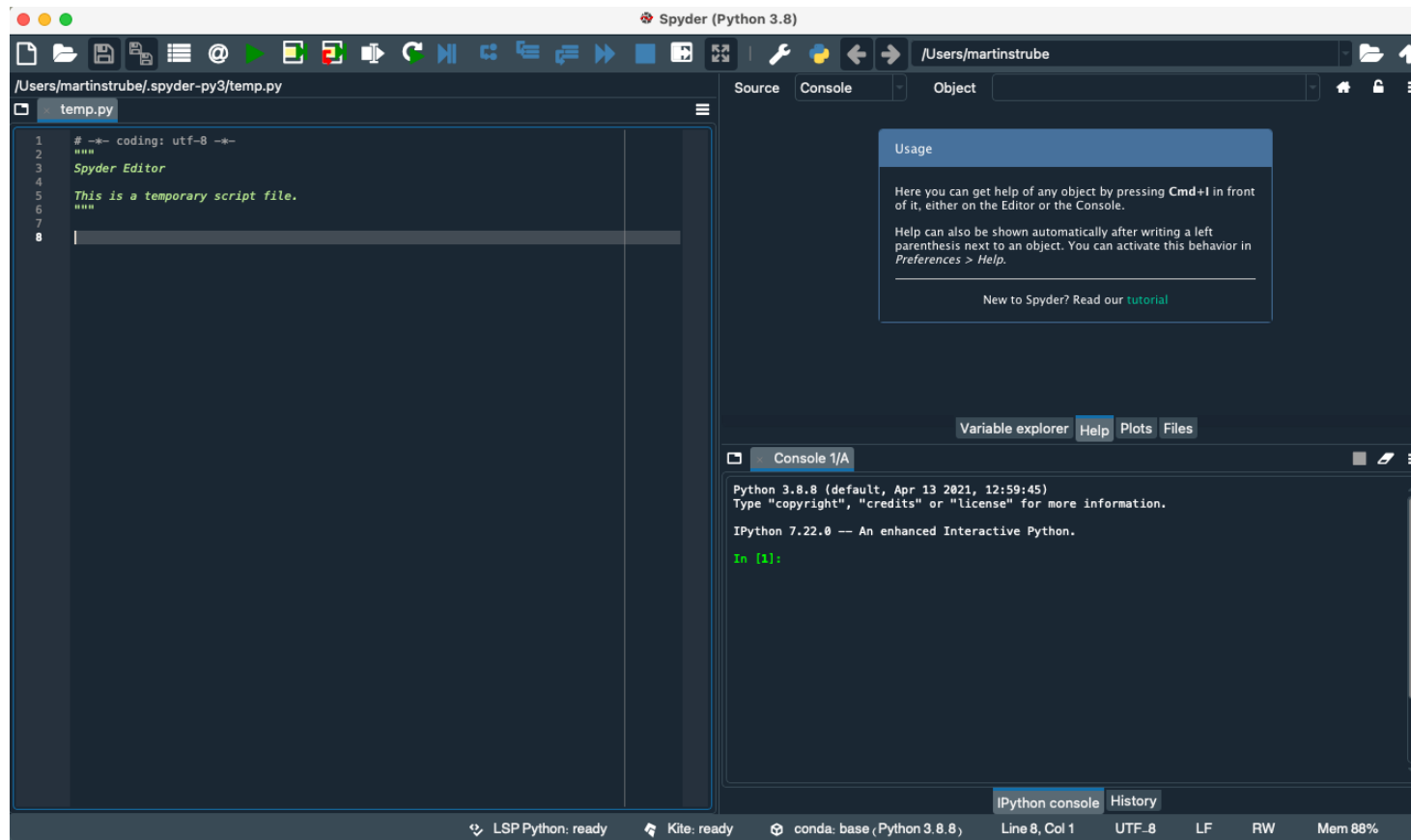
# Installation von Spyder

- Spyder ist Teil der Anaconda Distribution, daher steht Spyder nach Installation von Anaconda direkt zur Verfügung.
- Zum Starten von Spyder einfach „launch“ auf der Kachel Spyder anklicken.



# die IDE am Beispiel von Spyderdie Spyder Bedienoberfläche

- Nach dem ersten Start von Spyder öffnet sich folgende Ansicht der Bedienoberfläche

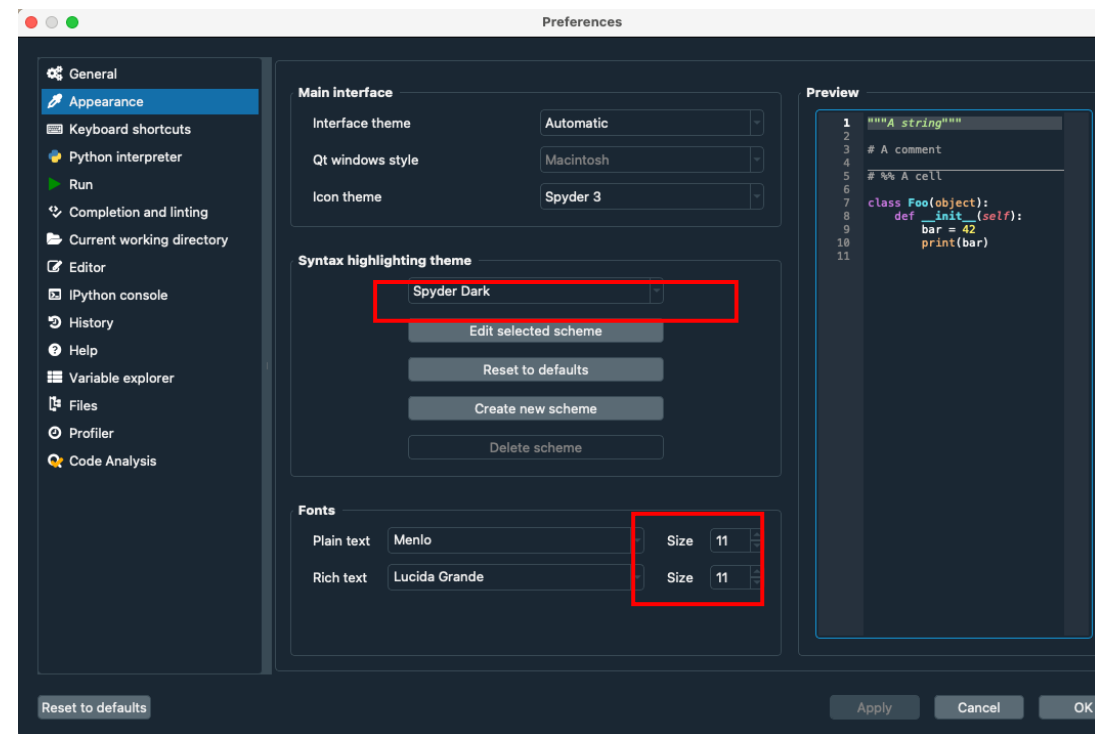


# Anpassen der Darstellung der Spyder Bedienoberfläche

- Anpassen von Schriftgröße und Darstellung

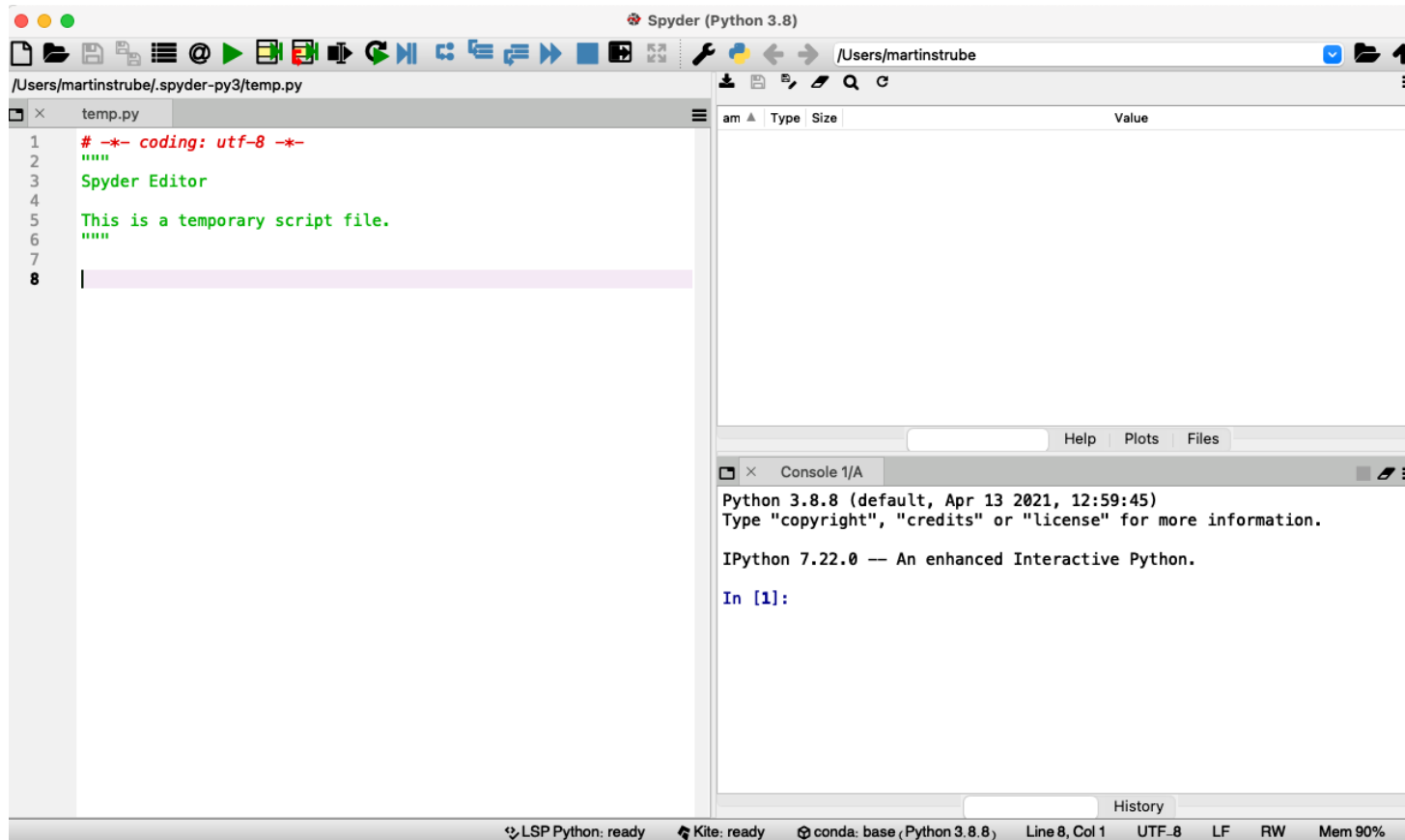
Anpassen der Darstellung:

- unter MacOS
  - Menü - python - Preferences - Appearance
- Unter Windows
  - Menü - Preferences - Appearance



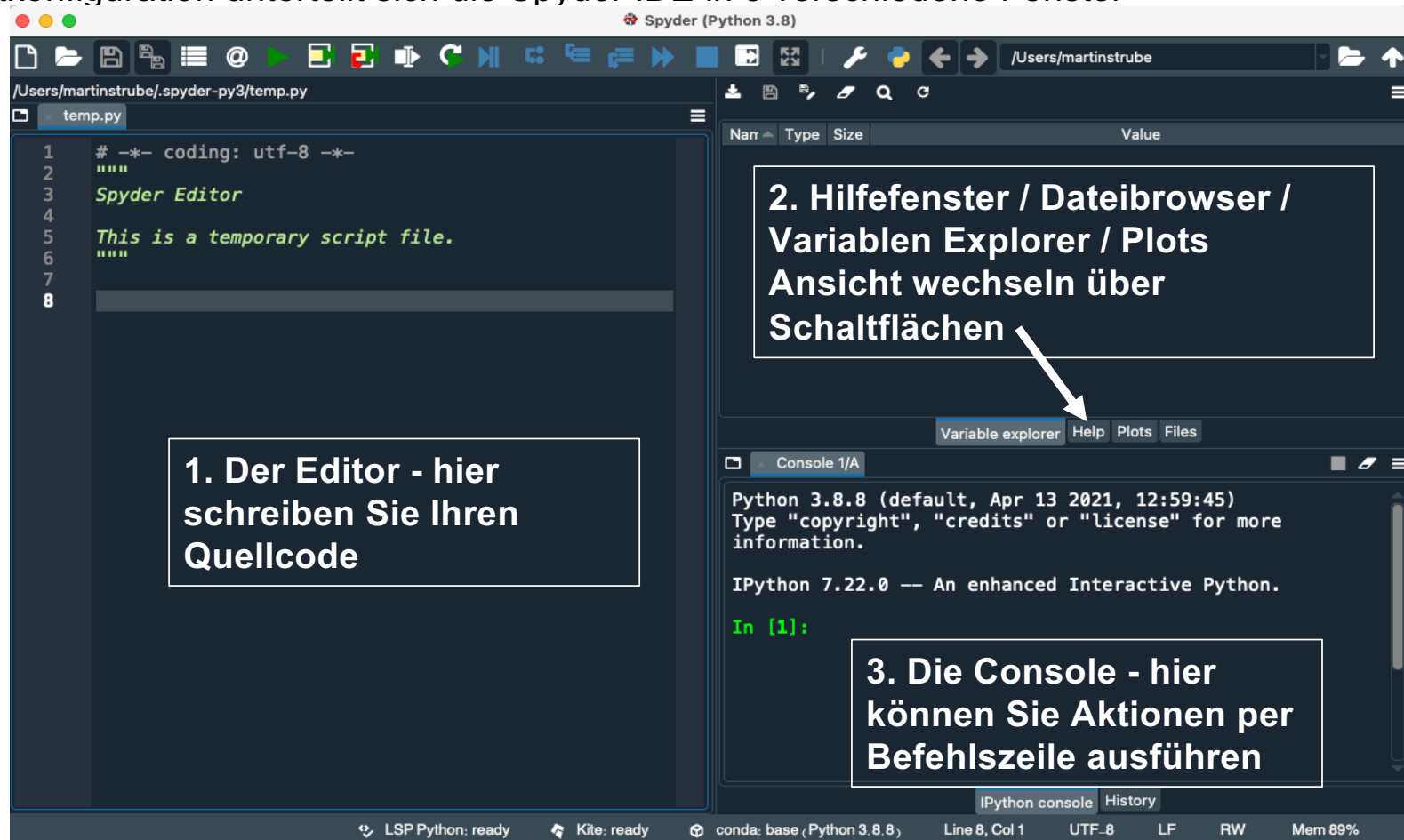
## Beispiel für eine andere Darstellung der Bedienoberfläche

- z.B. Anpassen von Schriftgröße auf 14 und Darstellung Mode IDLE



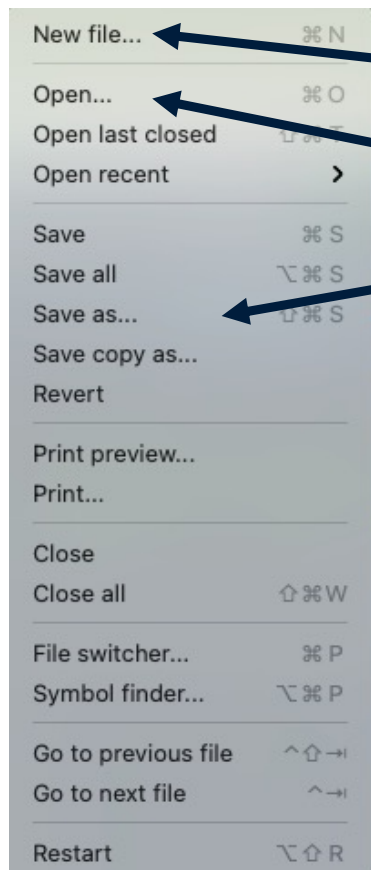
# Aufbau der Spyder Bedienoberfläche

- In der Startkonfiguration unterteilt sich die Spyder IDE in 3 verschiedene Fenster



## Erste Schritte – Dateien erstellen, umbenennen und speichern

- Über das Menü File können Sie z.B.



- eine neue python-Datei erstellen
- eine bestehende python-Datei öffnen
- eine python Datei unter neuem Namen speichern

### Aufgabe:

1. Speichern Sie die Datei *temp.py* unter dem Namen *Hallo Welt.py*
2. Erstellen Sie eine weitere Datei *EVA Prinzip.py*
3. Erstellen Sie eine weitere Datei *teilbar.py*

## Erste Schritte – Dateien erstellen, umbenennen und speichern

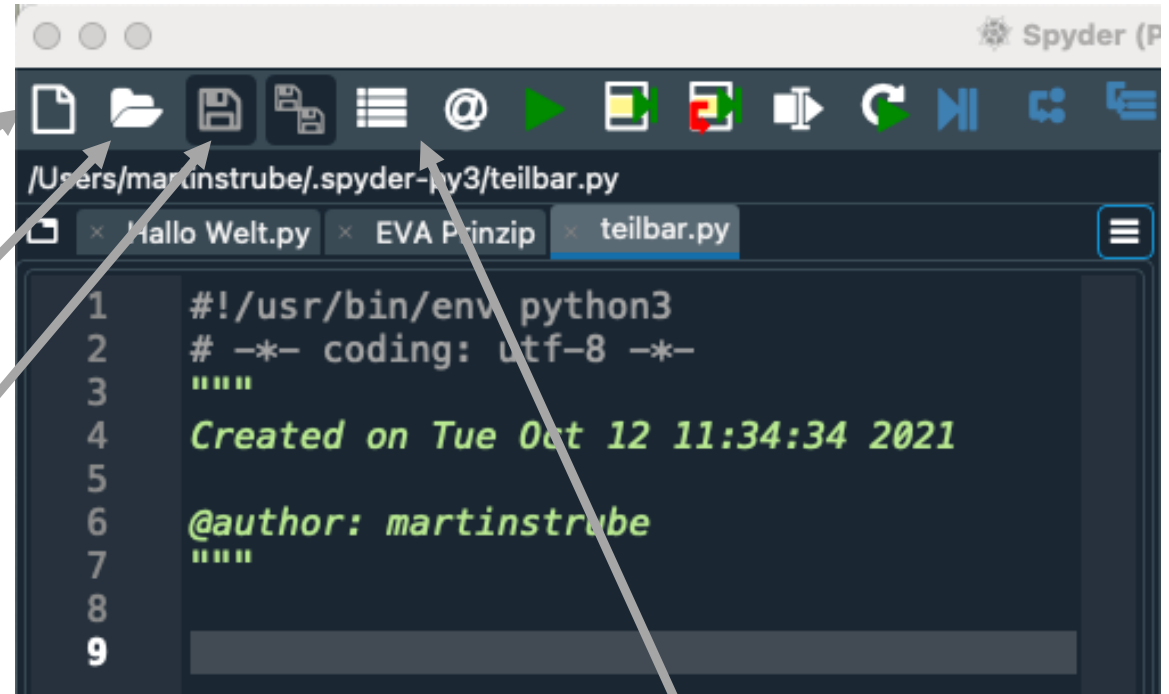
- Nun sehen Sie die 3 Programmdateien in Form von „Reitern“ in Ihrem Editor-Fenster.

- Gleiche Aktionen über Symbolleiste möglich.

- eine neue python-Datei erstellen

- eine bestehende python-Datei öffnen

- eine python-Datei speichern



- Wechsel zwischen den Reitern durch Klick auf einen Reiter oder über dieses Symbol

## Erste Schritte – Kommentare

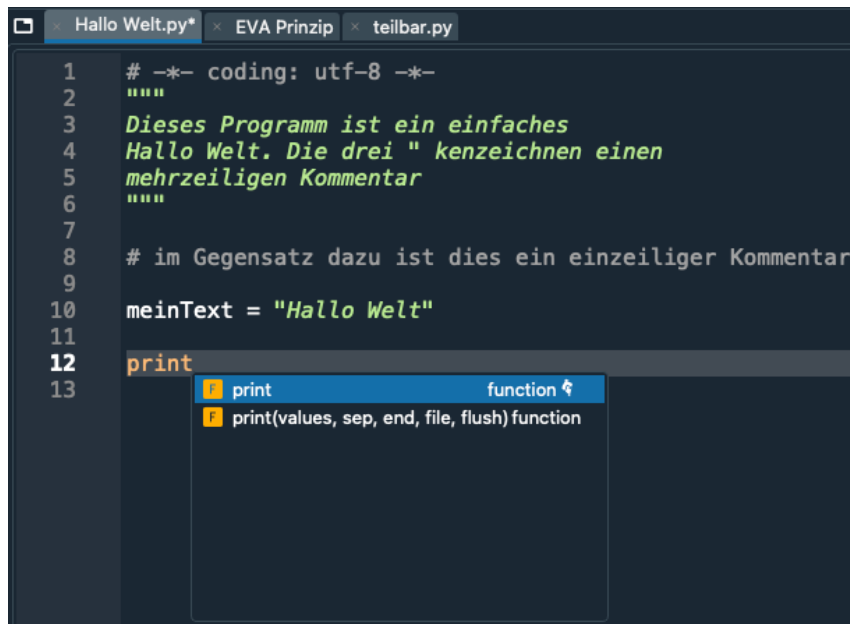
- Kommentare werden in Spyder in der Standardkonfiguration in zwei verschiedenen Farben dargestellt. Abhängig davon, ob es ein ein- oder mehrzeiliger Kommentar ist.

```
1  # -*- coding: utf-8 -*-
2  """
3  Dieses Programm ist ein ei
4  Hallo Welt. Die drei " ken
5  mehrzeiligen Kommentar
6  """
7
8  # im Gegensatz dazu ist di
```



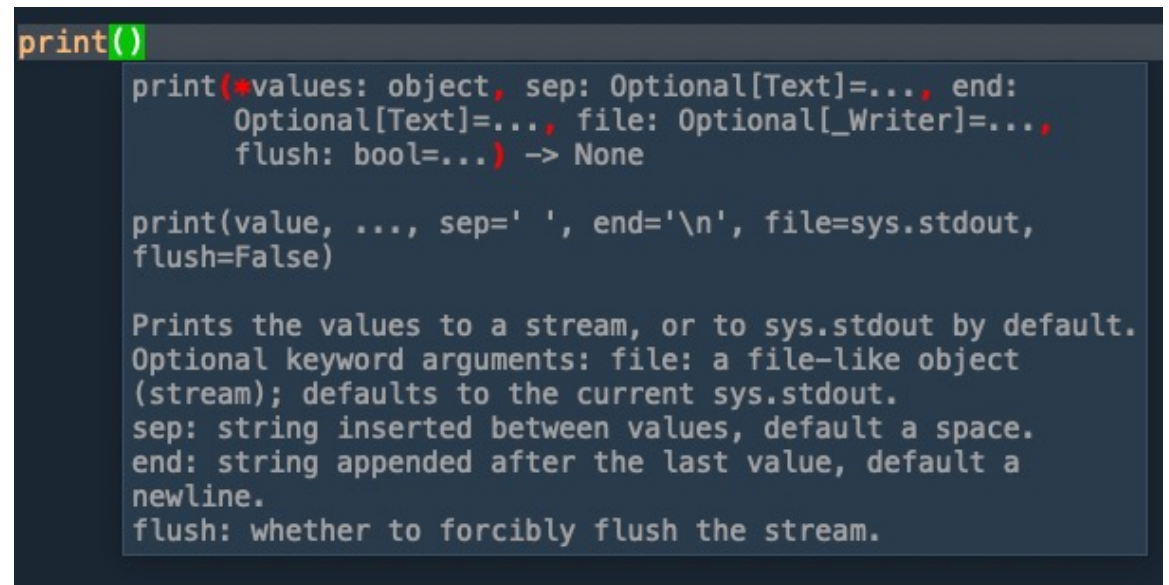
## Erste Schritte – automatische Hilfe

- Wenn wir Schlüsselwörter benutzen, blendet uns Spyder sofort eine Hilfe ein (Tooltip)



```
1  # -*- coding: utf-8 -*-
2  """
3  Dieses Programm ist ein einfaches
4  Hallo Welt. Die drei " kenzeichnen einen
5  mehrzeiligen Kommentar
6  """
7
8  # im Gegensatz dazu ist dies ein einzeliger Kommentar
9
10 meinText = "Hallo Welt"
11
12 print
13
```

- Klicken wir auf einen Eintrag der Hilfe, so erfolgt eine Autovervollständigung (hier grün markiert)
- Darunter öffnet sich ein Hilfefenster mit weiteren Informationen.



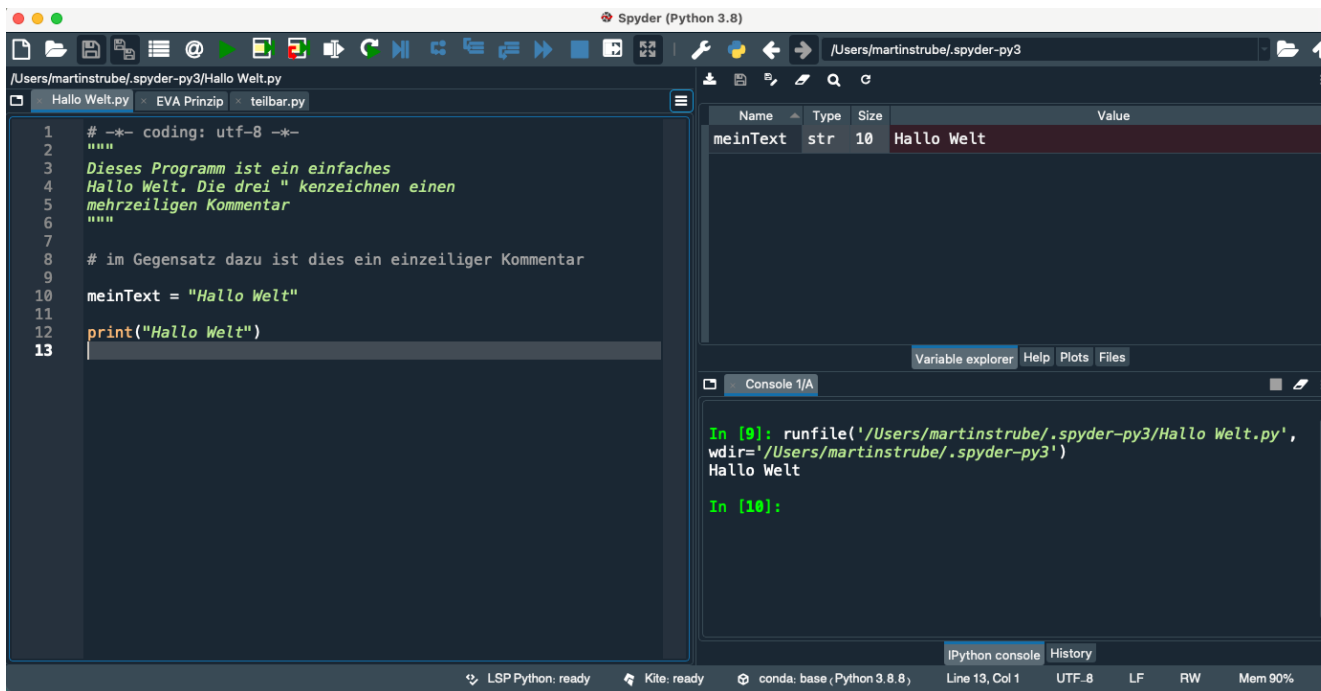
```
print()
print(*values: object, sep: Optional[Text]=..., end:
Optional[Text]=..., file: Optional[_Writer]=...,
flush: bool=...) -> None

print(value, ..., sep=' ', end='\n', file=sys.stdout,
flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments: file: a file-like object
(stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a
newline.
flush: whether to forcibly flush the stream.
```

## Erste Schritte – ein Programm ausführen

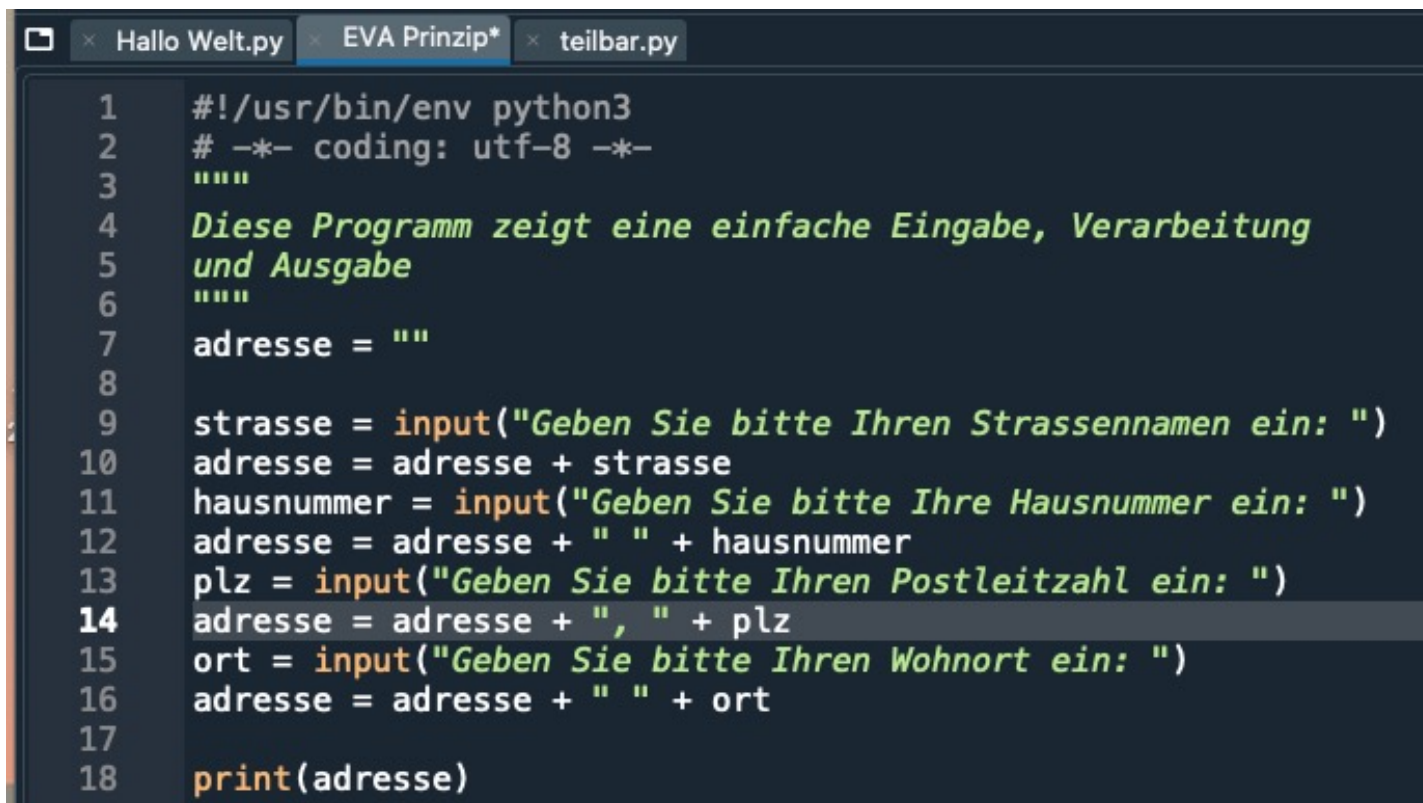
- Um ein Programm auszuführen, kann man auf den grünen Pfeil klicken und die gewünschte Ausgabekonsole wählen. Wir verwenden die Console der IDE und damit erscheint das Ergebnis im Consolen-Fenster unten rechts.



- Alternativ kann die Programmausführung mit der Taste F5 gestartet werden.

## Erste Schritte – der Variablenexplorer

- Erstellen Sie zunächst das folgende Beispielprogramm und führen Sie das Programm über Run aus.
- Aktivieren Sie vorher im Feld rechts oben den Reiter Variable Explorer.



The screenshot shows a Python IDE with three tabs: 'Hallo Welt.py', 'EVA Prinzip\*', and 'teilbar.py'. The 'EVA Prinzip\*' tab is active, displaying a Python script. The script starts with a shebang and a UTF-8 encoding declaration. It contains a multi-line comment in German describing the program's purpose. The code then initializes an empty string 'adresse' and uses 'input()' to collect the street name, house number, postal code, and location, concatenating them into the 'adresse' variable. Finally, it prints the complete address.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Diese Programm zeigt eine einfache Eingabe, Verarbeitung
5  und Ausgabe
6  """
7  adresse = ""
8
9  strasse = input("Geben Sie bitte Ihren Strassennamen ein: ")
10 adresse = adresse + strasse
11 hausnummer = input("Geben Sie bitte Ihre Hausnummer ein: ")
12 adresse = adresse + " " + hausnummer
13 plz = input("Geben Sie bitte Ihren Postleitzahl ein: ")
14 adresse = adresse + ", " + plz
15 ort = input("Geben Sie bitte Ihren Wohnort ein: ")
16 adresse = adresse + " " + ort
17
18 print(adresse)
```

## Erste Schritte – Der Variablen Explorer

- Im Fenster Variable Explorer sehen Sie nun die Inhalte, die jeder einzelnen Variablen zugewiesen wurden.

Name	Type	Size	Value
adresse	str	43	Salzdahlumer Strasse 46, 38302 Wolfenbüttel
hausnummer	str	2	46
ort	str	12	Wolfenbüttel
plz	str	5	38302
strasse	str	20	Salzdahlumer Strasse

Variablenname

Datentyp

Speicherbedarf

„Wert“ der Variablen

- Der Variablenexplorer hilft Ihnen, den Überblick über den Inhalt Ihrer Variablen zu behalten.
- Über den Variablen Explorer können Sie auch Wertzuweisungen ändern. Hierzu Rechtsklick auf den Variablennamen und *Edit* wählen.

## Erste Schritte – Zusammenspiel Console und Variablenexplorer

- Über die Console kann nun direkt auf die Variablen zugegriffen werden – entweder zum Ändern der Wertzuweisung oder zum Abrufen der gespeicherten Werte.

The screenshot displays the Jupyter Notebook interface. At the top, the 'Variable explorer' tab is active, showing a table of variables:

Name	Type	Size	Value
adresse	str	43	Salzdahlumer Strasse 46, 38302 Wolfenbüttel
hausnummer	str	2	46
ort	str	12	Wolfenbüttel
plz	int	1	25252
strasse	str	20	Salzdahlumer Strasse

Below the variable explorer, the 'Console 1/A' tab is active, showing the following IPython console output:

```
In [26]: plz = 25252
In [27]: print(ort)
Wolfenbüttel
In [28]: |
```

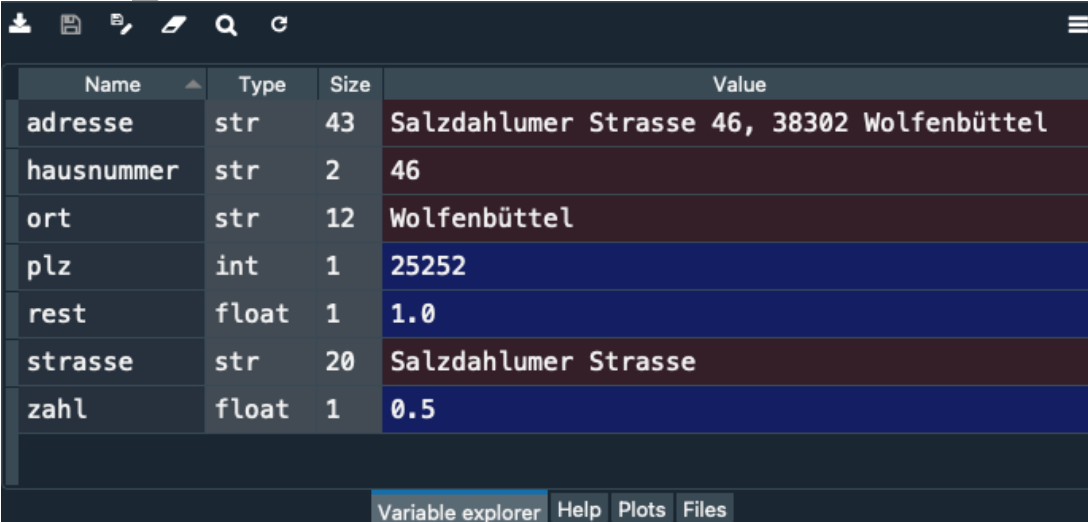
Wertzuweisung

Ausgabe eines Variablenwertes

## Erste Schritte –Variable explorer und Console löschen

- Wie Sie am letzten Beispiel gesehen haben, stehen die Variablenwerte auch nach Ende der Programmausführung weiter zur Verfügung.

Löschen per Klick auf  
das Radiergummi



Name	Type	Size	Value
adresse	str	43	Salzdahlumer Strasse 46, 38302 Wolfenbüttel
hausnummer	str	2	46
ort	str	12	Wolfenbüttel
plz	int	1	25252
rest	float	1	1.0
strasse	str	20	Salzdahlumer Strasse
zahl	float	1	0.5

Variable explorer Help Plots Files

- Die Einträge der Console können Sie über den Befehl `clear` löschen, indem Sie diesen Befehl direkt in der Konsole eingeben.

# Erste Schritte – der Debugger

- Schreiben Sie bitte zunächst das folgende Programm:

```
Hallo Welt.py  EVA Prinzip  teilbar.py*
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Dieses Programm ermittelt wie oft eine Zahl
5  ohne Rest durch 2 teilbar ist
6  """
7
8
9  # hier die Zahl
10 zahl = 256
11 print()
12 # mithilfe von while wird eine Bedingung überprüft
13 # solange die Bedingung erfüllt ist, wird der Code
14 # innerhalb der while-Schleife wiederholt ausgeführt.
15
16 # Startwert für die Variable rest festlegen
17 rest = 0
18
19 while(rest == 0):
20     rest = zahl%2
21     zahl = zahl/2
22
23     print(zahl)
```

- Starten Sie das Programm im Anschluss jedoch nicht durch Aufruf von Run File!
- Schauen Sie sich zunächst den Tooltip der folgenden Symbole an. Dazu einfach den Mauszeiger über dem jeweiligen Symbol stehen lassen.



- Bitte noch keines der Symbole anklicken!

## Erste Schritte – der Debugger

- Der Debugger bietet uns folgende Möglichkeiten:



1. Jede Zeile einzeln ausführen und im Variablen Explorer beobachten, welche Wertzuweisungen entstehen. Mann nennt dies auch „durchsteppen“.
- bei jedem Klick wird genau eine Zeile Code ausgeführt
  - ein Pfeil am Beginn einer Code-Zeile zeigt, wo wir uns im Code aktuell befinden

```
16 # Startwert für die Variable rest festlegen
17 rest = 0
18
19 while(rest == 0):
20     rest = zahl%2
21     zahl = zahl/2
22
23     print(zahl)
```

- Aufgabe: Gehen Sie das Programm im Einzelschrittmodus durch und beobachten Sie den Variablen Explorer.



## Erste Schritte – der Debugger

- Der Debugger bietet uns folgende Möglichkeiten:

2. Setzen von Haltepunkten (Breakpoints) im Code. Durch einen Klick rechts neben einer Zeilennummer können Haltepunkte gesetzt werden.

Über das Symbol Doppelpfeil kann eine Ausführung des Codes bis zum nächsten Haltepunkt gestartet werden.



```
9   # hier die Zahl
10  zahl = 256
11  print()
12  # mithilfe von while wird eine Bedingung überprüft
13  # solange die Bedingung erfüllt ist, wird der Code
14  # innerhalb der while-Schleife wiederholt ausgeführt.
15
16  # Startwert für die Variable rest festlegen
17  rest = 0
18
19  while(rest == 0):
20      rest = zahl%2
21      zahl = zahl/2
22
23  print(zahl)
```

- Aufgabe: Experimentieren Sie mit dem Setzen von Breakpoints an verschiedenen Stellen im Code.

## Erste Schritte – der Debugger

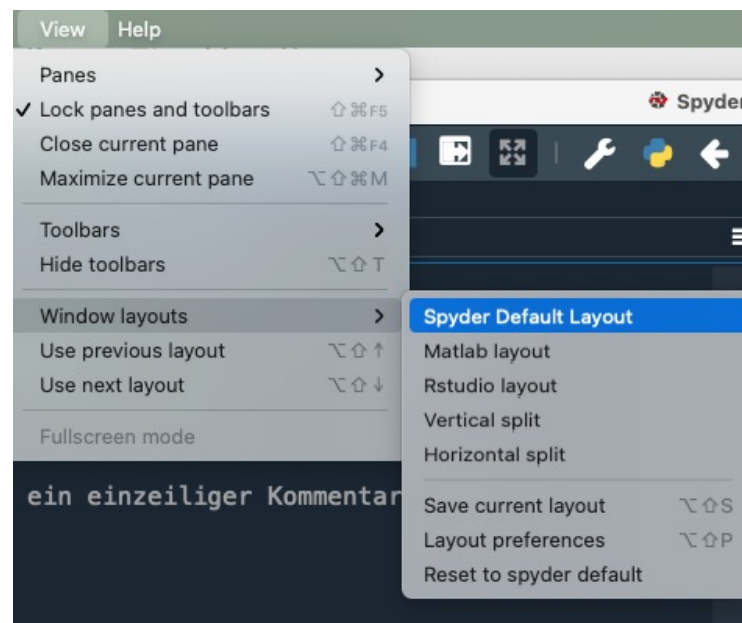
- Ausführlicher Informationen zu den Möglichkeiten des Debuggers finden Sie unter folgendem Link <https://docs.spyder-ide.org/current/panes/debugging.html>
- Sie können das Debugging jederzeit durch einen Klick auf folgendes Symbol beenden:



- Aufgabe: Öffnen Sie die folgende Datei aus Ihrem StudIP-Verzeichnis zur Laborveranstaltung: wasmachtdasprogramm.py und nutzen Sie die Debugger-Funktion, um den Programmablauf zu verstehen. Vervollständigen Sie anschließend alle Kommentare in diesem Programm.

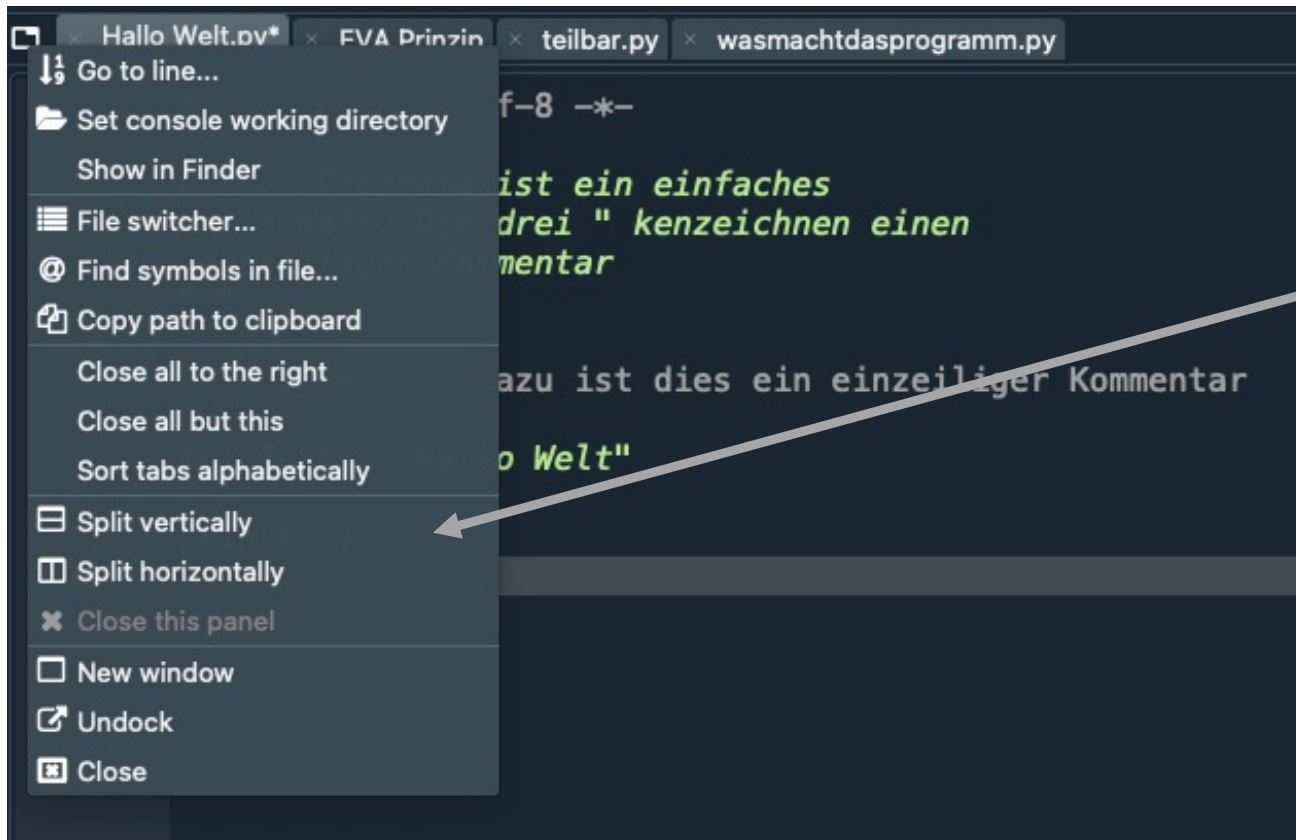
## das individuelle Fensterlayout

- Je nach Projekt interessieren Sie sich vielleicht für unterschiedliche Ansichten, um z.B. mehrere Programme gleichzeitig zu betrachten oder Plots und den Variablen Explorer zeitlich zu sehen.
- Hierzu bietet Ihnen Spyder sehr viele Möglichkeiten zu Anpassung des Fensterlayouts. Ganz wichtig: Sie können die ursprüngliche Ansicht immer wiederherstellen. Folgen Sie hierzu folgendem Menüpfad:



## das individuelle Fensterlayout

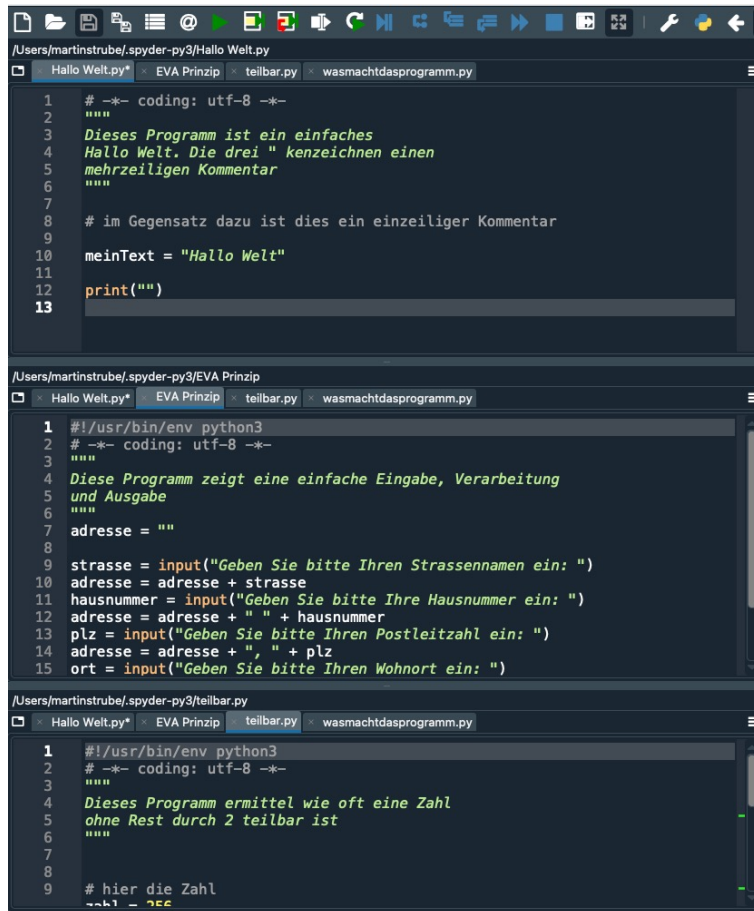
- Ein Rechtsklick auf das kleine Fenstersymbol öffnet das Menü zur Anpassung der Ansichten:



Oft ist es hilfreich, das Editorfenster zu unterteilen, damit man sich mehrere Programme zeitgleich anschauen kann.

## das individuelle Fensterlayout

- Ein Rechtsklick auf das klein Fenstersymbol öffnet das Menü zur Anpassung der Ansichten:



The screenshot displays the Spyder Python IDE interface with a custom window layout. The top toolbar includes a small window icon (a square with a dot) on the far right. Below the toolbar, the window title bar shows the path `/Users/martinstrube/spyder-py3/Hallo Welt.py`. The main editor area is divided into three horizontal panes, each containing a Python script. The first pane (top) shows a simple program with a multi-line comment and a print statement. The second pane (middle) shows a program that takes user input for address, street, house number, postal code, and location. The third pane (bottom) shows a program that calculates how many times a number is divisible by 2 without a remainder. The file explorer on the left shows the open files: `Hallo Welt.py*`, `EVA Prinzip`, `teilbar.py`, and `wasmachtdasprogramm.py`.

```
1  -*- coding: utf-8 -*-
2  """
3  Dieses Programm ist ein einfaches
4  Hallo Welt. Die drei " kennzeichnen einen
5  mehrzeiligen Kommentar
6  """
7
8  # im Gegensatz dazu ist dies ein einzeliger Kommentar
9
10 meinText = "Hallo Welt"
11
12 print("")
13
```

```
1  #!/usr/bin/env python3
2  -*- coding: utf-8 -*-
3  """
4  Dieses Programm zeigt eine einfache Eingabe, Verarbeitung
5  und Ausgabe
6  """
7  adresse = ""
8
9  strasse = input("Geben Sie bitte Ihren Strassennamen ein: ")
10 adresse = adresse + strasse
11 hausnummer = input("Geben Sie bitte Ihre Hausnummer ein: ")
12 adresse = adresse + " " + hausnummer
13 plz = input("Geben Sie bitte Ihre Postleitzahl ein: ")
14 adresse = adresse + ", " + plz
15 ort = input("Geben Sie bitte Ihren Wohnort ein: ")

```

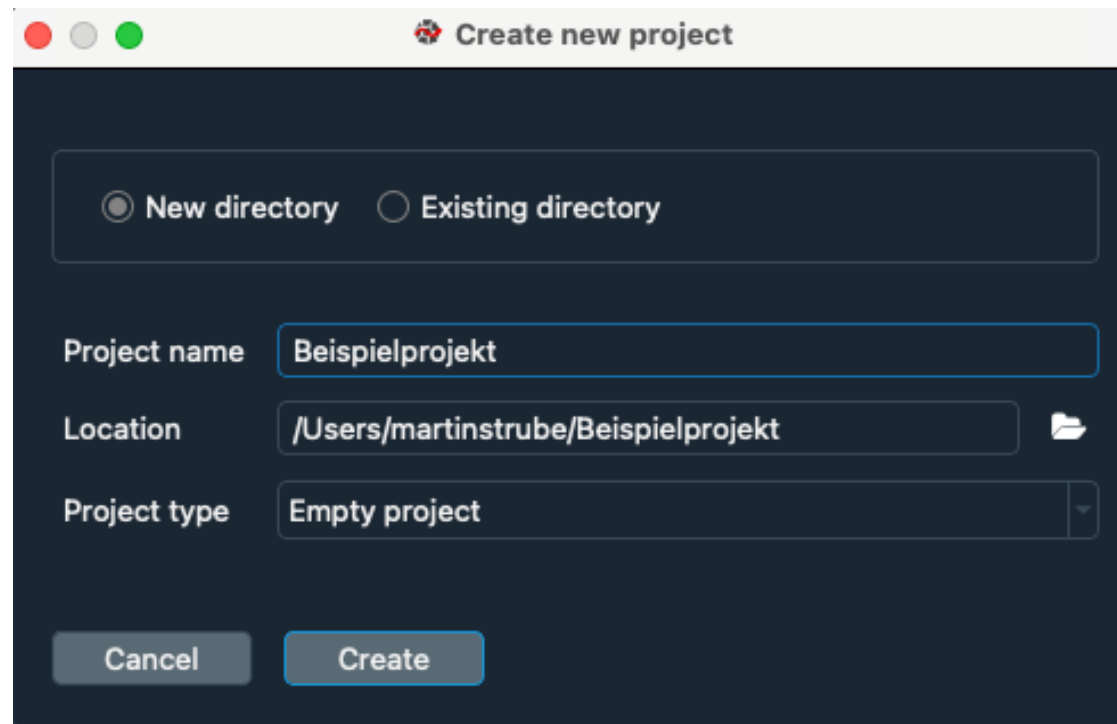
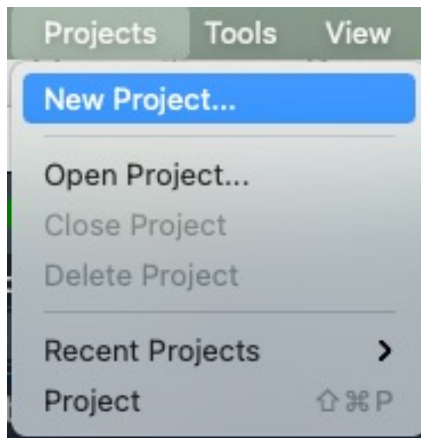
```
1  #!/usr/bin/env python3
2  -*- coding: utf-8 -*-
3  """
4  Dieses Programm ermittelt wie oft eine Zahl
5  ohne Rest durch 2 teilbar ist
6  """
7
8  # hier die Zahl
9  zahl = 256

```

In diesem Beispiel wurde das Editorfenster in drei Teile untergliedert.

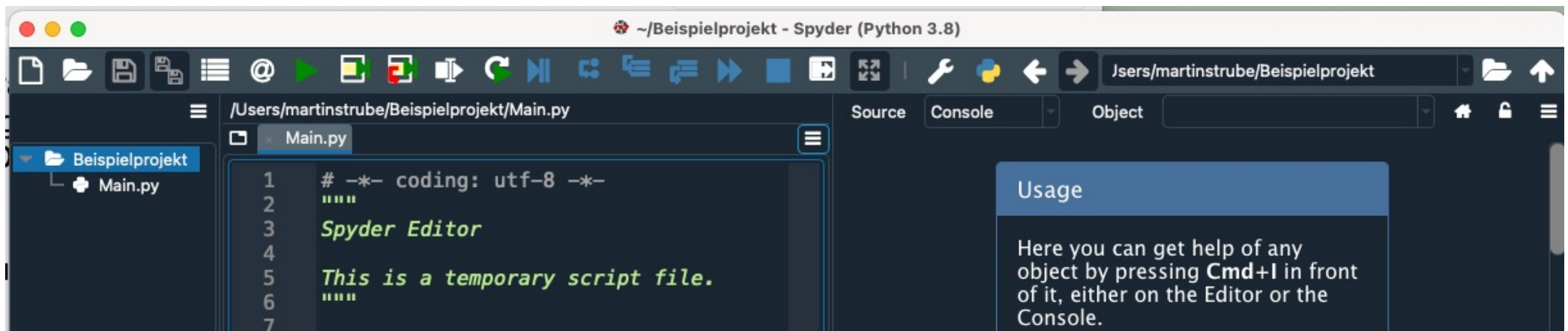
## Ein Projekt anlegen

- Sie haben in der letzten Woche bereits gesehen, dass Sie Ihre Programmiervorhaben mithilfe von Projekten strukturieren können.
- Das funktioniert natürlich auch in Spyder. Über das Menü Projects können Sie neue Projekte anlegen oder bestehende Projekte öffnen.



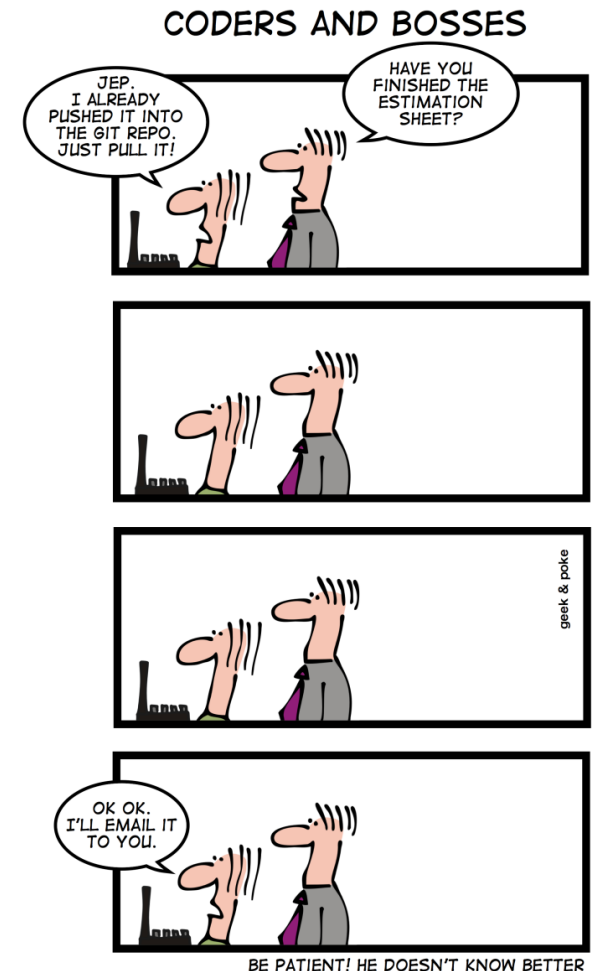
## Ein Projekt anlegen

- Sobald Sie ein Projekt angelegt oder geöffnet haben, wird auch der Projektexplorer ganz links eingeblendet. Über den Projektexplorer organisieren Sie alle Dateien, die für ein Programmierprojekt von Bedeutung sind.



# Standardwerkzeuge des Programmierers – 6. Versionsverwaltung

- Spätestens bei der Arbeit in Teams muss eine Möglichkeit gefunden werden, die erzeugten Dateien sicher zu verwalten und im Team zu verteilen.
- Dabei muss jederzeit klar sein: **Wer** hat **was warum** geändert!
- Auch wenn man alleine ein etwas umfangreicheres Projekt bearbeitet, ist es sehr zu empfehlen, den Quellcode mit einer Versionsverwaltung zu pflegen.
- Die wichtigste Funktion dabei ist die Verfolgung von Änderungen über sog. **Diff-Tools**.
- Bekannte Vertreter von Versionsverwaltungssystemen sind:
  - Concurrent Versions System (CVS)
  - Subversion (SVN)
  - Git





# Versionsmanagement - Begriffe

- **Repository:**

Zentraler Datenspeicher bzw. Archiv, in dem die versionierten Dateien vorgehalten werden.

- **Revision:**

Ein eindeutiger benannter Zustand des Dateibaums im Repository, meist eine Nummer oder eine eindeutige Zeichenkette.

- **Commit:**

Veröffentlichung von Änderungen in einem Repository.

- **Tag:**

Ein expliziter Name für eine Revision.

- **Branch:**

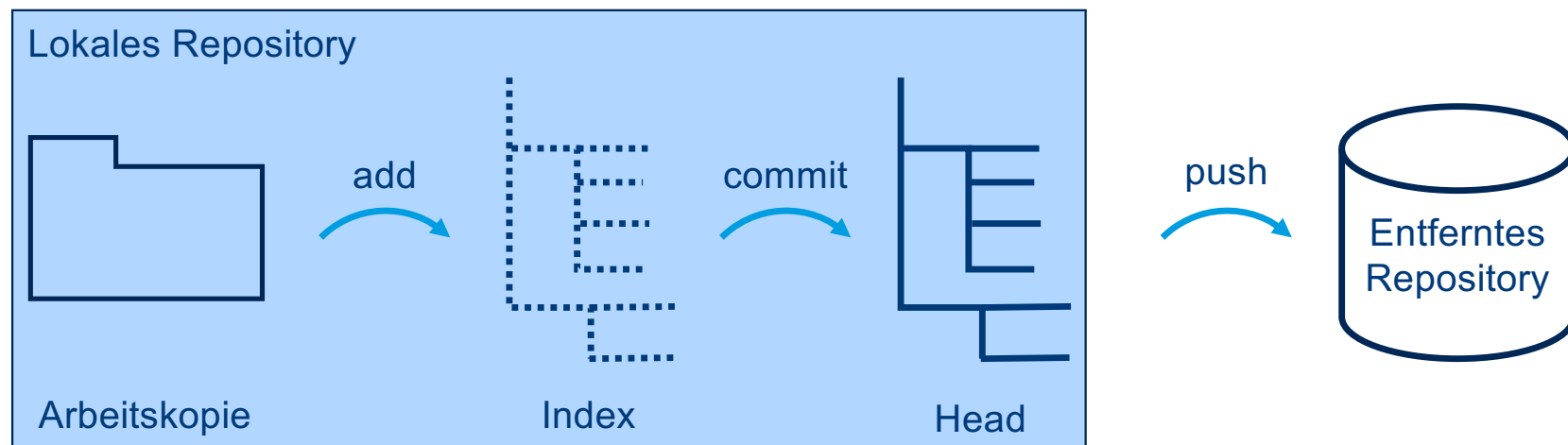
Ein Entwicklungslinie, die unabhängig von einer anderen Entwicklungslinie existiert.

- **Merge:**

Zusammenführen zweier unabhängiger Entwicklungslinien.

## Beispiel: Git-Workflow

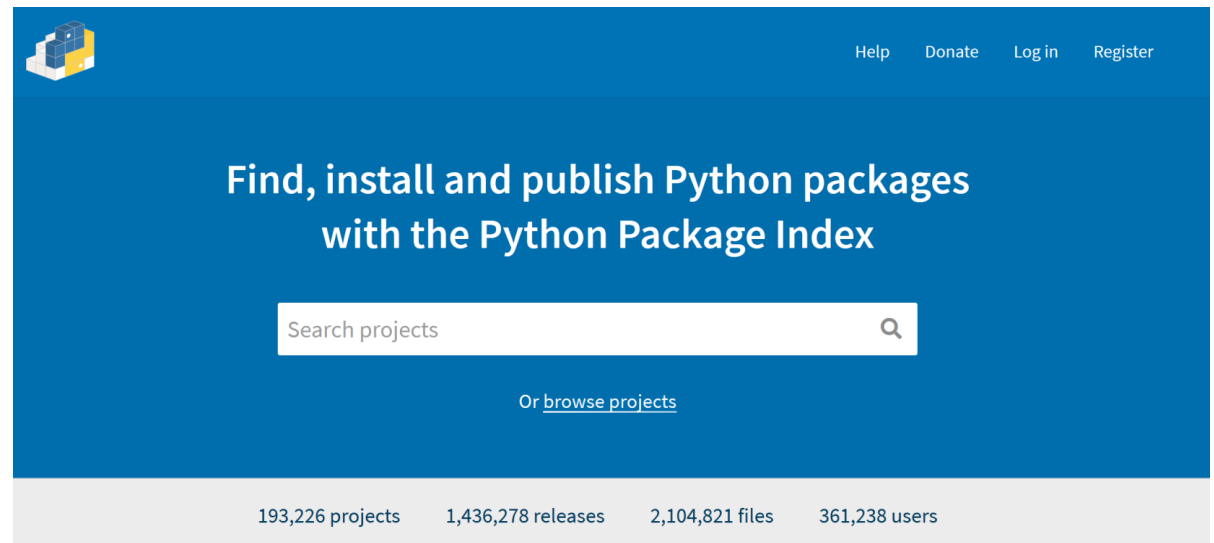
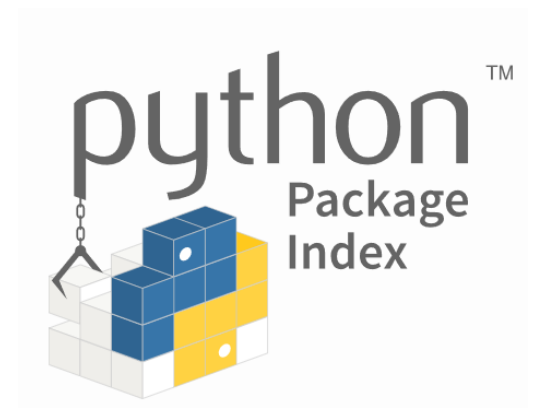
- Git gehört zur Gruppe der **verteilten Systeme**. Hier hat jeder Entwickler das gesamte Repository mit kompletter Historie lokal auf dem Entwicklungsrechner.
- Bei **zentralen Systemen** (CVS, SVN) ist nur der aktuelle Stand auf dem Rechner des Entwicklers.



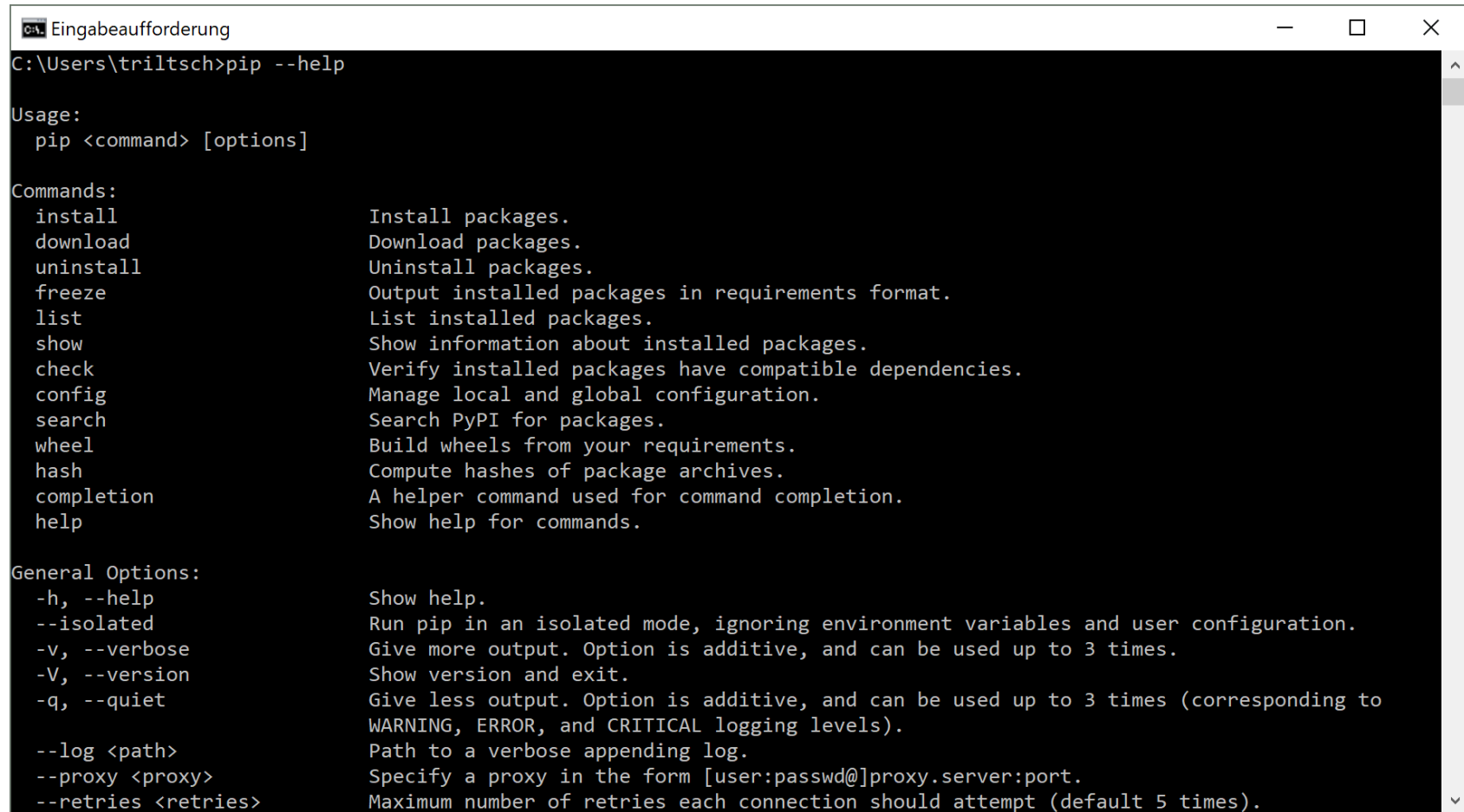
- Kurzanleitung zu git: <https://rogerdudler.github.io/git-guide/index.de.html>

# Python-Package-Index

- Ähnlich wie in den App-Stores der Smartphone-Betriebssysteme gibt es auch für Python-Entwickler die Möglichkeit, Ihre Entwicklungen als **Pakete** für andere Programmierer zur Verfügung zu stellen.
- Registrierte Entwickler können Ihre Entwicklungen als **Module** im **Python-Package-Index** zum direkten Download zur Verfügung stellen.
- Die Installation erfolgt mit einem speziellen Werkzeug, dem pip-Tool.
- Dieses ermöglicht sowohl eine einfache Installation, als auch das Aktualisieren der installierten Pakete über eine Update Funktion.



# Python Packages installieren – pip-Tool



```
C:\Users\triltsch>pip --help

Usage:
  pip <command> [options]

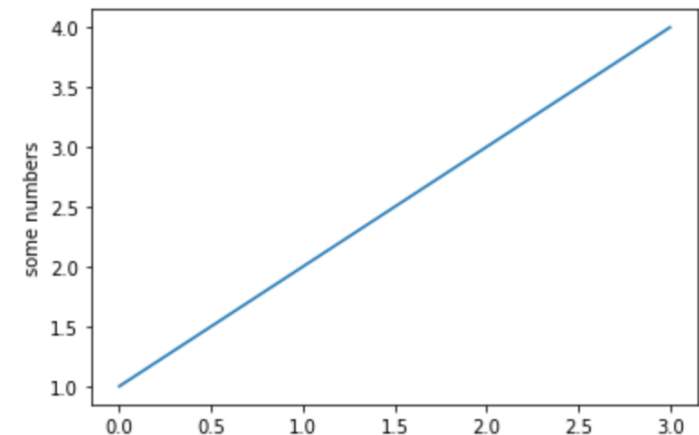
Commands:
  install           Install packages.
  download          Download packages.
  uninstall         Uninstall packages.
  freeze            Output installed packages in requirements format.
  list              List installed packages.
  show              Show information about installed packages.
  check             Verify installed packages have compatible dependencies.
  config            Manage local and global configuration.
  search            Search PyPI for packages.
  wheel             Build wheels from your requirements.
  hash              Compute hashes of package archives.
  completion        A helper command used for command completion.
  help              Show help for commands.

General Options:
  -h, --help        Show help.
  --isolated         Run pip in an isolated mode, ignoring environment variables and user configuration.
  -v, --verbose      Give more output. Option is additive, and can be used up to 3 times.
  -V, --version      Show version and exit.
  -q, --quiet        Give less output. Option is additive, and can be used up to 3 times (corresponding to
                     WARNING, ERROR, and CRITICAL logging levels).
  --log <path>      Path to a verbose appending log.
  --proxy <proxy>    Specify a proxy in the form [user:passwd@]proxy.server:port.
  --retries <retries> Maximum number of retries each connection should attempt (default 5 times).
```

## Standardbibliotheken (API)

- Standardbibliotheken sammeln Funktionalitäten, wie z.B.
  - mathematische Funktionen (**math**, <https://docs.python.org/3/library/math.html>),
  - Zufallszahlen (**random**, <https://docs.python.org/3/library/random.html>),
  - Umgang mit Datum und Uhrzeit (**time**, <https://docs.python.org/3/library/time.html>)
  - Diagrammerstellung (**matplotlib**, <https://matplotlib.org/users/index.html>).
- Diese müssen zur Verwendung in das eigene Programm eingebunden werden.
- In Python geschieht dies durch das Schlüsselwort **import**.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



# Beispiele und Übung zu Bibliotheken




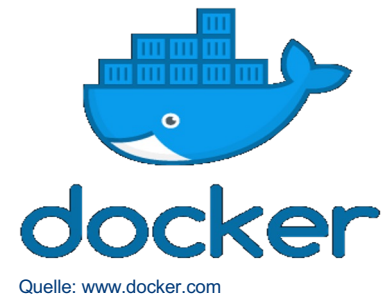
[https://github.com/Informatikvorlesung-Fak-M/Python-Beispiele/blob/master/Block\\_04/01\\_Bibliotheken.ipynb](https://github.com/Informatikvorlesung-Fak-M/Python-Beispiele/blob/master/Block_04/01_Bibliotheken.ipynb)



[https://mybinder.org/v2/gh/Informatikvorlesung-Fak-M/Python-Beispiele/master?filepath=%2FBlock\\_04%2F01\\_Bibliotheken.ipynb](https://mybinder.org/v2/gh/Informatikvorlesung-Fak-M/Python-Beispiele/master?filepath=%2FBlock_04%2F01_Bibliotheken.ipynb)

## Virtualisierung mit Containern – Geschützte Umgebungen für Programme bereitstellen

- Container haben die Aufgabe eine komplette Umgebung für Programme bereitzustellen.
- In diesen Containern ist also alles zusammengepackt, was zum Ausführen eines geschriebenen Programms dazugehört:
  - besondere Pakete oder Module, die vom Entwickler eingebunden wurden
  - spezielle Laufzeitmodule (z.B. Jupyter Notebook Server)
  - Systemwerkzeuge (Editoren, Compiler, ...)
- Solche Container können dann über das Internet an beliebigen Orte (Rechner) transportiert werden und dort stehen die Programme sofort lauffähig zur Verfügung.
- Eine weit verbreitete Software für diese Technologie stellt Docker dar.
- Eine konkrete Möglichkeit solche Container für Python mit Jupyter-Notebooks und git bereitzustellen bietet:  **binder**



# Fehlerkategorien

- **Syntaxfehler:**

- Die Regeln der Programmiersprache wurden verletzt.
- Falscher formaler Aufbau.
- „Schreibfehler“

- **Laufzeitfehler:**

- Diese Fehler treten erst beim Ausführen des fertigen Programmes auf.
- Denken Sie an den bzw. mit dem „**Dumbest Assumable User**“.
- Fehler treten oft erst lange nach der Inbetriebnahme (IBN) auf.

- **Logische Fehler (Semantikfehler):**

- Hier liegt kein Laufzeit- oder Syntaxfehler vor und dennoch liefert das Programm nicht die gewünschten Ergebnisse.
- Ursache sind häufig Flüchtigkeitsfehler oder Fehler in der logischen Struktur des Quellcodes.



# Übung: Fehlersuche



[https://github.com/Informatikvorlesung-Fak-M/Python-Beispiele/blob/master/Block\\_04/03\\_Debugger.ipynb](https://github.com/Informatikvorlesung-Fak-M/Python-Beispiele/blob/master/Block_04/03_Debugger.ipynb)



[https://mybinder.org/v2/gh/Informatikvorlesung-Fak-M/Python-Beispiele/master?filepath=%2FBlock\\_04%2F03\\_Debugger.ipynb](https://mybinder.org/v2/gh/Informatikvorlesung-Fak-M/Python-Beispiele/master?filepath=%2FBlock_04%2F03_Debugger.ipynb)