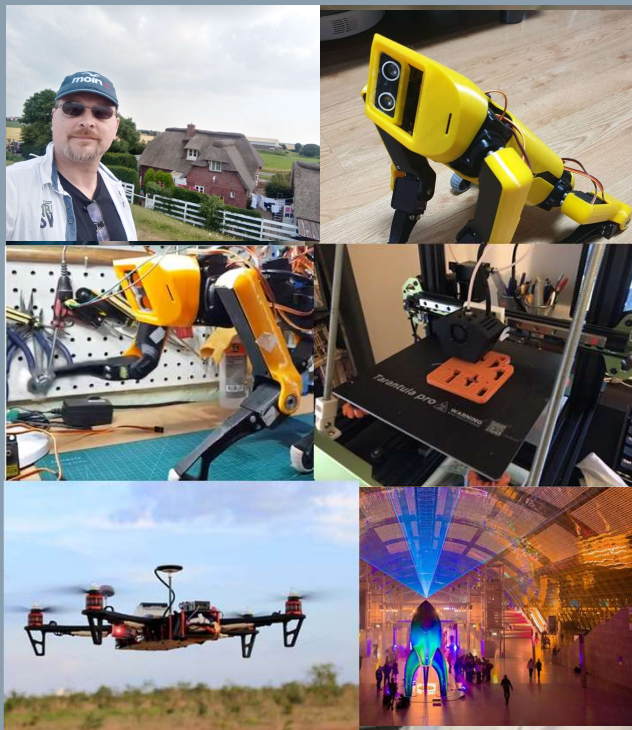


Python Programmieren

Zum Referenten:



Andreas Schmidt

- Jahrgang 1975, wohnhaft in Hamburg
- Fachinformatiker für Systemintegration
- Android-App Entwickler
- Java Entwickler
- Administrator für Heterogene Netzwerke
- Kommunikationselektroniker
- Über 20 Jahre im IT-Support und Consulting tätig
- ITIL
- Hardware-Entwicklung



Python – Warum?

- Wer Programmieren lernen will, muss sich ganz zu Beginn eine schwierige Frage stellen:
 - Welche Programmiersprache sollte ich überhaupt wählen?

Python – Warum?

- Wir wollen uns mit der Frage befassen, warum es sinnvoll ist, das Programmieren zu erlernen und warum Python sehr wahrscheinlich die richtige Wahl ist.
- Ganz zu Beginn dieses Workshops steht die Frage, weshalb du überhaupt damit anfangen solltest, das Programmieren zu erlernen.
- Für viele Menschen lautet die Antwort:
 - um die Karrierechancen zu verbessern.

Python – Warum?

- Programmierer erfreuen sich einer enormen Nachfrage, sodass die Löhne für diese Berufsgruppe ausgesprochen hoch sind.
- Doch selbst wenn du nicht als Programmierer arbeiten willst, sind Grundkenntnisse in diesem Bereich sehr wichtig.
- Die Digitalisierung umfasst immer weitere Wirtschaftsbereiche, so- dass gute Computerkenntnisse mittlerweile in fast allen Branchen eine bedeutende Rolle spielen.
- Daher stellen Programmierkenntnisse in vielen Berufen eine hilfreiche Zusatzqualifikation dar, die man auf keinen Fall unterschätzen sollte.

Python – Warum?

- Neben den Karrieremöglichkeiten stellt das Programmieren aber auch eine interessante intellektuelle Herausforderung dar:
 - Wenn du auf ein Alltagsproblem triffst, ist es sehr interessant, ein Programm zu erstellen, um dieses zu lösen.
 - Das ist eine hervorragende Denksportaufgabe und eröffnet auf viele Situationen eine völlig neue Sichtweise, die du im Laufe dieses Kurses immer wieder einnehmen wirst.
- Python eignet sich sowohl für professionelle Programmierer als auch für Anfänger, die ihre erste Programmiersprache erlernen.
- Die Programmiersprache bietet einen sehr großen Funktionsumfang, sodass es möglich ist, damit Software zu erstellen, die allen Anforderungen der Wirtschaft entspricht.
- Daher kommt Python in diesem Bereich sehr häufig zum Einsatz.
- Gleichzeitig ist die Programmiersprache sehr einfach strukturiert, sodass sie sich hervorragend für Anfänger eignet.

Python - Warum

- Wenn du dieses Workshop besuchst und die praktischen Programme aus den zugehörigen Einheiten durchführst, lernst du von Grund auf, mit Python zu programmieren.
- So wirst du am Ende der 25 Tage in der Lage sein,
 - für jede mögliche Problemstellung oder Idee eigenständig ein Programm zu coden
 - die entsprechenden Ansätze kennen, um das Problem mithilfe des Internets und verschiedenen Programmier- Tools Schritt für Schritt zu lösen.

Python - Warum

- Die Zahl an Programmiersprachen ist riesig.
- Namen wie C, C++, Java, PHP, Ruby oder Swift sind dir vielleicht bereits ein Begriff.
- Hinzu kommen hunderte weiterer Programmiersprachen.
- Daher stellt sich die Frage, weshalb du ausgerechnet mit Python anfangen solltest

Python - Vorteile

- Nachfrage:
 - Ein besonders wichtiger Grund für diese Entscheidung ist die hohe Nachfrage nach Python Programmierern.
 - Python ist sehr mächtig und ermöglicht es, für fast alle Probleme im Bereich der Informatik eine passende Lösung zu entwickeln.

Python - Vorteile

- Effizienz.
 - Zum anderen arbeiten Python-Programmierer sehr effizient.
 - Die einfache Struktur des Programmcodes sorgt dafür, dass die Software vergleichsweise schnell entsteht.
 - Aufgrund dieser Vorteile wird Python als Programmiersprache in einer Vielzahl von Organisationen eingesetzt.
 - Die CIA nutzt Python zum hacken,
 - Google um Websites zu crawlen
 - Spotify, um seinen Nutzern Songs zu empfehlen.
 - Sogar die NASA nutzt Python.
- Aber keine Sorge, mit Python zu programmieren ist bei weitem keine Raketenwissenschaft.

Python - Vorteile

- Einfachheit.
 - Tatsächlich ist ein weiterer Grund, der für Python spricht, die Einfachheit und Flexibilität dieser Programmiersprache.
 - Im Vergleich zu anderen Programmiersprachen musst dich bei Python deutlich weniger um strikte Formalitäten kümmern und kannst dich allein auf den Code fokussieren.
 - Das führt zu schnellen Ergebnissen, mit relativ wenig Aufwand.

Python - Vorteile

- Zum Vergleich:
- Java:

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
print(„Hello, World“)
```

Python - Vorteile

- Verbreitung.
 - All diese Vorteile führen dazu, dass auch erfahrene Programmierer Python immer öfter zur Programmiersprache ihrer Wahl machen.
 - Python hat eine riesige Community (häufig fällt der Begriff Pythonista, um die Mitglieder der Community zu beschreiben)
 - es gibt für beinahe jede Problemstellung zahlreiche Lösungsansätze in den zahlreichen Python-Foren.

Python - Vorteile

- Relevanz.
 - Python hat inzwischen Java als beliebteste Programmiersprache abgelöst und wird wahrscheinlich in wenigen Jahren auch die Programmiersprache sein, die am weitesten verbreitet ist.
 - Diese Entwicklung ist einerseits durch die Effizienz und Einfachheit von Python begründet, aber auch durch die Vielfältigkeit der Einsatzgebiete der Programmiersprache.
 - Verwendet wird Python unter anderem für
 - die Entwicklung von Desktop-Programmen und Applikationen,
 - den Aufbau von Web-Applikationen,
 - für zahlreiche Projekte rund um den Bereich Artificial Intelligence (insbesondere Machine Learning)
 - bei der Automatisierung von Tasks.
- Das alles sind Technologien, die die Zukunft maßgeblich formen werden.

Python - Philosophie

- Die im vorherigen Kapitel beschriebenen Aspekte hängen eng mit der Entwicklungsgeschichte von Python und der Philosophie, die hinter dieser Programmiersprache steckt, zusammen.
- Dabei handelt es sich um eine relativ junge Programmiersprache, die erst zu Beginn der 90er Jahre entstand.
- Für die Entwicklung verantwortlich war der Niederländer Guido van Rossum, der zu dieser Zeit am Centrum Wiskunde & Informatica in der niederländischen Stadt Amsterdam arbeitete.
- Um den Studenten das Programmieren beizubringen, verwendete dieses Institut damals die Programmiersprache ABC.
- Diese war sehr einfach strukturiert.
- Allerdings wies sie zu große Unterschiede zu den gängigen Programmiersprachen – insbesondere zu der älteren und bekannteren Programmiersprache C – auf.
- Dies führte dazu, dass die Akzeptanz von ABC nicht besonders hoch war.

Python - Philosophie

- Van Rossum beschloss daraufhin eine neue Programmiersprache, namens Python, zu entwickeln.
- Diese sollte, wie auch ABC, als Lehrsprache dienen und so einfach wie möglich aufgebaut sein.
- Python verwendet noch heute eine vergleichsweise unkomplizierte Syntax. (Syntax bezeichnet die formellen Regeln zur Gestaltung eines Programms.)
- Darüber hinaus ist Python-Code leicht verständlich, da sich die Befehle vorwiegend an der englischen Sprache orientieren.
- Das macht es gerade für Anfänger einfacher, den Sinn des Codes zu verstehen.
- Außerdem verzichtet Python so weit wie möglich auf Klammern und ähnliche Elemente, die bei vielen anderen Programmiersprachen für die Strukturierung der Programme notwendig sind.
- Hierdurch wird eine weitere häufige Fehlerquelle beseitigt.

Python - Philosophie

- Der Erfolg von Python war überwältigend.
- Als Ergebnis bildete sich sehr schnell ein großes Team aus frei- willigen Mitarbeitern heraus, die Python weiterentwickelten und dies bis heute tun, wodurch die Programmiersprache ständig verbessert wird und neueste Entwicklungen umsetzt.
- Van Rossum wirkte dabei lange Zeit in entscheidender Position mit
 - er hatte bei allen wichtigen Entscheidungen das letzte Wort.
 - Im Juli 2018 gab er bekannt, sich von dieser Funktion zurückzuziehen.
 - Dennoch arbeitet er nach wie vor an der Weiterentwicklung von Python mit.

Python2 vs Python3

- Seit die erste Vollversion von Python 1994 erschienen ist, hat sich viel getan.
- Die Entwicklergemeinschaft hat viele zusätzliche Funktionen eingefügt, die das Programmieren erleichtern.
- Mit der Version 2 von Python kam beispielsweise der Garbage-Collector, ein Feature zur Reduzierung des Speicherplatzes und zur Erhöhung der Effizienz, hinzu.

Python2 vs Python3

- 2008 erschien mit Python 3 die dritte Version der Programmiersprache.
- Dabei entschieden sich die Entwickler dazu, auf eine Kompatibilität zum bisherigen Code zu verzichten,
 - um von den Vorteilen von Python 3
 - Verbesserte Unterstützung von Unicode-Zeichen;
 - True Division: Ergebnisse der Division werden bei Bedarf als Kommazahl dargestellt;
 - Verbesserte Performance bei Computern mit mehreren Prozessoren,
 - usw.
 - Gebrauch zu machen.
- Das führt dazu, dass Programme, die in Python 2 geschrieben sind, in Python 3 häufig nicht mehr ausführbar sind.
- Dieser Workshop verwendet mit Python 3 stets die aktuelle Version und wird regelmäßig aktualisiert.

Python2 vs Python3

- Python 2 ist mittlerweile veraltet und kommt nur noch selten zum Einsatz.
- Wenn du im Internet oder in älteren Lehrbüchern nach weiterführenden Informationen suchst, dann solltest du diesen Unterschied trotzdem im Hinterkopf behalten.
- Wenn es sich bei den meisten entsprechenden Beispielen noch um Code in Python 2 handeln sollte, dann kannst du diesen mit einem Python-3-Interpreter nicht mehr ausführen.

(Was ein Python-Interpreter ist, erfährst du im nächsten Kapitel.)

Workspace anlegen

- Damit wir im weiteren Kursverlauf unsere Programme ablegen können, erstellen wir uns einen workspace.
- Auf Windows
 - `c:\users\<username>\Dokumente_Workspace\HalloWelt\`
- Auf Kali und dem B2 entsprechend:
 - `/home/<username>/_Workspace/HalloWelt/`
- Unter Windows legen wir uns ebenfalls eine kleine Batch-Datei an, die Unseren Workspace mit Kali und dem B2 Synchronisiert, also unseren Workspace uploaden!
- Diese nennen wir `python-sync-up.bat` und speichern darin 2 scp-Befehle, die uns diese Arbeit abnehmen.
 - Wie diese SCP-Befehle zu gestalten sind, wisst ihr noch aus dem vorherigen Kurs.

Interpreter und Compiler

- Wenn du ein Code-Programm schreibst, dann besteht der geschriebene Code aus reinem Text.
- Ein Computer selbst arbeitet hingegen mit elektrischen Impulsen, die als Maschinensprache bezeichnet werden.
- Damit ein Programm ausgeführt werden kann, ist es notwendig, den Programmcode von der Textform in die Maschinensprache zu übersetzen.

Interpreter und Compiler

- Interpreter und Compiler
- Für diese Übersetzung gibt es grundsätzlich zwei Möglichkeiten:
 - Du kannst einen Interpreter
 - oder einen Compiler verwenden.

Interpreter und Compiler

- Ein Compiler übernimmt die Übersetzung ein einziges Mal und erzeugt daraufhin ein ausführbares Programm in Maschinensprache.
- Ein Interpreter führt die Übersetzung hingegen jedes Mal aufs Neue aus, wenn du das Programm startest.
- Welche dieser beiden Alternativen zum Einsatz kommt, hängt von der Programmiersprache ab, die du verwendest.

Interpreter und Compiler

- Python ist eine interpretierte Programmiersprache.
- Das bedeutet, dass die Programme in Textform abgespeichert werden und bei jeder Ausführung erneut übersetzt werden.
- Das reduziert zwar die Effizienz, da diese Übersetzung etwas Zeit benötigt.
- Doch profitierst du davon, dass sich die Programme auf jedem Rechner ausführen lassen, auf denen ein entsprechender Interpreter installiert ist.
- Daher eignet sich Python hervorragend für plattformübergreifende Programme.

Interpreter und Compiler

- Um die Programme, die du mit Python schreiben wirst, selbst auszuprobieren, benötigst du also einen Python-Interpreter.
- Diesem kannst du auf folgender Seite kostenlos herunterladen:
 - <https://www.python.org/downloads/>

Python installieren

- Für Windows:



Python installieren

- Auf dieser Seite musst du lediglich auf die gelbe Schaltfläche (Download Python) klicken, um mit dem Download zu beginnen.
- Dabei findest du passende Versionen für alle gängigen Betriebssysteme:
 - Windows, macOS, Linux und einige weitere.
- Die Voraussetzungen für dieses Programm sind nicht anspruchsvoll.
- Wenn du ein aktuelles Betriebssystem verwendest, sollte die Installation keine Probleme bereiten.
- Nachdem du die Schaltfläche betätigt hast, wird zunächst der Installations-Assistent heruntergeladen.
- Diesen musst du daraufhin anklicken, um mit der Installation zu beginnen.

Python installieren

- Beim Fenster, das sich nun öffnet, ist es wichtig, ganz unten die Checkbox mit der Bezeichnung **Add Python X.X to PATH** auszuwählen.
- Tust du das nicht, ist Python nur in dem Verzeichnis zugänglich, in dem es installiert wurde.
- Da es jedoch nicht sinnvoll ist, alle Programme im gleichen Verzeichnis abzuspeichern, ist es notwendig, den Python-Interpreter auch aus anderen Bereichen zugänglich zu machen.
- Dafür musst du die entsprechende Pfadvariable hinzuzufügen.

Python installieren

- Solltest du das Anklicken der Checkbox bei der Installation vergessen haben, ist es auch möglich, die Pfadvariablen manuell einzufügen.
- Da das jedoch etwas kompliziert ist, ist es empfehlenswert, das Programm wieder zu löschen und daraufhin erneut zu installieren
 - – dieses Mal mit angeklickter Checkbox.

Python installieren

- Unter Linux gestaltet sich die Installation noch einfacher.
- Hier steht dir die Paketverwaltung APT zur Seite!
- Hier ist es nur nötig :
 - „sudo apt install python3* -y“ auszuführen!
 - Nun werden alle benötigten Pakete heruntergeladen und installiert.

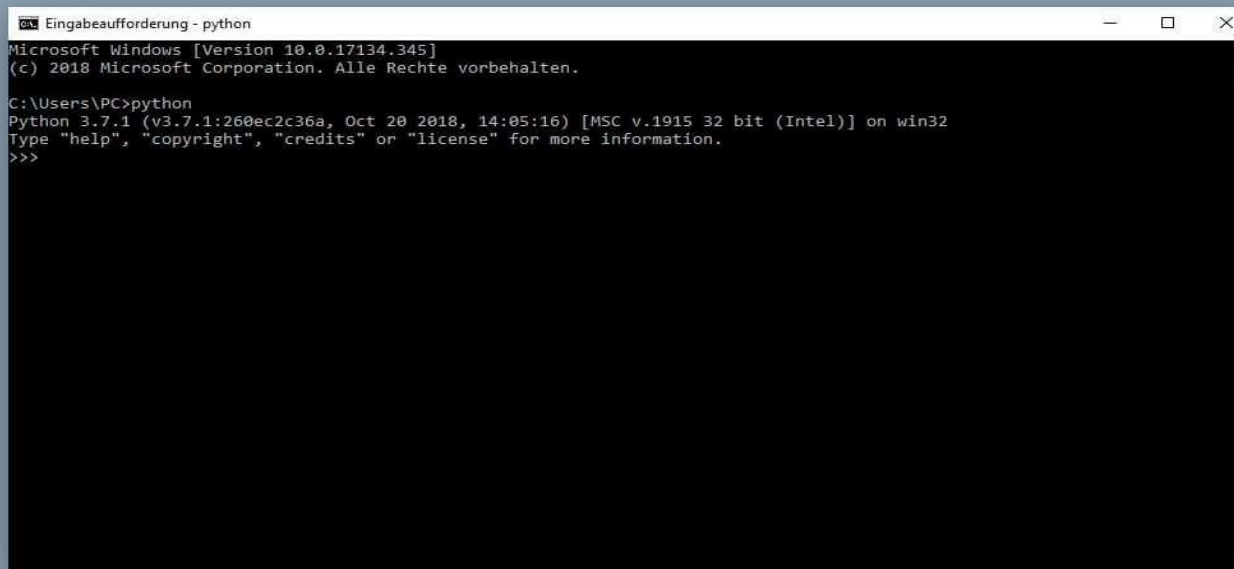
Python installieren

- Um zu überprüfen, ob du alles richtig installiert hast, kannst du einen Kommandozeileninterpreter öffnen.
- Diesen findest du unter Windows im Startmenü im Ordner Windows-System unter der Bezeichnung **Eingabeaufforderung**.
- Solltest du einen Linux-Rechner verwenden, musst du nach dem Terminal suchen, das unter anderem über den Shortcut Strg + Alt + T erreichbar ist, oder über z.B. per SSH direkt auf die Kommandozeile gehen.
- Im Betriebssystem macOS trägt der Kommandozeileninterpreter ebenfalls die Bezeichnung Terminal.
 - Öffne einfach die Suchfunktion (cmd + Leertaste) und suche nach Terminal.

Python installieren

- Um die korrekte Installation zu überprüfen, kannst du nun den Begriff **python** in den Kommandozeileninterpreter eingeben und mit **Enter** bestätigen.
- Hinweis: Die Eingabe bestätigst du immer mit **Enter** und springst dadurch automatisch in die nächste Zeile!
- Ist alles erfolgreich verlaufen, sollte dieser die installierte Version anzeigen.
- Ist der Interpreter nicht verfügbar, kommt es hingegen zu einer Fehlermeldung.

Python installieren



```
Eingabeaufforderung - python
Microsoft Windows [Version 10.0.17134.345]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\PC>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Screenshot: Wenn du den Interpreter richtig installiert hast, erscheint nach der Eingabe des Begriffs python die entsprechende Version.

Die Python Kommandozeile

- Nach der installation kann ein Interpreter, durch eingabe des Befehls „python“ oder „python3“, über die Kommandozeile gestartet werden. Dies haben wir auf der vorherigen Folie gesehen.
- In dieser Kommandozeile können wir **print(„Hallo, Welt!“)** eingeben und mit Enter bestätigen.
- Das Ergebnis ist die Ausgabe „Hallo, Welt!“
- Wir können also **Befehle** direkt testen oder anders gesagt in der Sprache **Python** mit dem **Interpreter** kommunizieren.

Die Python Kommandozeile

- Diese Kommandozeile werden wir auch immer einmal wider zum Testen von befehlen nutzen!
- Damit wir unsere eingegebenen Befehle auch wiederholt absetzen können, nutzen wir im späteren Verlauf, einen Editor.
- In dem Editor können wir kleine Programme schreiben und speichern.
- Diese Programme werden danach wie folgt ausgeführt:
 - „python Programmname.py“
 - Hierbei ist bei Programmname.py ggf der Volle Pfad zur Datei an zu geben!

Python IDLE

Integrated Development and Learning Environment

- Die Abkürzung IDLE steht für „Integrated Development and Learning Environment“
 - diese Entwicklungsumgebung ist bei einer normalen Python-Installation mit an Bord und muss nicht extra installiert werden.
 - Viele Programmierer kennen IDLE als eine integrierte Entwicklungsumgebung.
 - Bei Python wurde Wert daraufgelegt, dass es zum Lernen der Sprache der Einsteiger so einfach wie möglich tut.
 - Daher wurde bei der Entwicklung des Tools darauf geachtet, dass es den Einsteiger unterstützt und diesen nicht durch überfrachtete Möglichkeiten erschlägt.

Python IDLE

Integrated Development and Learning Environment

- Vorteile von IDLE:
 - Einfärben von Codeeingaben im Python-Shell-Fenster
 - direkte Ausgaben und Fehlermeldungen
 - plattformübergreifend verfügbar (Windows, Mac OS X und Unix)
 - Mehrere Fenster gleichzeitig
 - Debugging

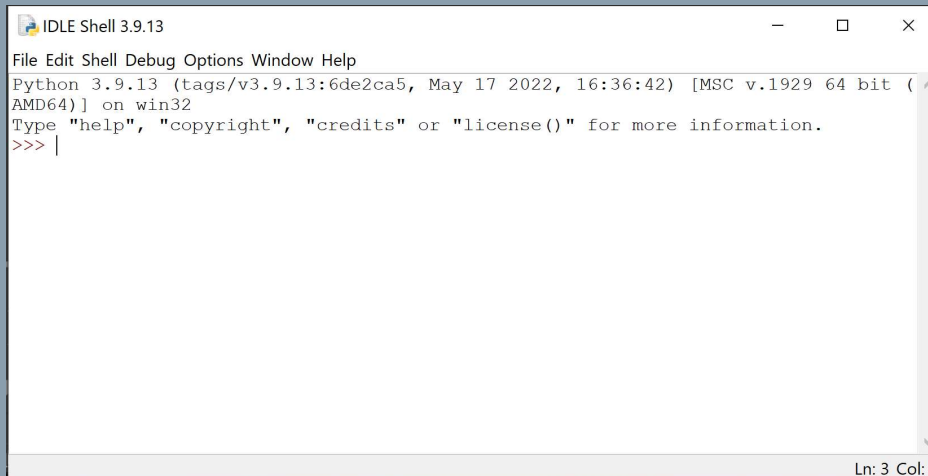
Python IDLE

Starten der IDLE

- Die interaktive Shell wird je nach Betriebssystem gestartet:
- Windows: auch Startmenü klicken und IDLE eingeben – auswählen von IDLE (Python GUI)
- macOS: Finder öffnen und auf Programm klicken. Dort unter Python auf das IDLE-Symbol klicken. Oder über Spotlight-Suche (CMD + Leertaste) und IDLE eintragen und die IDLE-App auswählen
- Linux: Terminal-Fenster öffnen und idle3 eingeben (z.B. auf Kali)

Python IDLE

- Nach dem Start erscheint ein Fenster mit der Angabe der Python-Version
- Jetzt können direkt nach der Eingabeaufforderung (sprich den >>>) Befehle eingegeben werden.



```
IDLE Shell 3.9.13
File Edit Shell Debug Options Window Help
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
Ln: 3 Col: 4
```


Python IDLE

Autovervollständigung über TAB-Taste

- Innerhalb der interaktiven Shell (also IDLE) kann man eine Autovervollständigung nutzen. Nach dem Tippen der ersten 2-3 Buchstaben einfach einmal die TAB-Taste drücken und man bekommt Vorschläge zur Auswahl bzw. wenn es eindeutig ist, den kompletten Python-Befehl ausgeschrieben. Sehr praktisch um Schreibfehler zu vermeiden!

Python IDLE

Weitere Tastenkombinationen innerhalb von IDLE

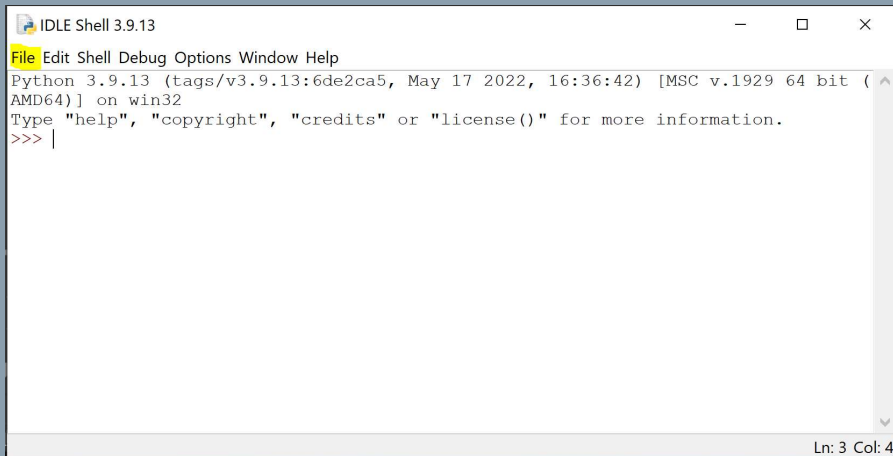
- Um nochmal die letzte Codeanweisung angezeigt zu bekommen einfach die Tastenkombination Alt + p drücken.
- Über Alt + n bekommt man die nächsten Codeanweisung angezeigt, sofern vorhanden.

Python IDLE

Programm mit IDLE Dateieditor schreiben und speichern

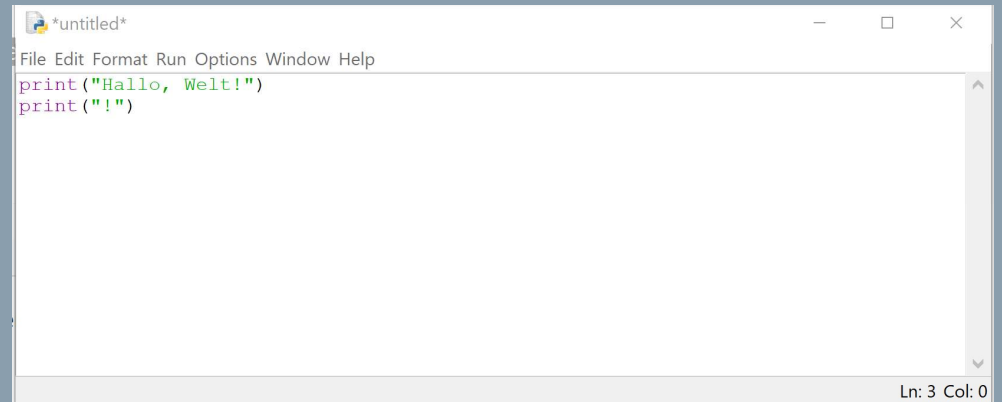
- IDLE verfügt über einen Dateieditor. Diesen Dateieditor kann gestartet werden, indem man im Menü oben auf den Reiter „File“ und dort auf „New File“ klickt.
- Wir erhalten ein weiteres leeres Fenster. In dieses können wir unser Python-Programm schreiben.

Python IDLE



The screenshot shows the Python IDLE Shell window. The title bar reads "IDLE Shell 3.9.13". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The text area contains the following text: "Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32", "Type 'help', 'copyright', 'credits' or 'license()' for more information.", and a prompt ">>> |". The status bar at the bottom right indicates "Ln: 3 Col: 4".

```
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```



The screenshot shows the Python IDLE editor window. The title bar reads "*untitled*". The menu bar includes "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The text area contains the following code: "print('Hallo, Welt!')", "print('!')", and a blank line. The status bar at the bottom right indicates "Ln: 3 Col: 0".

```
print("Hallo, Welt!")
print("!")
```

Python IDLE

- Zum Testen geben wir ein:

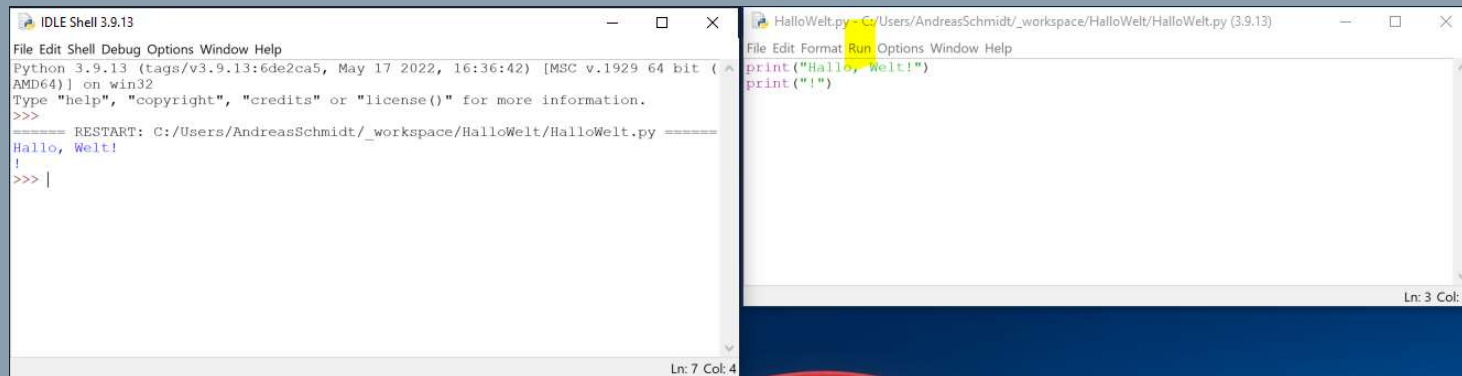
```
print ("Hallo Welt")  
print ("!")
```

- Hier wird bereits der Vorteil sichtbar. Wir erhalten automatisch eine Code-Highlighting (eine Einfärbung des Quellcodes).
- Jetzt speichern mit „File“ -> „Save as..“.
- Wählen unseren Ordner „_workspace/HalloWelt/“
- Und nennen es HalloWelt.py

Python IDLE

Zum direkten Ausführen können wir entweder

- im Menü „Run“ den Unterpunkt „Run Module“ auswählen
- oder einfach F5 drücken.
- Das Programm wird nun direkt in der IDLE Shell ausgeführt:



```
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/AndreasSchmidt/_workspace/HalloWelt/HalloWelt.py =====
Hallo, Welt!
!
>>> |
```

```
File Edit Format Run Options Window Help
print("Hallo, Welt!")
print("!")
Ln: 3 Col: 0
```

Python IDLE

- So ist ein schnelles Lernen möglich.
- Im folgenden Kapitel wird eine weitere Variante beschrieben, wie ein erstelltes Python-Programm direkt über die Befehlszeile gestartet wird bzw. die Python Command Line genutzt werden kann.

Befehle ausführen

- Um mit Python Befehle ausführen zu können, haben wir 2 Möglichkeiten:
 - wir schreiben ein Python-Programm in einem Texteditor und rufen dieses dann auf oder
 - wir tippen unsere Python-Befehle direkt in die „Python Command Line“ und die Befehle werden sofort und direkt ausgeführt.
- Beides hat seine Berechtigung! Schauen wir uns beide Vorgehensweisen an.

Befehle ausführen

Python Command Line

- Um schnell einen Befehl zu testen, können wir Python Command Line nutzen. Diese rufen wir über die Befehlszeile unseres Betriebssystems (egal ob Windows, Mac oder Linux) direkt auf.
- Dazu entsprechend dem Betriebssystem:
 - bei Windows „cmd.exe“ über das Suchfeld eingeben und somit erhalten wir die Befehlszeile
 - beim Mac öffnen wir die Kommandozeile über CMD + Leertaste und dort das Terminal über „terminal.app“
 - Im Linux öffnen wir uns ein Terminal-Fenster z.B. mittels ssh oder der GUI.

Befehle ausführen

- Haben wir die Befehlszeile des Betriebssystems, starten wir Python über **python**
- (unter Umständen ist bei der Installation von 2 verschiedenen Versionen)
 - bei Linux oder beim Mac als Start **python3** notwendig.

Befehle ausführen

- Wir erhalten als Rückmeldung folgende Ausgabe:

```
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

- Nach den 3 „>>>“ geben wir unseren Python-Befehl ein und dieser wird direkt ausgeführt, nachdem wir diesen mit Return bestätigen.

```
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/AndreasSchmidt/_workspace/HalloWelt/HalloWelt.py =====
Hallo, Welt!
!
>>>
```

- Danach können wir den nächsten Befehl eingeben. Werden Variablen definiert, liegen diese zur weiteren Nutzung vor, solange nicht diese Instanz über exit() geschlossen wird.

Befehle ausführen

- Unsere bisherigen Befehle sind so natürlich nicht gespeichert und wenn wir diese wieder ausführen lassen wollten, dann müssten wir alle Befehle in der entsprechenden Reihenfolge wieder tippen.
- Diese Vorgehensweise ist also ungeschickt für sich wiederholende Aufgaben.
- Da lohnt es sich dann immer über ein Python-Programm zu arbeiten, da in einer Datei gespeichert ist, was wir im nächsten Schritt machen.

Befehle ausführen

Python Programm schreiben und speichern

- Wir wollen unser Python-Programm erstens öfters ausführen lassen und zweitens auch ergänzen und erweitern.
- Dann hilft die „Python Command Line“ nicht weiter.
- Wir nutzen einen Texteditor unserer Wahl und speichern unser Python-Programm unter einem Dateinamen mit der Endung „.py“ ab.

Befehle ausführen

- Wir erstellen also eine Textdatei in unserem „_workspace/HalloWelt/“ Ordner mit dem Dateinamen „hallo.py“ und folgendem Inhalt:
 - `print(„Hallo, Welt!“)`
- Um unser Python-Programm nun ausführen zu lassen, rufen wir dieses über die Kommandozeile auf.
- Wie man die Kommandozeile von seinem Betriebssystem erhält, ist auf den vorherigen Folien beschrieben.

Befehle ausführen

- Gehen Sie in der Befehlszeile in das Verzeichnis, in dem unser Python-Programm gespeichert wurde. Dazu wechseln sie die Verzeichnisse mit „cd“, was für „change directory“ steht.
- Unter Windows z.B. „cd c:\users\<username>\Dokumente_workspace\HalloWelt\ “
- Jetzt starten wir Python mit Leerzeichen und dem Dateinamen danach:
 - „python hallowelt.py“
- Alternativ können wir das Programm auch mit seinem vollen Pfad aufrufen:
 - „ python c:\users\<username>\Dokumente_workspace\HalloWelt\hallowelt.py“
- Das Programm wird ausgeführt und wir erhalten die entsprechenden Ausgaben.

- Viel Spaß beim Testen.

Befehle ausführen - Debuggen

Tipp: Python Programm ausführen und danach direkt in Command Line debuggen

- Für dein Einsteiger hört sich folgendes Thema ein wenig abgefahren an. Aber es hilft später immens weiter, wenn man schnell und einfach einmal ein Programm debuggen möchte.
- DEBUGGING oder Entkäfern!
 - Debugging kommt als Begriff aus den Anfängen der IT
 - Computer waren Groß und warm und zogen Käfer an. Diese krochen gerne in diese Schrankgroßen Rechner und verglühten zwischen den Leiterbahnen der Platinen. Hierbei produzierten Sie Kurzschlüsse und somit Programmfehler.
 - Das Entkäfern wurde zu einer täglichen Aufgabe für die Administratoren der „Groß-Rechner“. Dieses Entwanzen prägte den Begriff Debugging!

Befehle ausführen - Debuggen

- Heutzutage sind die Leiterbahnen in Rechnern besser geschützt.
- Auch sind die Spannungen auf den Leiterbahnen geringer.
- Ebenso werden Programme heute anders gespeichert und verarbeitet. Als reine Software!
- Die Fehler oder auch Bugs sind meist logischer Natur und müssen ebenfalls gefunden und beseitigt werden.
- Dies bezeichnet man heute ebenfalls als Debugging.

Befehle ausführen - Debuggen

Wir haben die 2 Möglichkeiten kennen gelernt, wie wir Python-Code ausführen lassen können:

- Python Command Line: Befehle in Eingabe-Konsole eintippen und direkt ausführen
 - Programm mit Editor schreiben und ausführen lassen
-
- Jetzt wäre eine Kombination aus beiden gut – gesagt getan und einfach möglich.

Befehle ausführen - Debuggen

- Das Python-Programm ausführen mit anschließender Python Command Line und Zugriff auf alle Variablen und Möglichkeiten des ausgeführten Programms. Und das Ganze geht mit einem kleinen Kniff.
- Für Anfänger einfach für den Hinterkopf, damit man es später einsetzen kann bzw. nachschlagen.

Befehle ausführen - Debuggen

- Wir haben ein kleines Python-Programm, dass „Hallo Welt“ ausgibt (was auch sonst) und eine Variable speichert.

```
print("Hallo Welt")  
nachricht = "alles in Ordnung"
```

- Dieses Programm speichern wir mit dem Namen „hallowelt.py“ und starten es in der Kommandozeile über „python hallowelt.py“.
- Als Ausgabe sehen wir:

```
Hallo Welt
```

Befehle ausführen - Debuggen

- Wenn wir aber jetzt nachsehen wollen, was in der Variablen mit dem Namen nachricht steckt, haben wir anscheinend Pech.
- Theoretisch können wir unser Programm erweitern mit dem typischen `print()`, speichern, ausführen lassen und später wieder diese Zeile löschen.
- Diese war ja nur zum Debuggen da. ODER ...

Befehle ausführen - Debuggen

- Oder wir kennen den Trick, beides zu bekommen.
- Die Ausführung des Programms und nach Beendigung des Programms die Kommandozeile von Python mit den typischen 3 Größerzeichen:
 - `python -i programmname.py` – Ausführung + Python Kommandozeile
- Der Trick liegt im Aufruf. Wenn wir zwischen python und dem Dateinamen noch ein `-i` einbauen, dann haben wir diesen Effekt.

```
Hallo, Welt!  
!  
>>>
```

Befehle ausführen - Debuggen

- Das Programm wurde nun ausgeführt.
- Nachdem das Programm alles abgearbeitet hat (eine Ausgabe von „Hallo Welt“ ist ja nicht viel), landen wir direkt in der Python Kommando Zeile und können diese wie gewohnt nutzen.

Befehle ausführen - Debuggen

- Jetzt können wir den Inhalt der Variablen mit dem Namen nachricht einfach direkt abfragen.
- Einfach nach dem Prompt ">>>" den gewünschten Befehl tippen bzw. in diesem Fall den gewünschten Variablennamen um den Inhalt der Variablen zu erhalten:
- Hätten wir eine Funktion im Programm integriert, könnten wir auch diese nutzen.

```
===== RESTART: C:/Users/AndreasSchmidt/_workspace/HalloWelt/HalloWelt.py =====  
Hallo, Welt!  
!  
>>> nachricht  
'Alles in Ordnung!'  
>>> |
```


Befehle ausführen - Debuggen

- Wir bauen uns eine Funktion ein, um Zahlen zu quadrieren (sprich eine Zahl wird mit sich selber multipliziert).

```
print("Hallo Welt")  
nachricht = "alles in Ordnung"  
  
def quadrieren(zahl):  
    print(zahl*zahl)
```

- Nachdem wir das Programm neu gestartet haben, können wir diese Funktion nun in unserer Kommandozeile nutzen:

```
Hallo Welt  
>>> quadrieren(4)  
16  
>>> quadrieren(5)  
25
```

Befehle ausführen - Debuggen

Beenden von Python Kommandozeile

- Zum Beenden unsere Python Kommandozeile einfach `exit()` eingeben.

Python Hilfe

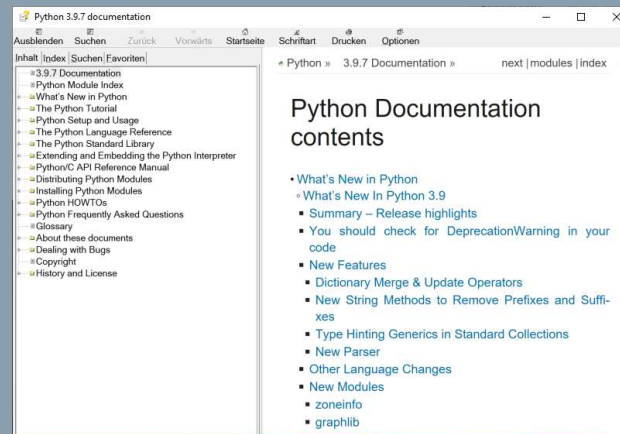
Hilfe-Funktionen in Python

- Es gibt mehrere Möglichkeiten sich in Python Hilfen ausgeben zu lassen.
 - Windows Hilfe Programm „Python Manuals“
 - Online als HTML-Dokumentation (mit Suche)
 - Offline (die gleiche HTML-Dokumentation)
 - über IDLE
 - in der Python-Konsole (Python Shell)

Python Hilfe

Windows Hilfe Programm „Python Manuals“

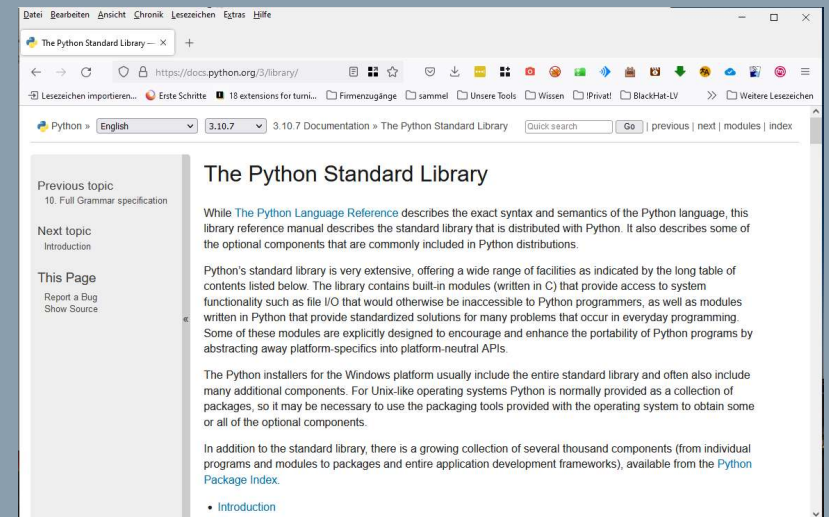
- Bei der Installation unter Windows wird auch das Windows-Hilfeprogramm „Python Manuals“ mit installiert. Dieses kann jederzeit aufgerufen werden und bequem durchsucht werden.



Python Hilfe

Hilfe Online als HTML-Dokumentation (mit Suche)

- Direkt Online kann man sich über die URL <https://docs.python.org/3/library/> die komplette aktuelle Hilfe anzeigen lassen.
- Diese ist mit einer Suchfunktion ausgestattet.
- Es gibt eine Auswahl der verschiedenen Python-Versionen und als Sprache Englisch, Französisch, Japanisch, Koreanisch, Chinesisch.



Python Hilfe

Offline (die gleiche HTML-Dokumentation)

- Die HTML-Dokumentation wird der Installation mitgeliefert. Diese findet sich in einem Unterverzeichnis.
- Beim Mac:
 - `file:///Library/Frameworks/Python.framework/Versions/3.7/Resources/English.lproj/Documentation/index.html`
- Effektiv ist diese identisch mit der Online-Dokumentation – allerdings wird man Online immer den aktuellen Stand bekommen.

Python Hilfe

Hilfe über IDLE aufrufen

- Über die Menüleiste von IDLE kann direkt die Python Dokumentation aufgerufen werden.
- Hier bekommen wir die HTML-Dokumentation angezeigt, die bei der Installation mitgeliefert wurde.
- Gibt es bereits eine neuere Python-Version, ist die Online-Dokumentation empfehlenswert (wie auch die neue Version zu installieren).
- Diese ist mittels „Help“->“Python Docs “ verfügbar
- Ebenfalls kann diese in der Konsole mittels `help()` aufgerufen werden

Python Hilfe

Module von Python

- Zu den Modulen gibt es später im Kurs entsprechenden Tutorials.
- Über die Python-Konsole können wir uns über die bereits standardmäßig in Python verfügbaren Module informieren. Dazu einfach nach dem Prompt `help>` dann `module` eingeben.

Python Hilfe

```
help> modules
Please wait a moment while I gather a list of all available modules...
__future__      _tkinter        glob             sched
__abc__          _tracemalloc    grp             secrets
__ast__          _uuid           gzip            select
__asyncio        _warnings       hallowelt       selectors
__bisect         _weakref        hashlib         setuptools
__blake2         _weakrefset     heapq           shelve
__bootlocale     _xxtestfuzz     hmac            shlex
__bz2            abc             html            shutil
__codecs         aifc            http            signal
__codecs_cn      antigravity     idlelib         site
__codecs_hk      argparse        imaplib         smtpd
__codecs_iso2022 array           imghdr          smtplib
__codecs_jp      ast             imp             sndhdr
__codecs_kr      asynchat        importlib       socket
__codecs_tw      asyncio         inspect         socketserver
__collections    asyncore        io              sqlite3
__collections_abc atexit          ipaddress       sre_compile
__compat_pickle  audioop         itertools       sre_constants
__compression    base64          json            sre_parse
__contextvars    bdb             keyword         ssl
__crypt          binascii        lib2to3         stat
```

Python Lern Editor

- Wer das erste Mal eine Programmiersprache lernt, möchte unter Umständen nicht etwas auf seinem System installieren.
- Für die ersten Schritte gibt es einen Online-Editor, der auch sofort Python-Code ausführt!

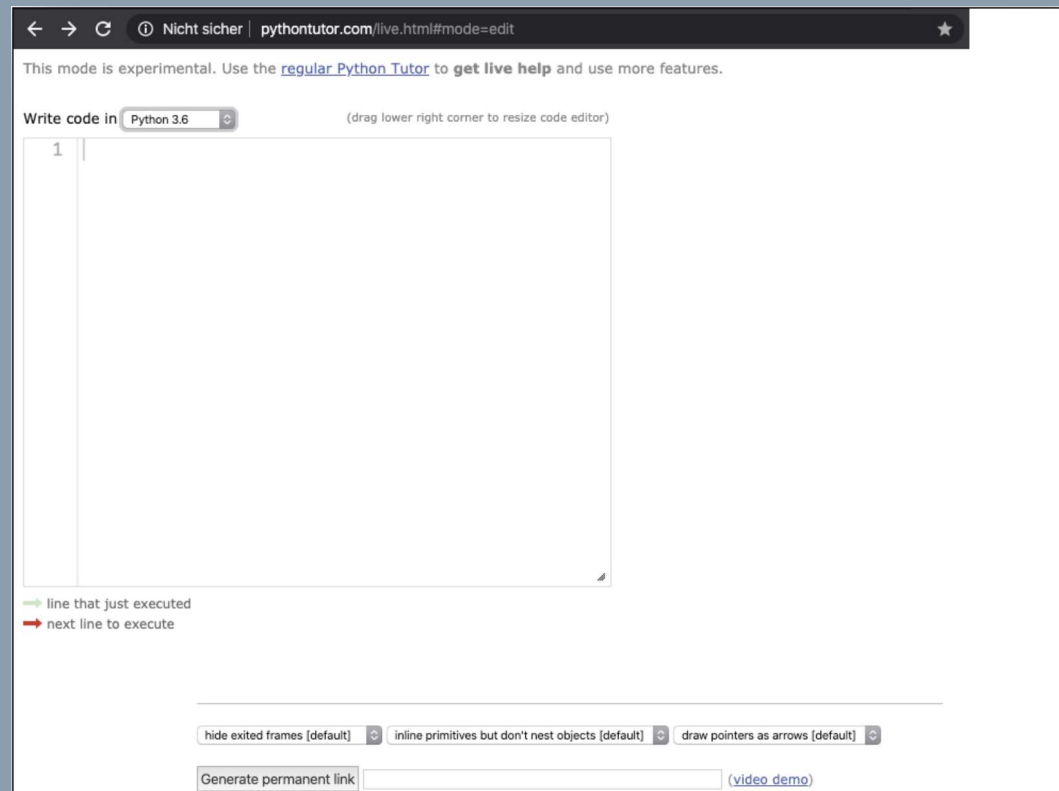
Python Lern Editor

- Das Tolle an diesem Online-Python-Editor ist, dass dieser einen idealen beim Lernen unterstützt.
- Klar ist, dass dieses Kapitel zum Vorstellen des Online-Lern-Editors vor den Kapiteln zum Lernen von Python etwas problematisch ist, da wir ja noch kein Python programmieren können.
- Aber einfach einmal ansehen und im Zweifel später, wenn nicht alles klar geworden ist (was logischerweise normal wäre), nochmals dieses Kapitel lesen.

Python Lern Editor

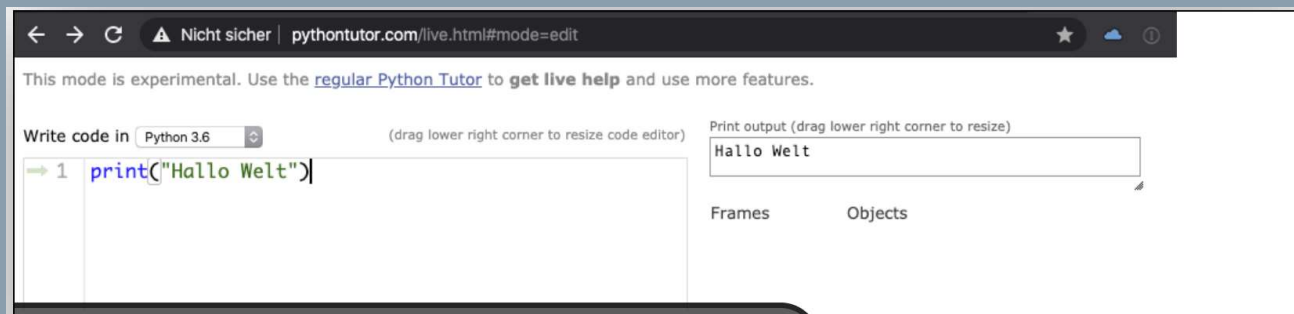
- Der Aufruf des Python-Online-Editors geschieht über folgende URL:
<http://pythontutor.com/live.html>
- Wir erhalten folgendes unscheinbares Fenster.
- Hier nicht täuschen lassen – die Möglichkeiten sind klasse und beschleunigt das Verständnis zum Lernen von Python!

Python Lern Editor



Python Lern Editor

- Jetzt gebe ich den ersten Python Code ein – das typische "Hallo Welt" Ausgabe Programm:



Python Lern Editor

- Rechts sehen wir nun die Ausgabe, die wir auch erhalten würden, wenn wir Python auf dem eigenen Computer installieren. Das ist schon mal ganz
- Jede Programmiersprache hat unterschiedliche Typen von Variablen und Datenstrukturen.
- Im folgenden Code lege ich eine Variable mit dem namen „ccPython“ und eine Datenstruktur mit der Bezeichnung "Liste" und eine als "Tupel" an.
- nett, aber das besondere kommt erst noch!

Python Lern Editor

- Sofort sehe ich rechts im Bereich "Global frame", die vergebenen Namen für unsere verschiedenen Variablen und Datenstrukturen.
- Zusätzlich weiter rechts, in Gelb unterlegt, noch den Inhalt von "list" und "tuple". Hier sieht man auch, dass Python immer bei 0 anfängt zu zählen.

Python Lern Editor

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
([known limitations](#))

```
1 print("Hallo, Welt!")
2 kursname = "ccpython"
3 teilnehmer = ["elke", "Uwe", "Kay"]
→ 4 mnr = (4331, 2543, 5672)
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

[Customize visualization](#)

Done running (4 steps)

Print output (drag lower right corner to resize)

Hallo, Welt!

Frames

Global frame

kursname	"ccpython"
teilnehmer	
mnr	

Objects

list

0	1	2
"elke"	"Uwe"	"Kay"

tuple

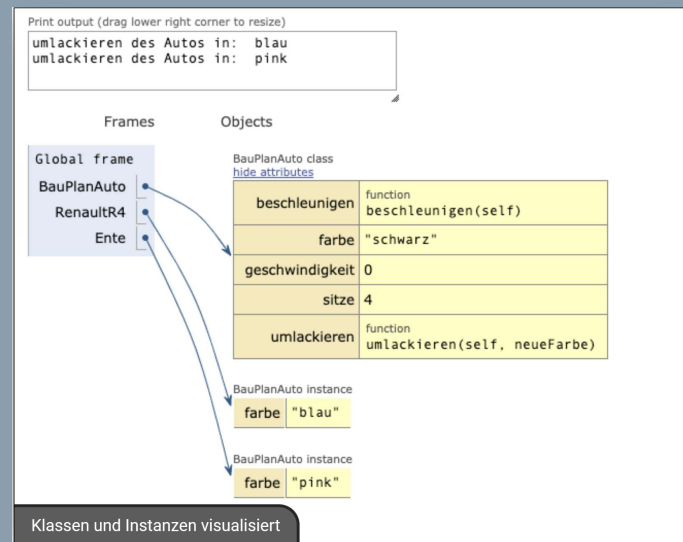
0	1	2
4331	2543	5672

Python Lern Editor

- Und hier hört der Spaß noch lange nicht auf.
- Wenn wir dann in das Kapitel zur objektorientierten Programmierung kommen, hilft auch hier der Online-Editor schnell weiter, um ein Verständnis aufzubauen.
- Ich möchte nur den rechten Ausgabeteil für OOP anzeigen.

Python Lern Editor

- Wir bekommen automatisch Klassen und Instanzen visualisiert.



Python - Editoren

- Computerprogramme bestehen aus Text.
- Wenn du nun ein normales Textverarbeitungsprogramm wie Word verwendest, dann ist es jedoch nicht möglich, das Programm auszuführen.
- Diese Programme speichern viele zusätzliche Informationen zur Gestaltung.
- Sie verwenden ein ganz eigenes Dateiformat, das sich nicht für die Programmierung eignet.
- Daher ist es notwendig, einen Texteditor zu verwenden, der lediglich den reinen Text abspeichert.

Python - Editoren

- Unter Windows können wir dazu z.B. den Notepad ++ verwenden.
- Unter linux können wir auf der Kommandozeile alternativ Nano verwenden.
- Beide Editoren unterstützen Syntax-Highlighting
- Beide Editoren unterstützen das Speichern im reinen Textformat6

Python – Editoren

nano – Kommandozeilen Editor

- Nano ist auf Linuxsystemen meist installiert
 - oder kann mittels des Befehls
- `sudo apt-get install nano`
- Installieren.
- Er unterstützt Syntaxhighlighting
- Er ist remote nutzbar
- Er wird auf der Linux Kommandozeile mittels des Befehls gestartet

```
nano zieldatei.txt
```

Python – Editoren

nano – Kommandozeilen Editor

- Steuerung des Nano-Editors
- Mittels Tastatur
 - ^ steht für die Strg-Taste und wird als Steuerzeichen erkannt.
 - Im unteren Rand der Kommandozeile ist eine Hilfe eingeblendet.
 - ^x = strg + x Exit
 - ^o = strg + o Write Out
 -
- Existiert die editierte Datei nicht, wird sie angelegt, solange sie Inhalt hat.

ATOM-Text Editor

- Es gibt mehrere Gründe, warum man mit einem anderen als dem von Python zur Verfügung gestellten Editor arbeiten möchte. Man kann deutlich mehr Komfort bekommen durch:
 - automatische Vervollständigung von Code (intelligent code completion)
 - Vorschläge von Anweisungen und möglichen Parametern
 - farbliche Hervorhebung, Syntaxhervorhebung (Syntax-Highlighting)
- Allerdings stellt die Wahl des Text-Editors für manche Programmierer fast schon eine Religionsfrage dar. Wenn man noch kein Programm seiner Wahl hat, kann ich den Editor Atom empfehlen.

ATOM-Text Editor

- Der Text-Editor Atom hat viele Komfortmerkmale zum Programmieren in Python von Haus aus und bietet noch deutlich mehr durch seine Erweiterbarkeit.
- Er ist für die Plattformen Windows, Mac OS X, Unix/X verfügbar und ist Open Source (sprich es entstehen keine Kosten).
- Er wurde von den GitHub Leuten entwickelt und steht seit 2014 unter der freien MIT-Lizenz zur Verfügung.
- Der Editor kann über Plug-ins und Themen erweitert werden.

ATOM-Text Editor

Installation Editor Atom

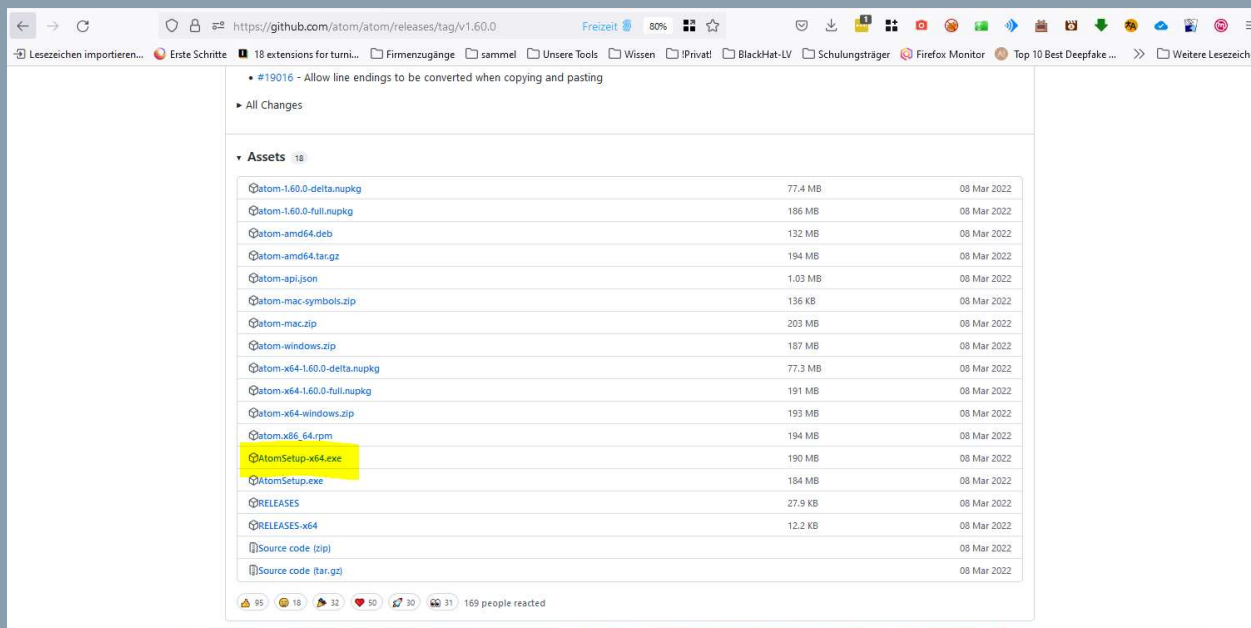
- Erster Schritt ist das Herunterladen der Software für das eigene Betriebssystem unter:
 - <https://github.com/atom/atom/releases/tag/v1.60.0>
 - Nach der Installation (nachdem alles in die Rakete geladen wurde) sieht der Editor eher unscheinbar aus.
- Was ein großer Vorteil ist:
 - wir haben eine aufgeräumte Oberfläche, die nicht vom eigentlichen ablenkt – dem programmieren.

ATOM-Text Editor

- Jetzt können wir uns im frisch gestarteten Editor über das Menü „File“ eine neue Datei „New File“ erstellen.
- Natürlich testen wir mit dem typischen "Hallo Welt"-Programm. Erst durch das Speichern ist dem Editor klar, dass wir in Python programmieren und die farbige Syntaxhervorhebung wird automatisch aktiviert. Will man die Syntaxhervorhebung ohne zu speichern aktivieren, kann man im Fußbereich rechts (links neben dem Logo von GitHub) die gewünschte Programmiersprache einstellen.

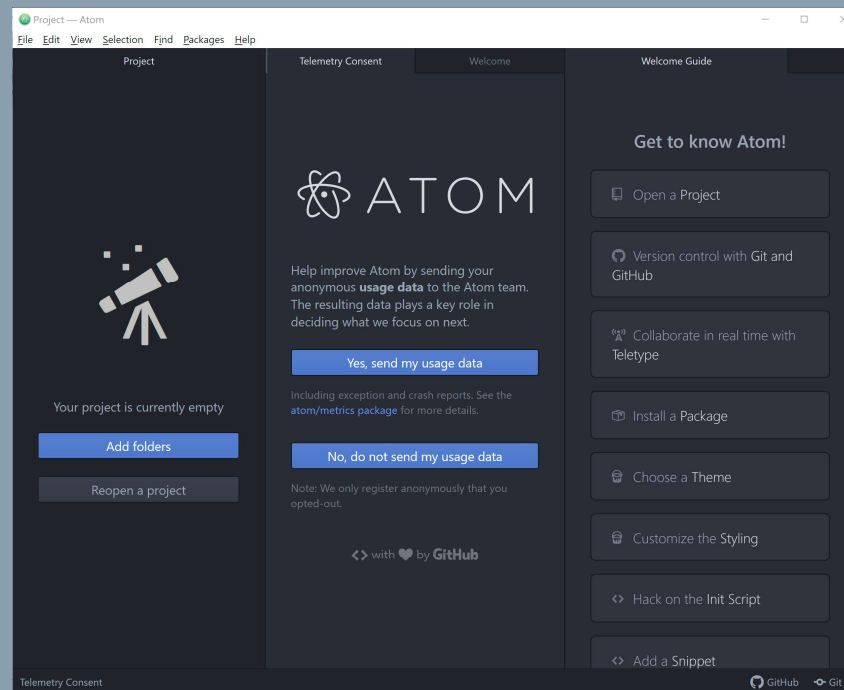
ATOM-Text Editor

- Download von Atom



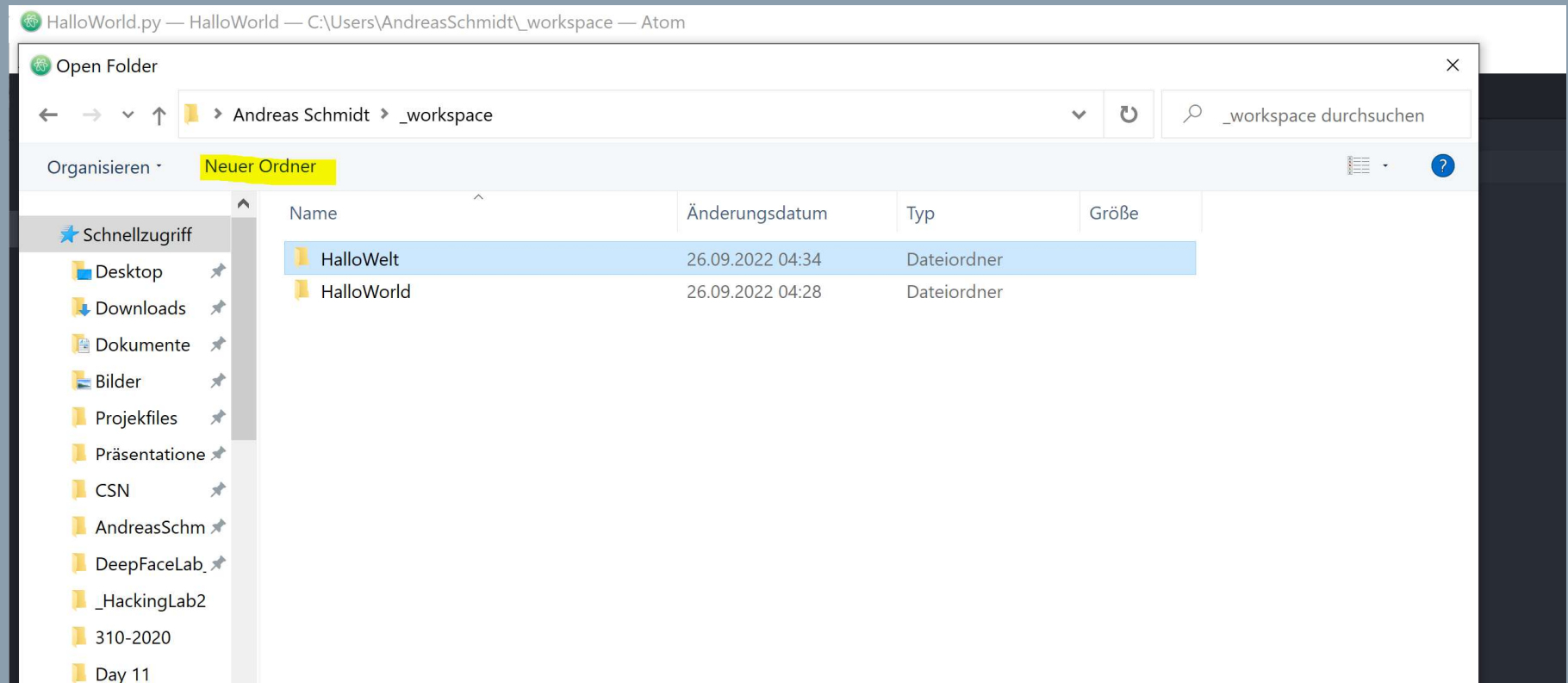
ATOM-Text Editor

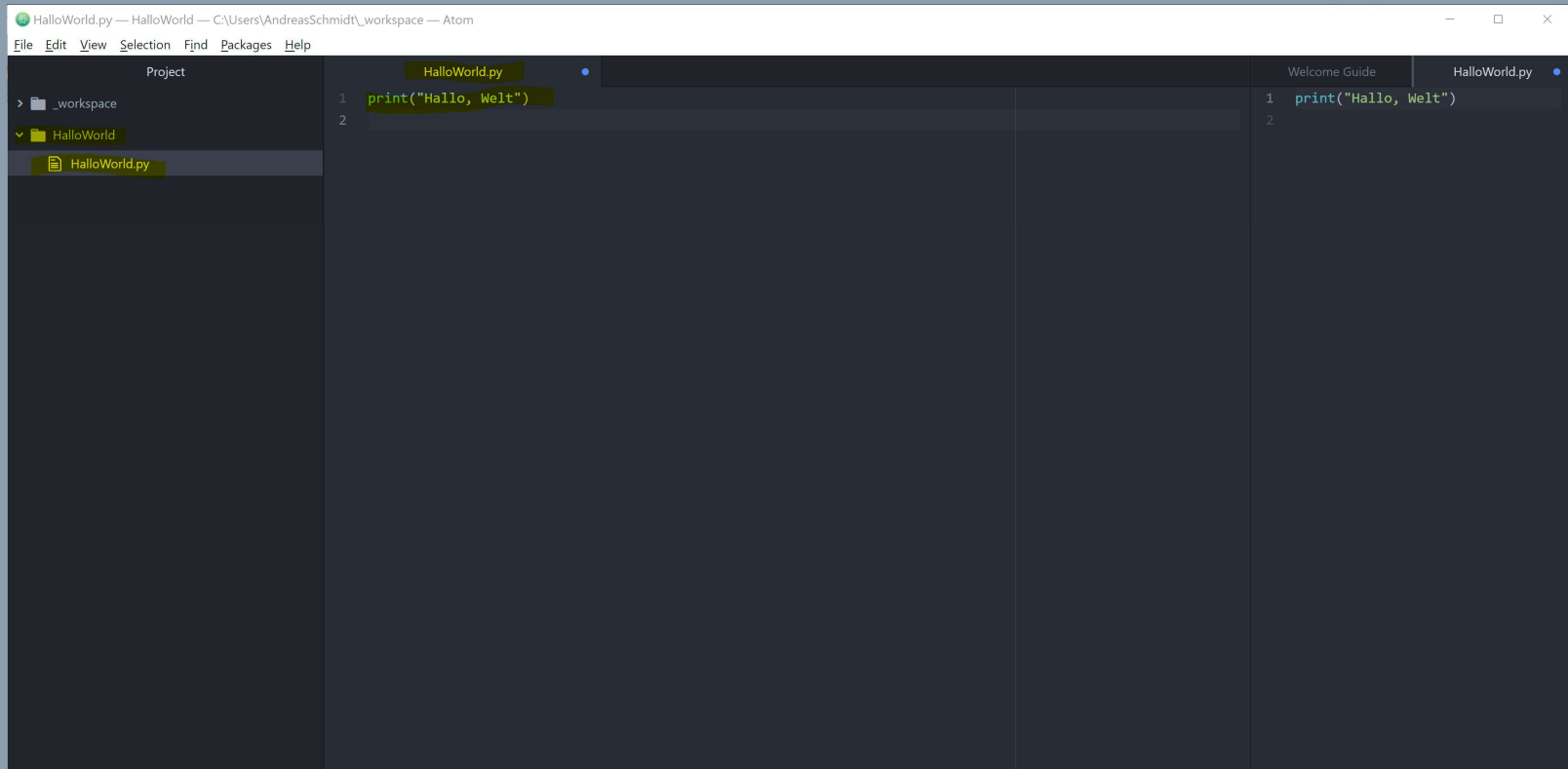
- Nach der Installation



ATOM-Text Editor

- Nun legen wir uns in unseren Dokumenten den Ordner „_workspace“ an.
- Danach öffnen wir diesen Ordner im **Atom**, indem wir über „Ordner Hinzufügen“ den eben erstellten Ordner auswählen.
 - Hiermit haben wir eine Übersicht über alle weiteren Projekte!
- Nun erstellen wir über File->“Add Project Folder,, einen Ordner mit dem Namen „HalloWelt“!
 - Diesen wählen wir danach aus.
 - In diesem Ordner können wir nun eine Datei „HalloWelt.py“ anlegen
 - und im Editorbereich öffnen.





The screenshot shows the Atom text editor interface. The title bar indicates the file is 'HalloWorld.py' located in 'C:\Users\AndreasSchmidt\workspace'. The menu bar includes 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. The left sidebar shows a project view with a folder named '_workspace' containing a subfolder 'HalloWorld' and a file 'HalloWorld.py'. The main editor area displays the code for 'HalloWorld.py' with line numbers 1 and 2. The code on line 1 is `print("Hallo, Welt")`. The right sidebar shows a 'Welcome Guide' and the file 'HalloWorld.py'.

```
1 print("Hallo, Welt")
2
```


ATOM-Text Editor

- nun können wir lokal Programme erstellen und die Quelltexte verwalten.

PyCharm IDE

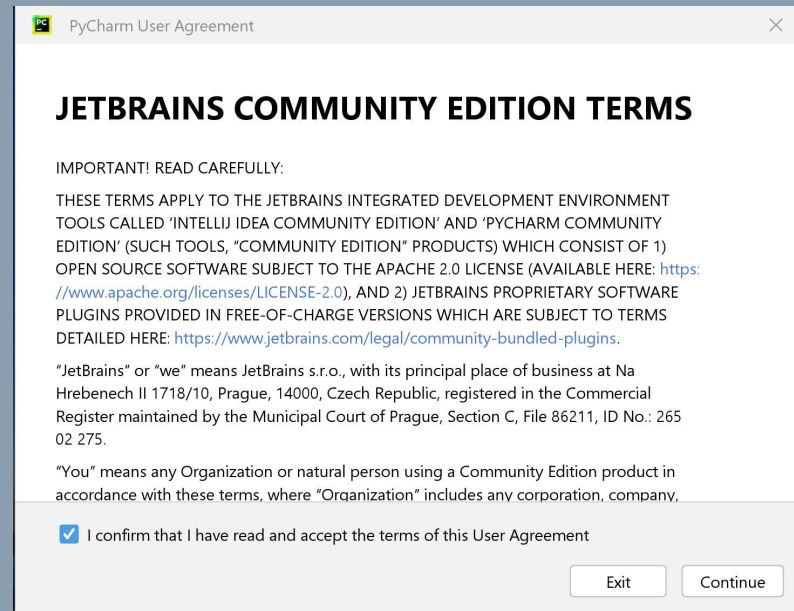
- Die PyCharm-DIE ist ein erweiterter Editor, in dem es auch möglich ist über ssh auf einem Server Quellcode zu erstellen.
- Eine IDE vereint mehrere, zur Entwicklung hilfreiche, Funktionen/Tools, die zusammen als **Integrated Development Environment (IDE)** bezeichnet wird.
- Für Pycharm benötigen wir einen JetBrains account, wegen der nötigen Lizenzverwaltung für die DIE
- Ebenfalls entwickeln wir auf unserem B2-Server und erstellen dort, in unserem Heimverzeichnis, den Ordner „_workspace“.

PyCharm IDE

- Wir gehen auf die Webseite von JetBrains
 - <https://www.jetbrains.com/pycharm/>
- Wir legen uns einen kostenlosen Account an!
 - Notiert euch die Logindaten des Accounts gut.
- Wir laden uns die „Community Edition“ herunter und installieren diese.

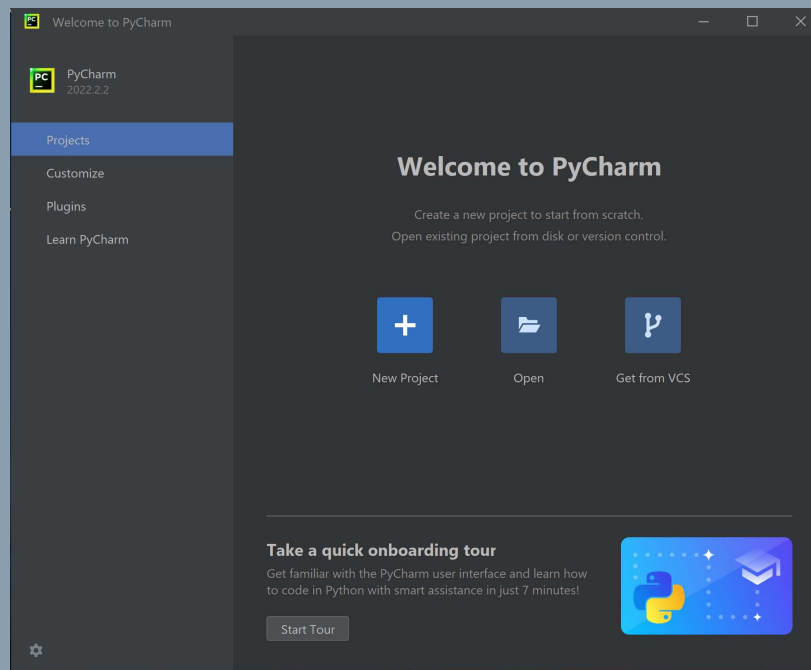
PyCharm IDE

- Beim ersten Start werden wir mit dem Fenster begrüßt und bestätigen die AGB



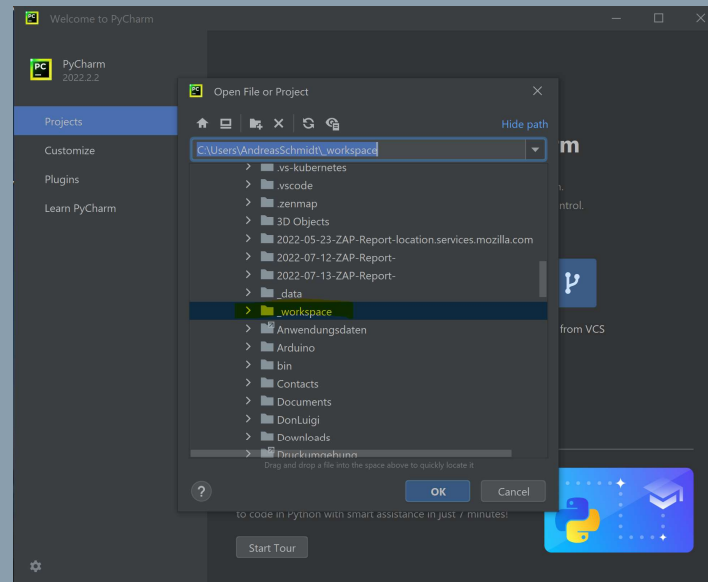
PyCharm IDE

- Hiernach startet PyCharm-Community Edition und begrüßt uns mit diesem Fenster:



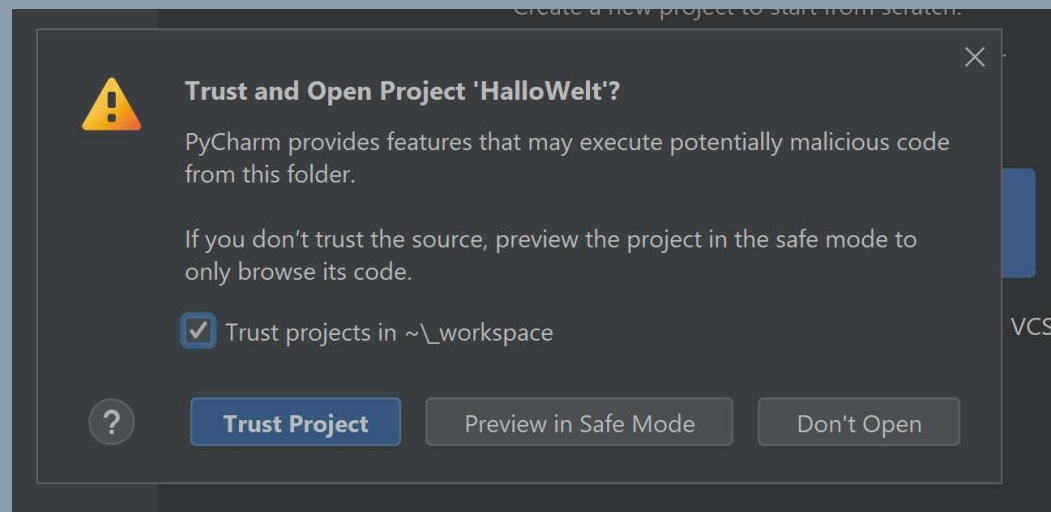
PyCharm IDE

- Nun können wir über die Oberfläche die „open“ Kachel auswählen.
- Hier suchen wir unseren Ordner „_workspace“, darin den Ordner „HalloWelt“ und öffnen ihn.



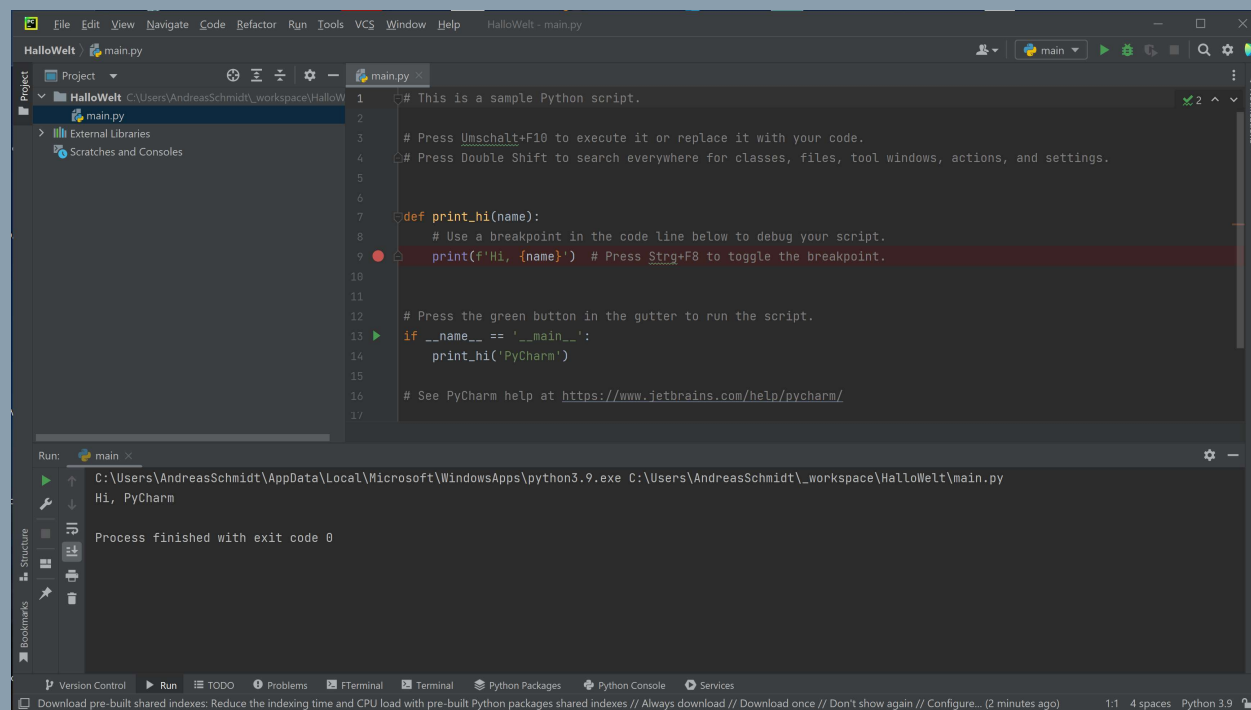
PyCharm IDE

- Nun bestätigen wir, dass wir dem Workspaceordner als Projektquelle vertrauen wollen.
- Dazu setzen wir das Häkchen und klicken auf „Trust Project“



PyCharm IDE

- Nun haben wir PyCharm mit einem geöffneten „Neuen“ Projekt!



PyCharm IDE

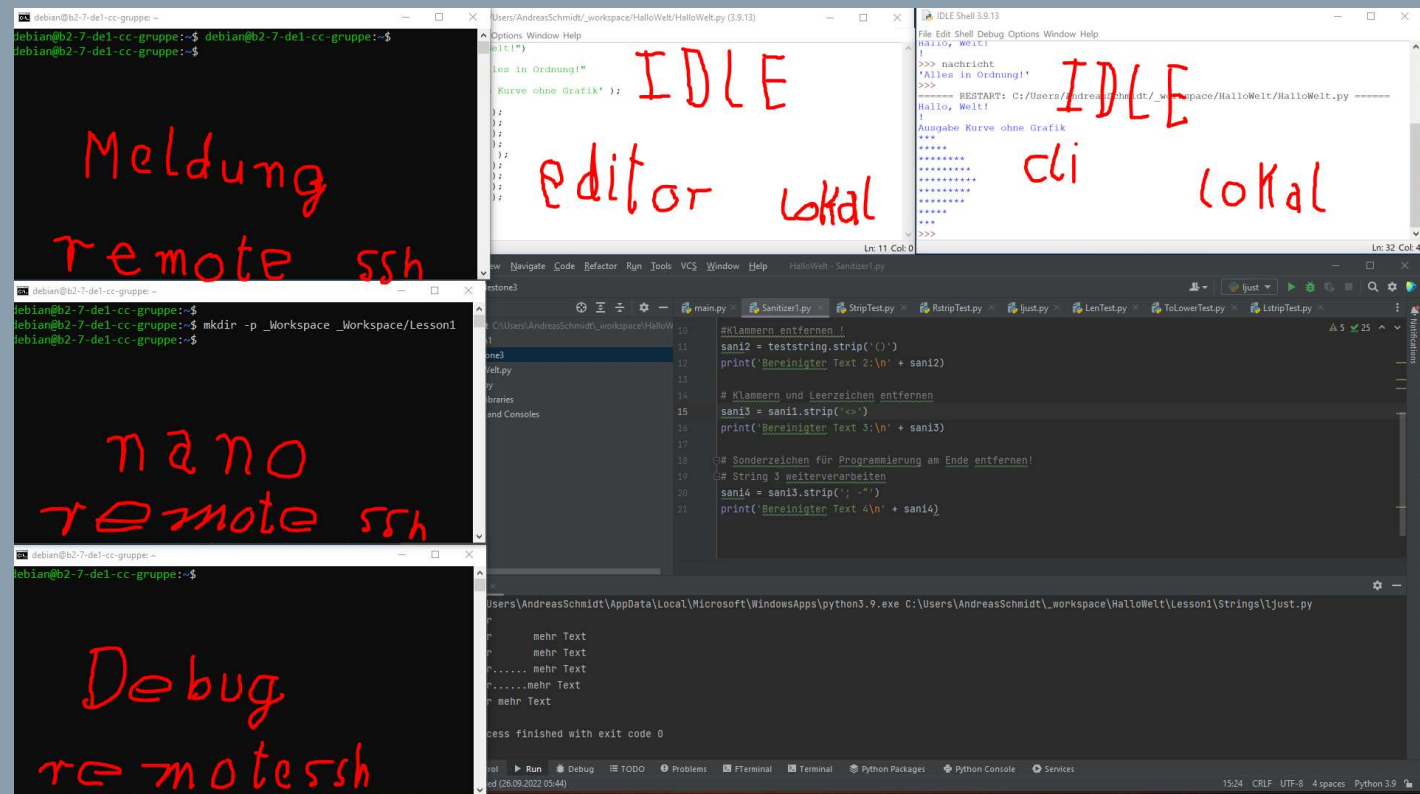
- In diesem Projekt ist schon ein Beispielscript enthalten, das wir mit „umschalt“+“F10“ direkt ausführen können.
- Hierbei öffnet sich am unteren Rand ein Ausgabefenster, das uns die Programmausgabe anzeigt.
- Wir können das Testscript im ganzen oder im Debugmodus ausführen.

Workspace herrichten

- Auf deinen Linuxinstallationen: B2 / Kali
 - `mkdir -p _Workspace _Workspace/Lesson1`
- Auf deinem Windows-Rechner
 - In deinem Dokumenten-Ordner!
 - `md _Workspace`
 - `md _Workspace\Lesson1`

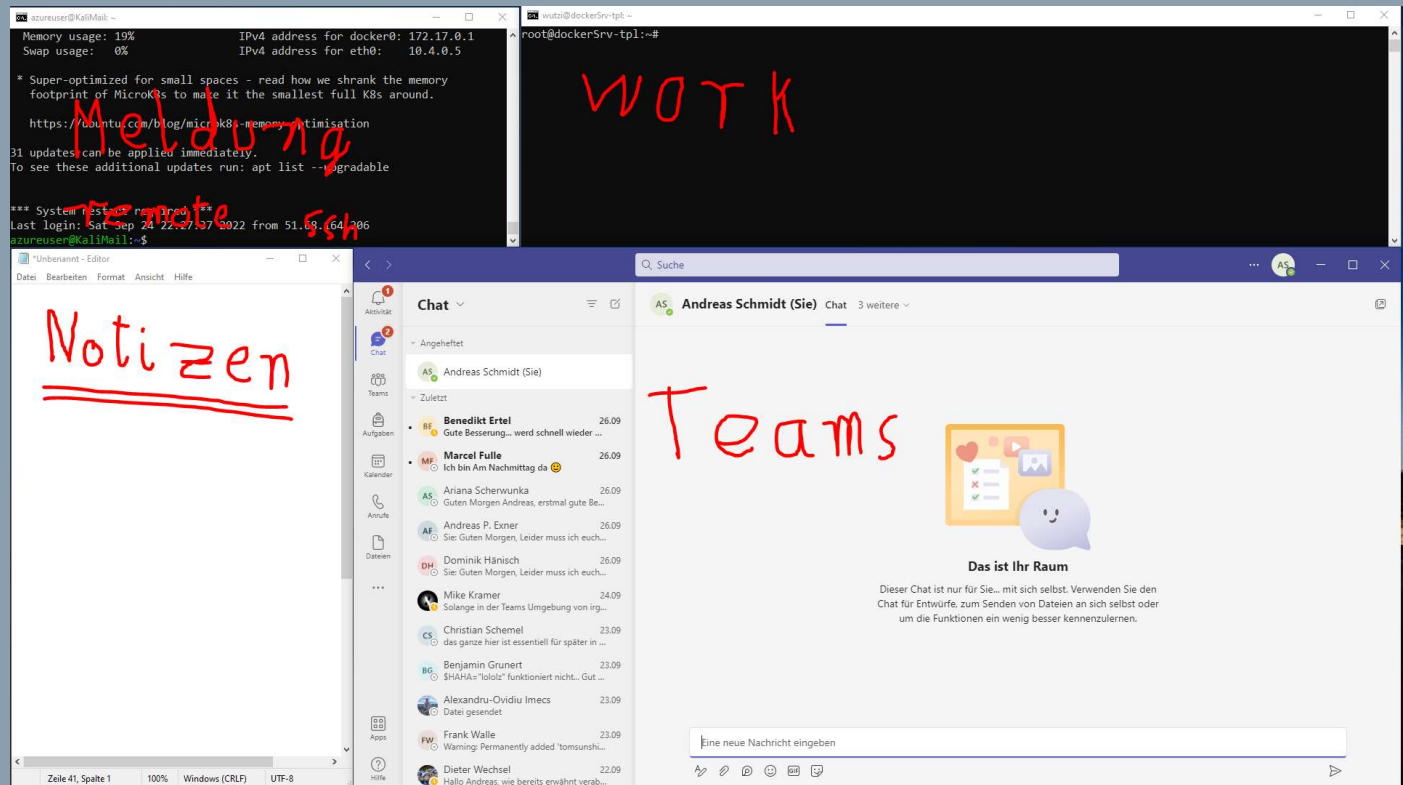
Workspace herrichten

Editor-Bildschirm 1



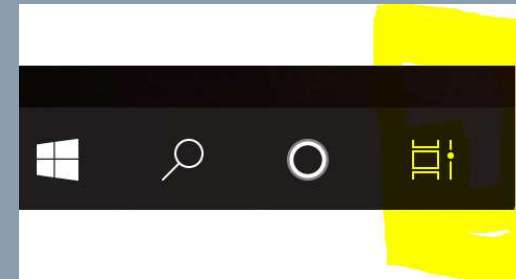
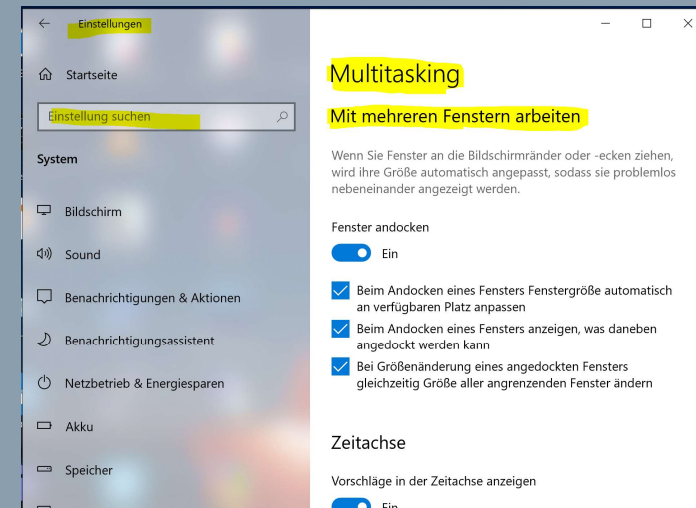
Workspace herrichten

- Teams Bildschirm 2



Multimonitor – auch Virtuel

- Öffne die Einstellungen und suche nach „Multitasking“
- Wähle die Einstellungen für virtuelle Desktops
- Aktiviere sie
- Stelle die Anzeigen (unten auf der Seite) auf „Nur dieser Desktop“
- (siehe Bild 1)
- Klicke nun auf das markierte Feld in der Taskleiste (Bild 2)
- Nun kannst du mehrere Bildschirme verwalten!
- Auf jedem bleiben deine Fenster geöffnet und aktiv. Ausserdem an der selben Stelle!
- Du kannst über das Symbol zwischen den Fenstern wechseln.



Multimonitor – Realer Multimonitor

- Hast du mehrere Monitore, Kannst du beide Nutzen.
- Brauchst du trotzdem mehr, dann kannst du dort auch den Virtuellen Desktop-Modus nutzen.
- So verdoppelst du deine Anzeigefläche.

Multimonitor – auch Virtuel

- Nun Hast du 2-4 Arbeitsflächen zwischen denen Du wechseln kannst.

Python - Grundlagen

- Zum Lernen einer Programmiersprache (und bevor man irgendwelche coolen Spiele programmieren kann), benötigt man ein paar Grundlagen. Dies ist das Grundlagen-Kapitel.
- Was benötigt man so als Grundlagen?
- Grundsätzlich zeichnen sich Programmiersprachen durch das EVA-Prinzip aus. Dabei steht jeder Buchstabe für eine wichtige Möglichkeit und beschreibt die Grundprinzipien der Datenverarbeitung:
 - E: Eingabe
 - V: Verarbeitung
 - A: Ausgabe
- In diesem Grundlagen-Kapitel drehen wir die Reihenfolge um – sprich wir machen „AVE“ und begrüßen Python und tolle Möglichkeiten mit einer schnellen und einfach zu erlernenden Programmiersprache.

Python - Grundlagen

Ausgabe in Python

- Wir schauen uns als Erstes die Ausgabe an – sprich wir lassen etwas über **print** auf dem Bildschirm ausgeben.

Python - Grundlagen

Verarbeitung

- Natürlich möchten wir nicht nur einen bestehenden Inhalt ausgeben, sondern diesen auch verarbeiten können. Dazu helfen mathematische Funktionen – großes Wort, gemeint ist damit einfach Grundrechenarten wie z.B. addieren.
- Zum Verarbeiten müssen wir den Text „zwischenspeichern“ können. Dazu lernen wir Variablen, Listen und Tupel kennen.

Python - Grundlagen

Eingabe

- Und ohne eine Nutzereingabe macht es viel weniger Spaß.
- Daher sollen in unser Programm auch Inhalte von außen kommen können – sprich wir fragen den Benutzer und verarbeiten dann diese Eingaben weiter.
- Es kann sehr einfach aussehen, dass das Eingegebene einfach wieder auf dem Bildschirm ausgegeben wird, oder damit gerechnet wird.

Python - Grundlagen

Kommentieren und Auskommentieren von Code

- Klar fragt man sich was Kommentieren und Auskommentieren im Grundlagen-Kapitel zu suchen haben. In einem Programm werden wir umfangreicheren Quellcode (und komplizierte Bereiche) kommentieren.
- Beides haben wir am Anfang nicht. Aber mit „auskommentieren“ können wir einzelne Teile von unserem Programm abschalten und somit Fehler eingrenzen.
- Daher wird diese Möglichkeit im Grundlagen-Kapitel gezeigt.
- Das sind unsere Grundlagen. In allen Programmiersprachen immer dasselbe – nur der Befehlsaufbau unterscheidet sich.
- Und nun viel Spaß beim Lernen von Python!

Python - Grundlagen

Syntax:

- Die Syntax ist die Grammatik der Sprache.
- Diese schreibt vor wie der Interpreter die Anweisungen erwartet.

Python - Grundlagen

Syntax:

- Python-Bezeichner
- Ein Python-Bezeichner ist ein Name, der zur Identifizierung einer Variablen, einer Funktion, einer Klasse, eines Moduls oder eines anderen Objekts verwendet wird.
 - Ein Bezeichner beginnt mit einem Buchstaben A bis Z oder a bis z oder einem Unterstrich (`_`), gefolgt von null oder mehr Buchstaben, Unterstrichen und Ziffern (0 bis 9).
 - Python erlaubt keine Interpunktionszeichen wie `@`, `$` und `%` innerhalb von Bezeichnern.
 - Python ist eine Programmiersprache, die zwischen Groß- und Kleinschreibung unterscheidet.
 - `Manpower` und `manpower` sind also zwei verschiedene Bezeichner in Python.

Python - Grundlagen

Syntax:

- Python-Bezeichner
- Hier sind die Namenskonventionen für Python-Bezeichner
 - Klassennamen beginnen mit einem Großbuchstaben.
 - Alle anderen Bezeichner beginnen mit einem Kleinbuchstaben.
 - Ein Bezeichner, der mit einem einzelnen führenden Unterstrich beginnt, bedeutet, dass der Bezeichner privat ist.
 - Beginnt ein Bezeichner mit zwei führenden Unterstrichen, bedeutet dies, dass der Bezeichner streng privat ist.
 - Wenn der Bezeichner auch mit zwei nachgestellten Unterstrichen endet, ist der Bezeichner ein sprachdefinierter Spezialname.

Python - Grundlagen

Reservierte Wörter

- In der folgenden Liste sind die Python-Schlüsselwörter aufgeführt.
- Diese Wörter sind reserviert und können nicht als Konstanten-, Variablen- oder andere Bezeichnernamen verwendet werden.
- Alle Python-Schlüsselwörter enthalten nur Kleinbuchstaben.

and	exec	not
as	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Python - Grundlagen

- Zeilen und Einrückung Python bietet keine geschweiften Klammern zur Kennzeichnung von Codeblöcken für Klassen- und Funktionsdefinitionen oder zur Ablaufsteuerung.
- Codeblöcke werden durch Zeileneinrückung gekennzeichnet, die strikt eingehalten wird.
- Die Anzahl der Leerzeichen in der Einrückung ist variabel, aber alle Anweisungen innerhalb des Blocks müssen gleich weit eingerückt werden.

Python - Grundlagen

Zum Beispiel.

```
if True:
    print "Answer"
    print "True"
else:
    print "Answer"
    print "False"
```

Der folgende Block erzeugt jedoch einen Fehler

```
if True:
print "Answer"
print "True"
else:
print "Answer"
print "False"
```

- So würden in Python alle durchgehenden Zeilen, die mit der gleichen Anzahl von Leerzeichen eingerückt sind, einen Block bilden. Das folgende Beispiel enthält verschiedene Anweisungsblöcke -
- Hinweis: Versuchen Sie nicht, die Logik zu diesem Zeitpunkt zu verstehen. Vergewissern Sie sich nur, dass Sie die verschiedenen Blöcke verstanden haben, auch wenn sie ohne geschweifte Klammern stehen.

Python - Grundlagen

Mehrzeilige Anweisungen

- Anweisungen in Python enden normalerweise mit einer neuen Zeile.
- Python erlaubt jedoch die Verwendung des Zeilenfortsetzungszeichens (`\`), um anzuzeigen, dass die Zeile fortgesetzt werden soll.

Python - Grundlagen

- Zum Beispiel.

```
total = item_one + \  
        item_two + \  
        item_three
```

Python - Grundlagen

Mehrzeilige Anweisungen

- Anweisungen, die innerhalb der Klammern [], {} oder () stehen, müssen nicht das Zeichen für die Zeilenfortsetzung verwenden.

```
days = ['Monday', 'Tuesday', 'Wednesday',  
        'Thursday', 'Friday']
```

Python - Grundlagen

Anführungszeichen in Python

- Python akzeptiert einfache ('), doppelte (") und dreifache (''' oder """) Anführungszeichen, um Zeichenkettenlitterale zu bezeichnen, solange die gleiche Art von Anführungszeichen am Anfang und Ende der Zeichenkette steht.
- Die dreifachen Anführungszeichen werden verwendet, um die Zeichenkette über mehrere Zeilen zu spannen. Zum Beispiel sind alle folgenden Zeichen zulässig -

```
word = 'word'  
sentence = "This is a sentence."  
paragraph = """This is a paragraph. It is  
made up of multiple lines and sentences."""
```

Python - Grundlagen

Kommentare in Python

- Ein Rautezeichen (#), das nicht innerhalb eines Zeichenketten-Literales steht, leitet einen Kommentar ein.
- Alle Zeichen nach dem # und bis zum Ende der physikalischen Zeile sind Teil des Kommentars und werden vom Python-Interpreter ignoriert.

Python - Grundlagen

Beispiele für Kommentare:

```
#!/usr/bin/python  
  
# First comment  
print "Hello, Python!" # second comment
```

Erzeugt folgende Ausgabe:

```
Hello, Python!
```


Python - Grundlagen

Beispiele für Kommentare

Du kannst Kommentare in der gleichen Zeile hinter einer Anweisung schreiben

```
name = „madisetti“ # Dies ist ein Kommentar
```

Du kannst Kommentare auch über mehrere Zeilen schreiben

```
# Dies ist ein Kommentar  
# Dies ist auch ein Kommentar  
# Dies ist ebenfalls ein Kommentar  
# Ich sagte es bereits
```

Python - Grundlagen

Beispiele für Kommentare

- Die folgende Zeichenkette in dreifachen Anführungszeichen wird vom Python-Interpreter ebenfalls ignoriert und kann als mehrzeiliger Kommentar verwendet werden:

“““

Dies ist ein mehrzeiliger
Kommentar.

“““

Python - Grundlagen

Mehrere Anweisungen in einer einzigen Zeile

- Das Semikolon (;) ermöglicht mehrere Anweisungen in einer einzigen Zeile, sofern keine der Anweisungen einen neuen Codeblock einleitet.

Hier ist ein Beispiel für die Verwendung des Semikolons:

```
Import sys; x = 'foo'; sys.stdout.write(x + '\n'); # Mehrere Anweisungen in einer Zeile
```

Python - Grundlagen

Mehrere Anweisungsgruppen als Suites

- Eine Gruppe von Einzelanweisungen, die einen einzigen Codeblock bilden, werden in Python **Suites** genannt.
- Zusammengesetzte oder komplexe Anweisungen wie **if**, **while**, **def** und **class** erfordern eine **Kopfzeile** und eine **Suite**.
- **Kopfzeilen** beginnen die **Anweisung** (mit dem **Schlüsselwort**) und enden mit einem **Doppelpunkt** (:), gefolgt von einer oder mehreren Zeilen, aus denen die **Suite** besteht.

Zum Beispiel:

```
if expression :  
    suite  
elif expression :  
    suite  
else :  
    suite
```

Python - Grundlagen

Befehlszeilenargumente

- Viele Programme können ausgeführt werden, um Ihnen einige grundlegende Informationen darüber zu geben, wie sie ausgeführt werden sollen.
- In Python können Sie dies mit **-h** tun

```
$ python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d      : debug output from parser (also PYTHONDEBUG=x)
-E      : ignore environment variables (such as PYTHONPATH)
-h      : print this help message and exit

[ etc. ]
```

Python - Grundlagen

Sie können Ihr Skript auch so programmieren, dass es verschiedene Optionen akzeptieren soll. Befehlszeilenargumente sind ein fortgeschrittenes Thema und sollten etwas später behandelt werden, wenn Sie die restlichen Python-Konzepte durchgenommen haben.

Variablen und Strings

Ausgaben über print()

- Um eine Ausgabe in Python zu erhalten, gibt es die Funktion **print()**.
- Innerhalb der Klammer kann eine **Zeichenkette (String)** eingetragen werden, die dann auf dem Bildschirm ausgegeben wird.
- Dazu wird diese **Zeichenkette** in **Anführungszeichen** gesetzt.
- Dabei ist es egal, ob **doppelte oder einfache Anführungszeichen!**

```
print('Hallo, Welt');
```

- Die in der Klammer angegebenen Anführungszeichen dienen zum Umschließen der Ausgabe und werden nicht mit ausgegeben.

Ausgabebefehl - Print

- Mit der Funktion `print()` können wir auch den Inhalt von Variablen (mehr zu Variablen im folgenden Kapitel) ausgeben lassen.
- Dazu wird einfach der Variablennamen in die Klammern geschrieben (auf jeden Fall ohne Anführungszeichen, sonst wird der Variablenname einfach als Zeichenkette ausgegeben und nicht der Inhalt des Strings).

```
print(vorname)
```

- Ist der Variable noch kein Inhalt zugewiesen, wird eine Fehlermeldung (Traceback) ausgegeben:
NameError: name 'vorname' is not defined

Ausgabebefehl - Print

- Daher erst einmal den Inhalt der Variable vor der Ausgabe über print() festlegen:

```
Kursname = ' Python Kurs'  
Print(kursname)
```

- Nun, da die Variable existiert kann sie auch ausgegeben werden.
- Da in der Variablen der Wert „Python Kurs“ gespeichert ist, wird folgendes ausgegeben:

```
Python Kurs
```

Ausgabebefehl - Print

Leerzeilen ausgeben über print()

- Wir können auch eine Leerzeile ausgeben, in dem wir ein leeres print() nutzen:

```
Print(„Hallo Welt“)  
Print()  
Print(„Leerzeile davor“)
```

- Es gibt noch weitere Möglichkeiten, eine Leerzeile auszugeben.
- Dazu benötigen wir aber erst unseren Sonderfall im sofort folgenden Abschnitt!

Ausgabebefehl - Print

Sonderfall Backslash

- Enthält eine Variable ein Backslash, dann ist spannend, was die Ausgabe daraus macht.
- Natürlich fragt man sich, wann man überhaupt einen Backslash in einer Variablen haben sollte.
- Nehmen wir an, wir wollen einen Pfadnamen von der Festplatte in einer Variablen speichern.
- Dann wäre die Schreibweise

```
pfad = 'c:\niedlichverzeichnisname'  
pfad = "c:\niedlichverzeichnisname"
```

Lassen wir uns nun unsere Variable **pfad** über **print()** ausgeben

```
print(pfad)
```

Ausgabebefehl - Print

- Nun erhalten wir eine Ausgabe über 2 Zeilen umgebrochen ohne den Backslash und vor wurde allem unser "niedlich" zu "iedlich":

```
C:  
iedlichverzeichnisname
```

Was ist passiert?

- Den Backslash haben wir bei den Variablen bereits kennengelernt.
- Hier war er dafür da, bestimmte Zeichen zu maskieren, damit diese überhaupt ausgegeben werden können.
- Unser Backslash hat also mehr als eine Bedeutung.

Ausgabebefehl - Print

- Wollen wir einen Backslash ausgeben, müssen wir den Backslash mit einem Backslash maskieren:

```
pfad = "C:\\niedlicherverzeichnisname"
```

- Aber was ist nun wirklich im Hintergrund passiert?
- Es gibt sogenannte **Steuersequenzen**, die die Ausgabe beeinflussen.
- Und genau so eine Steuersequenz haben wir mit `\n` versehentlich erwischt.
- Dabei steht `\n` für den Zeilenumbruch.
- Wollen wir also eine Leerzeile gezielt ausgeben lassen, dann einfach 2-mal den Zeilenumbruch `\n\n` in der Anweisung **print()** nutzen.

Ausgabebefehl - Print

Steuersequenzen für Tabulator

- Andere Steuersequenzen sind z.B. `\t` für den **Tabulator**.
- Einfach einmal in `print()` testen!

Ausgabebefehl - Print

Ausgabe mit print(r)

- Möchte man der print()-Funktion abgewöhnen, dass diese Inhalte interpretiert, kann man bei dieser eine "rohe" Ausgabe (raw) über das Kürzel "r" am Anfang erzwingen.
- Dann erhält man auch die unverfälschte Ausgabe:

```
print(r"C:\niedlicherverzeichnisname")
```

- So richtig praktisch ist das bei Variablen allerdings nicht:

```
print(rpfad)
```

- Nun haben wir da keine eindeutige Angabe und bekommen eine Fehlermeldung!

Ausgabebefehl - Print

Mehrere Umbrüche bei der Ausgabe

- Das Umbrechen in der Ausgabe mit dem Steuerzeichen ist eine Möglichkeit, mehrere Zeilen bei der Ausgabe zu erzeugen.
- Das Ganze klappte auch durch die Angabe von 3 Anführungszeichen am Anfang und dann natürlich auch am Ende.
- Hier können sowohl die doppelten wie die einfachen Anführungszeichen verwendet werden.
- Es sollten nur dieselben verwendet werden:

```
print("""Hallo  
Welt  
– in 3 Zeilen""")
```


Variablen und Strings

Variablen in Python einsetzen

- Variablen dienen zum Speichern von Inhalten.
- Mit diesen Variablen kann man dann weitere Aktionen machen.
- Aber was ist eine Variable?
- Variablen sind wie ein Schrank mit vielen kleinen Schubladen.
- In jede Schublade kann nur 1 Inhalt gesteckt werden.
- Wird ein neuer Inhalt in eine Schublade gesteckt, fällt der alte Inhalt raus.

Variablen und Strings

Unser virtueller Schrank hat beliebig viele Schubladen.

- Jetzt müssen wir natürlich irgendwie in Python sagen, welche Schublade wir nutzen möchten.
- Um eine bestimmte Schublade auszuwählen, ist die Variante "dritte Schublade von rechts und zweite von oben" ein wenig unhandlich.
- Daher benötigen wir ein System, das handlicher ist und verständlich.
- Daher können wir unseren Schubladen Namen geben.
- So können wir Beispielsweise eine **Schublade beschriften** mit "**vorname**".
- Somit wissen wir, dass wir in dieser **Schublade** einen **Vornamen** finden.
- Bei der Vergabe von **Namen** für unsere **Variablen** gibt es folgende Kriterien:
 - **keine Leerzeichen**

Variablen und Strings

- Um nun die Schublade zu nutzen, also etwas in der Schublade (Variablen) abzulegen wird dem Variablennamen über ein Gleichheitszeichen ein Inhalt zugewiesen.

```
vorname = 'Axel'
```

- Wichtig ist, dass der Inhalt (hier der Name 'Axel') mit einfachen Anführungszeichen umschlossen wird (das neben der Enter-Taste – NICHT "accent circonflexe/grave"!).

Variablen und Strings

- Es können auch doppelte Anführungszeichen verwendet werden – wo der Unterschied liegt, wird später erklärt.

```
vorname = "Axel"
```

- Dieses Umschließen ist aus zwei Gründen wichtig, da der Variableninhalt auch aus mehreren Worten bestehen kann.
- Somit umschließen unsere einfachen Anführungszeichen unseren Inhalt.

Variablen und Strings

- Würde wir ganz auf Anführungszeichen verzichten, würden wir einer Variablen den Inhalt der zweiten Variablen zuweisen.
- Technisch funktioniert dies problemlos – nur wenn es nicht gewünscht ist, dann kommen irritierende Ergebnisse!
- Der Vollständigkeit halber:

```
vorname = Axel
```

- Das klappt nicht!
- Wir bekommen eine Fehlermeldung **"NameError: name 'Axel' is not defined"**.
- Warum?
- Hier versuchen wir der Variablen vorname den Inhalt der (jetzt kommt es) Variable mit dem Namen "Axel" zuzuweisen.
- Daher kommt es zu dieser Fehlermeldung.

Variablen und Strings

- Funktionieren würde (Über den Sinn kann man sich Gedanken machen):

```
Axel = "Test"  
vorname = Axel
```

- Wir können auch leere Inhalte zuweisen:

```
vorname = "
```

- Das Beispiel sieht natürlich verwirrend aus, da 2 hintereinander- kommende einfach Anführungszeichen aussehen wie 1 doppeltes Anführungszeichen.

Variablen und Strings

- Die Schreibweise mit doppelten Anführungszeichen sieht auch lustig aus.

```
Vorname = ""
```

Variablen und Strings

- Was passiert aber, wenn wir das einfache Anführungszeichen als Inhalt benötigen?
- Also Wörter mit Apostroph – im Deutschen wird das Hochkomma auch Apostroph genannt und wird zur kennzeichnet beispielsweise den Genitiv von Eigennamen oder Auslassungen (sorry für den Ausflug in die Grammatik der deutschen Sprache).
- Als Beispiel dafür "Ku'damm für Kurfürstendamm".

Variablen und Strings

- Wollten wir nun aber den "Ku'damm" in einer Variablen nutzen:

```
strasse = 'Ku'damm'
```

- Dann ergibt diese Nutzung Probleme!
- Warum?
- Python schaut nach dem Anfangs-Anführungszeichen und startet dann, alles danach bis zum nächsten einfachen Anführungszeichen dies der Variablen zuzuweisen.
- In unserem Fall wäre das dann "Ku".
- Mit dem Rest ist nun Python komplett verwirrt und weiß nicht mehr was tun und wirft mit einer Fehlermeldung nach uns:
"SyntaxError: invalid syntax"

Variablen und Strings

- Um nun keine Katastrophe mit dem Apostroph zu erleben, müssen wir diesen "tarnen".
- Hier spricht man bei Programmiersprachen von "maskieren", oder „Escaping“.
- Vor diesem Zeichen wird ein Backslash gepackt und die Welt ist für Python wieder in Ordnung

```
strasse = 'Ku\'damm'
```

- Im Zusammenhang mit Variablen fallen Begriffe wie "string".
 - Dahinter verbirgt sich, dass eine Zeichenkette (sprich "string") genutzt wird.
 - Unterschieden werden u.A. z.B. Ganzzahlen, die dann als "int" (ausgeschrieben Integer bzw. Ganzzahl") genutzt werden.

Variablen und Strings

Konvention der Schreibweise

- Kleinschreibung der Variablennamen
- Variablennamen werden kleingeschrieben.
- Das ist ein Abkommen, sprich Konvention, die Sinn ergibt, eingehalten zu werden.
- Dadurch weiß ein anderer Programmierer beim Lesen des Codes „Hey, hier handelt es sich um eine Variable“.

Variablen und Strings

Unterstriche bei mehreren Wörtern

- Besteht der Variablennamen aus mehreren Wörtern, wird als Trennung der Unterstrich verwendet:
- **Beispiel Variablennamen mit mehreren Worten:**

```
meine_variable = "Inhalt"
```

Variablen und Strings

Sprechende Variablennamen

- Ein Variablenname sollte sprechend, also erkennbar sein, was in der Variablen gespeichert ist.
- Früher hat man lustig Variablennamen wie „x1“ oder „i3“ vergeben.
- Spätestens nach 2 Wochen wusste selbst der Programmierer des Codes nicht mehr so genau, was hinter der Variablen steckt (sobald das Programm ein wenig komplexer war als ein 3-Zeiler).
- Daher ist es sinnvoll, kurze aussagekräftige Variablennamen zu vergeben.

Variablen und Strings

Variableninhalt über mehrere Zeilen

- Wenn der Inhalt unserer Variablen über mehrere Zeilen gehen soll, weil wir z.B. Macbeths Hexenszene in einer Variablen speichern wollen, dann klappt folgende Variante NICHT:

```
# FUNKTIONIERT NICHT!  
macbethtext = "Schön ist häßlich, häßlich schön.  
Wir weichen wie Wolken und Windeswehn."
```

- Hier kommen nun 3 Anführungszeichen hintereinander zum Einsatz. Dadurch können wir über beliebig viele Zeilen den Inhalt der Variable schreiben:

```
macbethtext = """"Schön ist häßlich, häßlich schön.  
Wir weichen wie Wolken und Windeswehn."""""
```

- Wobei sowohl doppelte als einfache Anführungszeichen funktionieren (nur müssen die gewählten Anführungszeichen sowohl am Anfang wie am Ende genutzt werden). Das letzte Beispiel mit einfachen Anführungszeichen:

```
macbethtext = "'Schön ist häßlich, häßlich schön.  
Wir weichen wie Wolken und Windeswehn.'"
```

Variablen und Strings

Funktionen und Methoden bei Variablen/Strings

- Wir haben im letzten Kapitel Variablen kennen gelernt. In einer Variablen kann also ein Wert abgelegt und wieder ausgegeben werden.
- Über die Funktion **print()** erhalten wir den abgelegten Inhalt einer Variablen angezeigt.

```
kursname = Python Grundkurs'  
print(kursname)
```

Variablen und Strings

Funktionen bei Variablen

- Einer Funktion wird also nach dem Funktionsnamen in den folgenden Klammern Werte (und Anweisungen) mit übergeben.
- Unserer Funktion `print(kursname)` bekommt also in der Klammer die Variable mit übergeben und weiß somit, was auf dem Bildschirm ausgegeben werden soll.

Variablen und Strings

- Es muss allerdings nicht immer einer Funktion weitere Werte und Anweisungen in der Klammer mit übergeben werden.
- Wird `print()` einfach mit einer leeren Klammer aufgerufen, dann erhalten wir als Ergebnis eine Leerzeile ausgegeben.
- Schauen wir uns eine andere Funktion neben `print()` an.

Variablen und Strings

len()

- Zum Bestimmen der Länge eines Strings gibt es die Funktion len(x).
- Als Information erhalten wir hier die Anzahl der enthaltenen Zeichen.
- Bei len handelt es sich um die Abkürzung des englischen Worts „length“.
- Allerdings müssen wir irgendwas mit dieser Information machen.
- Lassen wir uns diese ausgeben:

```
kursname = 'Python lernen'  
print(len(kursname))
```

- Funktionen können also innerhalb anderer Funktionen genutzt werden.
- Funktionen sind unabhängig.
- Wir können die Funktionen nicht nur für Strings nutzen!
- Funktionen sind soweit einfach verständlich.
- Was aber sind Methoden und was können wir damit bei Variablen anstellen?

Variablen und Strings

Methoden bei Variablen/Strings

- Ändern wir unsere Sichtweise.
- Wenn wir unsere Variable als ein Objekt ansehen, dann kann dieses Objekt verschiedene Möglichkeiten bieten.
- Ich schreibe hier bewusst Möglichkeiten.
- Warum?
- Wenn wir es aus objektorientierter Sicht ansehen, stecken wir schon voll in der objektorientierten Programmierung (OOP).
- Diese wird später im Detail besprochen.

Variablen und Strings

- Grundlegend ist, dass bei der OOP die Punktschreibweise genutzt wird.
- Wir haben unser Objekt, was an erster Stelle steht und wollen hier für dieses Objekt etwas abfragen bzw. eine festgelegte Aktion ausführen:
 - Die Aktion wird über einen Punkt direkt dahinter gepackt.

```
objektname.aktion
```

Variablen und Strings

- Die wirkliche Kunst liegt darin zu wissen, welche Objekte welche Möglichkeiten (sprich Methoden) bieten.
- Diese Hilfe können wir über die Anweisung `help(str)` anfordern für unsere Strings (was unsere Variablen sind).
- Hier kommt dann ein sehr umfangreicher Text:

Variablen und Strings

Help on class str in module builtins:

```
class str(object)
| str(object='') -> str
| str(bytes_or_buffer[, encoding[, errors]]) -> str
|
| Create a new string object from the given object. If encoding or
| errors is specified, then the object must expose a data buffer
| that will be decoded using the given encoding and error handler.
| Otherwise, returns the result of object.__str__() (if defined)
| or repr(object).
| encoding defaults to sys.getdefaultencoding().
| errors defaults to 'strict'.
|
| Methods defined here:
...
```

Variablen und Strings

- Beispielsweise könnten wir unseren in der Variablen gespeicherten Text komplett in Kleinschreibung erhalten.

```
kursname = ' Python lernen '  
kursname.lower()
```

- Wer sich jetzt denkt, das fühlt sich doch an wie eine Funktion, die einfach alles in Kleinbuchstaben umwandelt, liegt nicht wirklich falsch.
- Es ist einfach eine andere Sichtweise – sprich die objektorientierte Sichtweise.
- Und somit wird die objektorientierte Schreibweise notwendig.

Variablen und Strings

- Zum Scrollen die Pfeiltasten nutzen, zum Beenden der Hilfe einfach „q“ drücken.

Variablen und Strings

Möchte man eine kürzere Übersicht, was uns Strings (sprich „str“) bieten, dann können wir diese über `dir(str)` erhalten:

```
>>> dir(str) ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
['__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
['__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
['__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
['__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
['__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

Alles ohne Unterstriche sind Methoden, die für String-Objekte zur Verfügung stehen. Hier taucht unser `lower` aus dem letzten Beispiel auf.

Variablen und Strings

- Und wollen wir nur genauere Hilfe zu der Methode lower dann bekommen wir diese über die Anweisung `help(str.lower)`

Help on method_descriptor:

`lower(self, /)`

Return a copy of the string converted to lowercase.

- Das war jetzt ein größerer Brocken.

Variablen und Strings

- Wer unseren oberen Code einmal getestet hat, wird sich wundern, dass keine Ausgabe in Kleinschreibung erfolgt ist.
- Das ist auch korrekt so.
- Die Methode lower() wandelt nur in Kleinschreibung um.
- Wollen wir eine Ausgabe, benötigen wir wie gewohnt unsere print()-Anweisung:

```
kursname = 'Python lernen'  
print(kursname.lower())
```

Variablen und Strings

- Am Rande bemerkt: Unser Inhalt der Variable „kursname“ wird nicht verändert! Diese bleibt unberührt.
Einfach mal folgenden Code testen:

```
kursname = ' Python Kurs'  
print(kursname.lower())  
print(kursname)
```

Variablen und Strings

Zusammengefasst:

- Unterschied zwischen Funktionen und Methoden
- Funktionen können direkt über ihren Namen aufgerufen werden – Methoden dagegen benötigen immer ihr Objekt.
- Die Schreibweise:
- Funktionen: `funktionsname()`
- Methoden: `objekt.methode()`
- Funktionen sind unabhängig.
 - Einer Funktion kann beliebiges übergeben werden, mit dem dann weitergearbeitet wird.
 - Methoden sind festgelegt – sprich jedes Objekt verfügt über bestimmte Möglichkeiten (sprich Methoden).
 - Die Kunst ist nur zu wissen, was es an Methoden bereits „standardmäßig“ gibt.

Variablen und Strings

Tipp für Einsteiger

- In den folgenden Unterkapiteln schauen wir uns Variablen (100 % korrekt für Strings) an, welche Methoden existieren und wie diese angewendet werden können.
- Als Einsteiger kann man die Unterkapitel überfliegen, wird sich aber besser später hier nochmals genauer die Möglichkeiten ansehen.
- Bevor man also durch viele kleine Möglichkeiten erschlagen wird, bitte erst den grundlegenden Überblick verschaffen und im nächsten Kapitel „Operatoren für Strings“ ([operatoren-strings.htm](#)) weitermachen und sich später diese Unterkapitel ansehen!

Variablen und Strings

Leerzeichen am Anfang entfernen - lstrip()

- Gerne werden bei Benutzereingaben, vom Nutzer versehentlich, Leerzeichen am Anfang mit eingegeben.
- Leerzeichen auf der linken Seite des Strings (sprich am Anfang der Zeichenkette) können über lstrip (l = left) entfernt werden!
- Englisch am Rande gelernt: „tease“ bedeutet „Verzögerung“.
- „Strip“ hat im Englischen die Bedeutung von „ausziehen, abziehen, abkratzen“ und ist auch die Kurzform von „Striptease“, wobei in Python sofort und nicht kunstvoll langsam, die unerwünschten Zeichen (Dinge;) entfernt werden.

Variablen und Strings

Leerzeichen am Anfang entfernen - lstrip()

- Befehlsaufbau:

```
str.lstrip([Zeichen])
```

- Nehmen wir unseren Beispieltext mit Leerzeichen rechts und links:

```
inhalt = " Python rocks "  
ausgabe = inhalt.lstrip()  
print(ausgabe)
```

- Als Ergebnis erhalten wir einen linksbündigen Text ohne führende Leerzeichen:

```
Python rocks
```

- Die Leerzeichen rechts bleiben bestehen.

Variablen und Strings

Leerzeichen am Anfang entfernen - lstrip()

Wenn wir einen zusätzlichen Text ausgeben lassen, sieht man das:

```
inhalt = "  Python rocks  "  
ausgabe = inhalt.lstrip()  
print(ausgabe + ", daher Python lernen")
```

- Und als Ergebnis:

```
Python rocks  , daher Python lernen
```

Variablen und Strings

Leerzeichen am Anfang entfernen - lstrip()

Parameter nutzen:

Beliebige Zeichen, die entfernt werden

- Wir können mehr als Leerzeichen entfernen lassen! Über den Parameter können wir beliebige Zeichen mitgeben, die links entfernt werden sollen. Im Beispiel haben wir einen merkwürdigen Text mit vielen Zahlen am Anfang. Diese sollen beseitigt werden.

```
inhalt = "321   Python 3 rocks"  
ausgabe = inhalt.lstrip('123')  
print(ausgabe)
```

- Als Ergebnis erhalten wir:

```
Python 3 rocks
```

- Es ist möglich mehrere Zeichen einzugeben, die entfernt werden sollen

Variablen und Strings

Leerzeichen am Anfang entfernen - lstrip()

Es ist möglich mehrere Zeichen einzugeben, die entfernt werden sollen - auch das Leerzeichen!

```
inhalt = "321   Python 3 rocks"  
ausgabe = inhalt.lstrip(' 123456789')  
print(ausgabe)
```

- Es werden also folgende Zeichen entfernt: Zahlen von 1 bis 9 und Leerzeichen:
- Alles am Anfang wird entfernt, bis die Methode auf das Erste nicht zu entfernende Zeichen stößt. Daher bleibt die 3 nach Python mitten im Text stehen!

```
Python 3 rocks
```

Variablen und Strings

Leerzeichen am Anfang entfernen - lstrip()

Alle String-Methoden zum Beseitigen von Leerzeichen:

- **lstrip()** – links Zeichen entfernen (meistens Leerzeichen)
- **strip()** – rechts und links bestimmte Zeichen entfernen (meistens Leerzeichen)
- **rstrip()** – rechts Zeichen entfernen (meistens Leerzeichen)

Variablen und Strings

Leerzeichen am Anfang entfernen - lstrip()

Variablen und Strings

Umschließende Leerzeichen entfernen - strip()

Leerzeichen am Anfang und Ende entfernen über strip()

- Leerzeichen schleichen sich gerne ein – vor allem bei Nutzereingaben über Formulare.
- Da ist schnell ein Leerzeichen am Anfang oder am Ende zu viel, was bei vielen Formularfelder auch für den Nutzer fast nicht sichtbar ist.
- Diese Leerzeichen wollen wir wieder entfernen.
- Dafür gibt es in Python den Befehl strip().
- Die englische Bedeutung ist „ausziehen, abziehen, abschälen und abkratzen“.
- Wir wollen also die unnötigen Leerzeichen abkratzen.

Variablen und Strings

Umschließende Leerzeichen entfernen - strip()

Schauen wir uns den Befehlsaufbau an.

- Befehlsaufbau:

```
str.strip([Zeichen])
```

- Der Parameter ist optional (daher die eckigen Klammern). Der Parameter steht für das Zeichen bzw. die Zeichen, die am Anfang und Ende entfernt werden sollen. Wird hier nichts angegeben, wird das Leerzeichen (der übliche Übeltäter) entfernt. Schauen wir es uns im Beispiel an:

```
inhalt = "  Python lernen  "  
ausgabe = inhalt.strip()  
print(ausgabe)
```

- Alle Leerzeichen um unseren Beispieltext (eine URL, in der es einfach auch Leerzeichen geben darf) sind entfernt.

```
Python lernen
```

Variablen und Strings

Umschließende Zeichen entfernen - strip()

beliebige umschließende Zeichen entfernen durch strip()

- Das Tolle an der Python-Methode ist, dass nicht nur Leerzeichen entfernt werden können. Es können beliebige Zeichen entfernt.
- Kommt unser Text aus einer Aufzählung z.B. heraus, können wir die Zahlen und den Aufzählungspunkt auch entfernen:

```
inhalt = "1.) Python lernen "  
ausgabe = inhalt.strip('1')  
print(ausgabe)
```

- Das Ergebnis, das noch nicht befriedigend ist:

```
.) Python lernen
```


Variablen und Strings

Umschließende Leerzeichen entfernen - strip()

- Wir wollen auch keine Punkte, Klammern und Leerzeichen. Also geben wir alle unerwünschten Zeichen als Parameter an:

```
inhalt = "1.) python lernen "  
ausgabe = inhalt.strip('.) 1234567890')  
print(ausgabe)
```

- Und nun haben wir unser gewünschtes Ergebnis:

```
Python lernen
```

Variablen und Strings

UmschließendeLeerzeichen entfernen - strip()

- Auch wenn wir alle Punkte entfernen, dann bezieht sich das nur auf die Punkte am Anfang und Ende!
 - Die strip()-Methode hört sofort mit dem entfernen auf, sobald das erste erwünschte Zeichen kommt.
 - Sprich unsere Punkte innerhalb der URL bleiben erhalten.
 - Das wäre anders, wenn wir einen deutschen Satz mit einem Punkt am Ende hätten.
 - Dieser Punkt würde bei unserem Beispielaufbau der strip()-Methode zum Opfer fallen.
-
- Langer Rede kurzer Sinn: diese Methode wird überaus häufig in der Praxis genutzt.

Variablen und Strings

Leerzeichen rechts entfernen über `rstrip()`

- Gerne werden bei Benutzereingaben, vom Nutzer versehentlich, Leerzeichen am Anfang oder Ende mit eingegeben.
- Leerzeichen auf der rechten Seite eines Strings (sprich am Ende der Zeichenkette) können über `rstrip()` entfernt werden.
- Das „r“ am Anfang steht für rechts (englisch „right“).

Variablen und Strings

Leerzeichen rechts entfernen über `rstrip()`

Befehlsaufbau:

```
str.rstrip([Zeichen])
```

- Nehmen wir unseren Beispieltext mit Leerzeichen rechts und links:

```
inhalt = " Python 3 rocks "  
ausgabe = inhalt.rstrip()  
print(ausgabe + ", daher Python lernen")
```

- Für die bessere Sichtbarkeit des Effekts lassen wir einen zusätzlichen Text ausgeben.
- Wir erhalten folgende Ausgabe:

```
Python 3 rocks, daher Python lernen
```

Variablen und Strings

beliebige Zeichen rechts entfernen `rstrip()`

Parameter nutzen:

Beliebige Zeichen, die entfernt werden

- Wir können mehr als Leerzeichen entfernen lassen!
- Über den Parameter können wir alle gewünschten Zeichen mitgeben, die rechts entfernt werden sollen.
- Jetzt haben wir zum Beispiel eine merkwürdige Eingabe mit vielen 4ern am Ende. Diese sollen beseitigt werden.

```
inhalt = " Python 3 rocks 1233 4444"  
ausgabe = inhalt.rstrip('4')  
print(ausgabe + ", daher Python lernen")
```

- Als Ergebnis erhalten wir:

```
Python 3 rocks 1233 , daher Python lernen
```

Variablen und Strings

beliebige Zeichen rechts entfernen über `rstrip()`

- Es ist möglich mehrere Zeichen einzugeben, die entfernt werden sollen.

```
inhalt = " Python 3 rocks 1233 4444"  
ausgabe = inhalt.rstrip('1234 ?XYZ')  
print(ausgabe + ", daher Python lernen")
```

- Es werden also folgende Zeichen entfernt: Zahlen von 1 bis 9, Fragezeichen, Leerzeichen und „XYZ“ (aber nur in Großschreibung):

```
Python 3 rocks, daher Python lernen
```

- Alles am rechten Ende wird entfernt, bis die Methode auf das Erste nicht zu entfernende Zeichen stößt.
- Daher bleibt die 3 nach Python mitten im Text stehen!

Variablen und Strings

Zeilenumbruch entfernen über `rstrip()`

- Entfernen von Zeilenumbruch, Newline `\r\n` mit Python
- Die `rstrip()`-Methode von Python entfernt standardmäßig alle Arten von nachgestellten Leerzeichen. Dazu gehören nicht Zeilenumbruch und neue Zeilen, die über den Steuercode `\r\n` übertragen werden.

```
inhalt = " Python rocks \n \r\n "  
ausgabe = inhalt.rstrip()  
print(ausgabe + ", damit sichtbar wird, was gelöscht wurde")
```

- Das Ergebnis:

```
Python rocks  
  
, damit sichtbar wird, was gelöscht wurde
```

Variablen und Strings

Zeilenumbruch entfernen über rstrip()

Möchte man gezielt einen bestimmten Zeilenumbruch nur entfernen, muss dieser als Parameter mit Übergeben werden:

```
inhalt = " Python rocks \n \r\n "  
ausgabe = inhalt.rstrip('\n ')  
print(ausgabe + ", damit sichtbar wird, was gelöscht  
wurde")
```

- Als Ergebnis werden alle Umbruch (\n) und Leerzeichen gelöscht. Ein Umbruch (\r) bleibt erhalten:

```
Python rocks  
, damit sichtbar wird, was gelöscht wurde
```


Variablen und Strings

Zeilenumbruch und Leerzeichen entfernen über `rstrip()`

- Sollen alle Zeilenumbruch und Leerzeichen entfernt werden, müssen alle 3 Angaben in `rstrip('\n \r')` gemacht werden! Dabei ist die Reihenfolge egal. Es funktioniert genauso: `rstrip(' \r\n')`

```
inhalt = " Python rocks \n \r\n "  
ausgabe = inhalt.rstrip('\n \r')  
print(ausgabe + ", damit sichtbar wird, was gelöscht  
wurde")
```

- Das Ergebnis:

```
Python rocks, damit sichtbar wird, was gelöscht wurde
```

Variablen und Strings

Zwischenstand

Wo kann ich diese Funktionen nutzen? Oder wozu brauche ich Sie?

- Denke an benutzereingaben!
- 1. Regel ist, „Traue keiner ungeprüften Eingaben!“
 - Bekommt deine Funktion Eingaben, ist es zwar lästig, jedoch nötig!
 - Dies nennt sich sanitizing!
 - schneide Sonderzeichen aus Benutzereingaben und unterbinde an solchen Stellen die Code-eingabe.
 - Schneide die Eingabe ggf auf die erlaubte länge zur Weiterverarbeitung

Variablen und Strings

Lernzielkontrolle Strings1.txt

- Aufgabe:
- Kopiere dir die Test-Strings aus dem Projektordner auf dem Trainerserver.
 - Lade die TestStrings1.list.txt herunter und speichere Sie in deinem Projektordner unter `_lists`
- Diese Sollen sanitized werden.
 - Dies soll Hacking unterbinden!
- Lege dir Sanitizer1.py in deinem Projektordner Lesson1/Strings/ an.

Variablen und Strings

Lernzielkontrolle Strings1.txt

- Entwickle eine Sanitizer1.py, die jeden String der ankommt
 - Auf seine Länge prüft,
 - Alle Sonderzeichen für das Programmieren entfernt
 - Den Text von führenden und abschließenden Leerzeichen befreit
- Kopiere dir hierzu jeden einzelnen String in die Variable „teststing“
- Prüfe hiernach mit einer Debugausgabe den Zwischenstand und gib ihn aus.

Variablen und Strings

Lernzielkontrolle Strings1.txt

- Das Programmkonstrukt Hierfür ist folgendes:

```
teststring = ' dies ist ein Teststring(); \r\nmit Klammern und <html tags>! \n und Umbrüchen - ,'  
  
sani1 = teststring.strip("")  
print('Bereinigter String:\n' + sani1)  
  
sani2 = teststring.strip("")  
print('Bereinigter Text 2:\n' + sani2)  
  
sani3 = sani1.strip("")  
print('Bereinigter Text 3:\n' + sani3)  
  
sani4 = sani3.strip("")  
print('Bereinigter Text 4\n' + sani4)
```

Variablen und Strings

Lernzielkontrolle Strings1.txt

- Das Programmkonstrukt Hierfür ist folgendes: Auflösung

```
teststring = ' dies ist ein Teststring(); \n\nmit Klammern und <html tags>! \n und Umbrüchen - ,  
  
#Leerzeichen am Anfang und Ende, sicher entfernen  
sani1 = teststring.strip()  
print('Bereinigter String:\n' + sani1)  
  
#Klammern entfernen !  
sani2 = teststring.strip()  
print('Bereinigter Text 2:\n' + sani2)  
  
# Klammern und Leerzeichen entfernen  
sani3 = sani1.strip()  
print('Bereinigter Text 3:\n' + sani3)  
  
# Sonderzeichen für Programmierung am Ende entfernen!  
# String 3 weiterverarbeiten  
sani4 = sani3.strip()  
print('Bereinigter Text 4\n' + sani4)
```

Variablen und Strings

Lernzielkontrolle Strings1.txt

- Auflösung:
- Die Klammern können wir so nicht erreichen, diese können wir später entfernen.
- Die „–“ und „;“ Zeichen und endende Tags können wir bereinigen
- Die Leerzeichen am Anfang können wir bereinigen
- Hierzu

Variablen und Strings

Lernzielkontrolle Strings1.txt

Variablen und Strings

Linksbündig ausgeben, rechts auffüllen mit Zeichen ljust()

Die String Methode ljust() füllt rechts mit vorbestimmten Zeichen auf.

- Dabei haben wir beim Methodennamen wieder die für Python typischen Abkürzung von englischen Begriffen.
- Die Bedeutung von „left justify“ ist „linksbündig“.
- Das mag auf den ersten Blick irritieren, da die Methode rechts auffüllt.
- Wichtig ist, dass der Text links platziert ist und somit linksbündig.
 - Schauen wir uns den Befehlsaufbau ab:

```
str.ljust(Breite[, Füllzeichen])
```

Variablen und Strings

Linksbündig ausgeben, rechts auffüllen mit Zeichen ljust()

Schauen wir unser erstes Beispielprogramm an.

- Wir lassen uns das Wort „Vier“ ausgeben, dass 4 Buchstaben breit ist. Es bekommt einen Platz von 10 über ljust():

```
inhalt = "Vier"  
ausgabe = inhalt.ljust(10)  
print(ausgabe)
```

- In der Ausgabe sieht man nicht direkt die Auswirkung von ljust():

```
Vier
```

Variablen und Strings

Linksbündig ausgeben, rechts auffüllen mit Zeichen ljust()

Würde danach gleich ein Text kommen, wäre die Auswirkung sichtbar:

```
inhalt = "Vier"  
ausgabe = inhalt.ljust(10)  
print(ausgabe, "mehr Text")
```

- Und nun ist der Abstand 6 Zeichen – 4 Zeichen von unserem Beispielwort minus den 10 vorgegebenen Zeichen ergibt 6 Leerzeichen:

```
Vier      mehr Text
```

Variablen und Strings

Linksbündig ausgeben, rechts auffüllen mit Zeichen ljust()

Wir können bei der Methode noch die Füllzeichen mitgeben und damit wird das Auszählen der Abstände einfacher:

```
inhalt = "Vier"  
ausgabe = inhalt.ljust(10, '.')  
print(ausgabe, "mehr Text")
```

- Und das Ergebnis:

```
Vier..... mehr Text
```

- Hier taucht ein Leerzeichen zwischen unserem weiteren Text und den 10 Zeichen von ljust() auf.

Variablen und Strings

Linksbündig ausgeben, rechts auffüllen mit Zeichen ljust()

Hier tauchte ein Leerzeichen zwischen unserem weiteren Text und den 10 Zeichen von ljust() auf.

- Durch eine Verknüpfung mit „+“ wird dies nicht mehr erscheinen:

```
inhalt = "Vier"  
ausgabe = inhalt.ljust(10, '.')  
print(ausgabe + "mehr Text")
```

- Allerdings hat der Inhalt Vorrang vor Breitenangabe. Ist der Inhalt breiter als die mitgegebene Breite, wird der komplette Inhalt gefolgt von einem Leerzeichen ausgegeben:

```
inhalt = "Vier"  
ausgabe = inhalt.ljust(2, '.')  
print(ausgabe, "mehr Text")
```

- Und als Ausgabe erhalten wir:

```
Vier mehr Text
```

Variablen und Strings

Zentrierte Ausgabe über center()

- Über die String-Methode `.center()` kann ein Text zentriert ausgegeben werden.
- Dabei wird eine Länge vorgegeben, in der vorhandener Text mittig platziert wird.
- Zusätzlich kann auch ein beliebiges Füllzeichen mitgegeben werden.
- Wird kein Füllzeichen mitgegeben, wird ein Leerzeichen als Füllzeichen verwendet.

- Beispielcode:

```
inhalt = "mittig"  
print( inhalt.center(12) )
```

- Als Ergebnis erhalten wir:

```
mittig
```

Variablen und Strings

Zentrierte Ausgabe über center()

- Besser sichtbar ist die Arbeitsweise von zentrierten Ausgaben, wenn wir ein anderes Füllzeichen als das Leerzeichen wählen:
- Beispielcode:

```
inhalt = "mittig"  
print( inhalt.center(12,"^") )
```

- Als Ergebnis erhalten wir:

```
^^^mittig^^^
```

- Unser Beispielwort ist 6 Zeichen lang und wir wollen es zentriert auf 12 Zeichen ausgeben.
- Dann kommen 3 Füllzeichen rechts und 3 Füllzeichen links neben unserem Beispielwort.

Variablen und Strings

Zentrierte Ausgabe über center()

- Geht es nicht so schön auf, dass wir die gleiche Anzahl von Füllzeichen links wie rechts haben, wird auf einer Seite ein Füllzeichen mehr ausgegeben:

```
inhalt = "mittig"  
print( inhalt.center(11,"^") )
```

- Bringt als Ergebnis:

```
^^^mittig^^
```

- Lustigerweise ist die Verteilung, ob das Füllzeichen rechts oder links von unserem Text mehr ausgegeben wird, abhängig davon, ob wir eine gerade oder ungerade Anzahl von Zeichen bei unserem Text haben.

Variablen und Strings

Zentrierte Ausgabe über center()

- Lustigerweise ist die Verteilung, ob das Füllzeichen rechts oder links von unserem Text mehr ausgegeben wird, abhängig davon, ob wir eine gerade oder ungerade Anzahl von Zeichen bei unserem Text haben. Einfach einmal probieren.

```
inhalt = "Mitte"  
print( inhalt.center(8,"^") )
```

- Ergibt dann rechts mehr Füllzeichen:

```
^Mitte^^
```

Variablen und Strings

Zentrierte Ausgabe über center()

nur ein Füllzeichen möglich

- Kommt man auf die Idee, mehrere Füllzeichen eingeben zu wollen, erhält man die Fehlermeldung: „TypeError: The fill character must be exactly one character long“.

Variablen und Strings

Zentrierte Ausgabe über center()

zu wenig Zeichen zum Zentrieren

- Wird als Breite zu wenig Zeichen angegeben, wird trotzdem eine komplette Ausgabe des Textes stattfinden:

```
inhalt = "Mitte"  
print(inhalt.center(2,"^"))
```

- Unser Beispielwort „Mitte“ benötigt mindestens 5 Zeichen, bekommt aber im Beispiel nur 2 zur Verfügung gestellt. Macht nichts, da wir die Ausgabe des kompletten Beispielwortes erhalten, was allerdings nicht zentriert werden kann.

```
Mitte
```

Variablen und Strings

Rechtsbündig ausgeben, links auffüllen mit Zeichen: rjust()

- Die String Methode rjust() füllt links mit vorbestimmten Zeichen auf. Dabei haben wir beim Methodennamen wieder die für Python typischen Abkürzung von englischen Begriffen. Die Bedeutung von „right justify“ ist „rechtsbündig“. Schauen wir uns den Befehlsaufbau ab:

```
str.rjust(Breite[, Füllzeichen])
```

- Wird kein Füllzeichen festgelegt, wird ein Leerzeichen verwendet.

Variablen und Strings

Rechtsbündig ausgeben, links auffüllen mit Zeichen: rjust()

- Schauen wir unser erstes Beispielprogramm an. Wir lassen uns das Wort „Vier“ ausgeben, dass 4 Buchstaben breit ist. Es bekommt einen Platz von 10 über rjust():

```
inhalt = "Vier"  
ausgabe = inhalt.rjust(10)  
print(ausgabe)
```

- In der Ausgabe sieht man direkt die Auswirkung von rjust(). Es werden 6 Leerzeichen vor dem Wort „Vier“ ausgegeben:

```
Vier
```

Variablen und Strings

Rechtsbündig ausgeben, links auffüllen mit Zeichen: rjust()

Nachfolgender Text kommen mit einem Leerzeichen davor:

```
inhalt = "Vier"  
ausgabe = inhalt.rjust(10)  
print(ausgabe, ", weiterer Text")
```

- Und nun der Abstand von 6 Zeichen (4 Zeichen von unserem Beispielwort minus den 10 vorgegebenen Zeichen ergibt 6 Leerzeichen):

```
Vier , weiterer Text
```

- Hier fällt das Leerzeichen vor unserem „weiteren Text“, sprich vor dem Komma auf.

Variablen und Strings

Rechtsbündig ausgeben, links auffüllen mit Zeichen: rjust()

Durch eine Verknüpfung mit „+“ wird dies nicht mehr erscheinen:

```
inhalt = "Vier"  
ausgabe = inhalt.rjust(10)  
print(ausgabe + ", weiterer Text")
```

- Ergebnis:

```
Vier, weiterer Text
```

Variablen und Strings

Rechtsbündig ausgeben, links auffüllen mit Zeichen: `rjust()`

- Wir können bei der Methode noch die Füllzeichen mitgeben und damit wird das Auszählen der Abstände einfacher:

```
inhalt = "Vier"  
ausgabe = inhalt.rjust(10, '.')  
print(ausgabe + ", weiterer Text")
```

- Und das Ergebnis:

```
.....Vier, weiterer Text
```

- Allerdings hat der Inhalt Vorrang vor Breitenangabe.

Variablen und Strings

Rechtsbündig ausgeben, links auffüllen mit Zeichen: `rjust()`

- Ist der Inhalt breiter als die mitgegebene Breite, wird der komplette Inhalt gefolgt von einem Leerzeichen ausgeben:

```
inhalt = "Vier"  
ausgabe = inhalt.rjust(2, '.')  
print(ausgabe + ", weiterer Text")
```

- Und als Ausgabe erhalten wir:

```
Vier, weiterer Text
```

Variablen und Strings

Alle String-Methoden für formatierte Ausgabe

- `ljust()` = String wird linksbündig zurückgeliefert (Füllzeichen möglich)
- `center()` = String wird zentriert ausgegeben (Füllzeichen möglich)
- `rjust()` = String wird rechtsbündig zurückgeliefert (Füllzeichen möglich)
- `zfill()` = String wird mit Nullen (Zero) aufgefüllt

Variablen und Strings

Führende 0 am Anfang auffüllen über .zfill()

Führende 0 am Anfang auffüllen über .zfill()

- Über die Methode `zfill()` werden am Anfang Nullen auffüllen. Es muss ein Parameter für die gewünschten Anzahl aufzufüllenden Stellen mitgegeben werden.
- Dabei steht das „z“ bei der Methode für „Zero“ – also mit „Nullen füllen“ wenn man den englischen Begriff übersetzt.
- Beispiel:

```
text = "20"  
print(text.zfill(8))
```

- Als Ergebnis erhalten wir:

```
00000020
```

Variablen und Strings

Führende 0 am Anfang auffüllen über .zfill()

- Wird der Parameter vergessen, erhält man als Fehlermeldung:
„TypeError: zfill() takes exactly one argument (0 given)“
- Es muss auch ein String übergeben werden!
- Man denkt zwar beim Auffüllen mit Nullen immer an Zahle
- allerdings werden wir auch dann eine Fehlermeldung erhalten, wenn wir eine Zahl anstelle eines Strings übergeben würden (zahl = 20).
- Als Fehlermeldung kommt dann:
„AttributeError: 'int' object has no attribute 'zfill'“

Variablen und Strings

Führende 0 am Anfang auffüllen über .zfill()

Inhalt länger als Anzahl der aufzufüllenden Nullen

- Wenn unser Inhalt länger ist als die Anzahl der aufzufüllenden Nullen, kommt einfach unser Inhalt und keine Nullen.

```
text = "Inhalt länger als 8 Zeichen"  
print(text.zfill(8))
```

- Ergebnis:

```
Inhalt länger als 8 Zeichen
```

Variablen und Strings

Führende 0 am Anfang auffüllen über .zfill()

Führendes +/- Zeichen - wird nach vorne geschoben

Wenn es ein führendes +/- Zeichen gibt, werden die anzuhängenden Nullen nach den +/- Zeichen aufgefüllt.

```
text = "-123"  
print(text.zfill(8))
```

Somit erhalten wir als Ergebnis:

```
-0000123
```

Variablen und Strings

Führende 0 am Anfang auffüllen über .zfill()

Alternativen zu zfill() - rjust()

- Die Methode .rjust() kann auch einen String auffüllen, als weiteren Vorteil können wir aber das Zeichen angeben, mit dem der String aufgefüllt werden soll. Hierzu folgendes Beispiel:

```
text = "-123"  
print(text.rjust(8,"0"))
```

- Als Ergebnis erhalten wir:

```
0000-123
```

Variablen und Strings

Umwandeln in Großschreibung über die String-Methode .upper()

Wollen wir alle Buchstaben eines Strings in Großbuchstaben, können wir über die Methode upper() diese umwandeln:

```
inhalt = "Hier kommt ein String-Inhalt"  
grossbuchstaben = inhalt.upper()  
print ( grossbuchstaben )
```

- Als Ausgabe erhalten wir:

```
HIER KOMMT EIN STRING-INHALT
```


Variablen und Strings

Umwandeln in Kleinschreibung über die String-Methode .lower()

Alles wird in Kleinschreibung umgewandelt:

```
inhalt = "Hier kommt ein String-Inhalt"  
kleinbuchstaben = inhalt.lower()  
print ( kleinbuchstaben )
```

- Als Ausgabe erhalten wir:

```
hier kommt ein string-inhalt
```

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

- Eine weitere String-Methode zur Umwandlung in Kleinbuchstaben. Aber warum zur Hölle benötigen wir neben der Methode .lower() eine weitere Methode? Hier kommt das schöne Wort „aggressiv“ zum Tragen.
- casefold() ändert unter Umständen auch Zeichenfolgen.
- Was geht da vor.
- Normalerweise wird man sich denken, zu jedem Großbuchstaben gibt es einen Kleinbuchstaben.
- Aus einem großen „A“ wird einfach ein kleines „a“.

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

- Aber was ist beispielsweise mit „ß“? Schauen wir uns an, was casefold() uns daraus bastelt:

```
text = "Inhalt mit Umlauten: ÄÖÜß"  
print("Originaltext:")  
print(text)  
print()  
print("Umwandlung durch lower:")  
print(text.lower())  
print()  
print("Umwandlung durch casefold:")  
print(text.casefold())
```

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

- Als Ergebnis erhalten wir:

Originaltext:

Inhalt mit Umlauten: ÄÖÜß

Umwandlung durch lower:

inhalt mit umlauten: äöüß

Umwandlung durch casefold:

inhalt mit umlauten: äöüss

- Aus einem „ß“ wird korrekt ein „ss“.

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

- Aus einem „ß“ wird korrekt ein „ss“.
- Hier werden als nationale Besonderheiten berücksichtigt.
- Wer es genau benötigt, kann sich die Zusammenstellung beim Unicode-Konsortium herunterladen (siehe https://unicode.org/faq/casemap_charprop.html unter CaseFolding.txt).

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

- Hier als kleine Übung wofür man casefold() sinnvoll einsetzen können.
- Es bietet uns die Möglichkeit zur Normalisierung von Text und wir können von einem vorliegenden Text überprüfen, ob dieser ein Palindrom ist.
- Was ist ein Palindrom?
- Für alle, bei denen der Schulunterricht schon ein paar Tage zurückliegt:
 - Übersetzt bedeutet Palindrom „rückwärts laufend“.
 - So ist der Name „Otto“ und „Hannah“ rückwärts geschrieben exakt das Gleiche.

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

Aufgabe Palindrom über casefold() testen

- Aufgabe: Testen, ob bei dem Wort „Rentner“ ein Palindrom vorliegt.
- Erst selber ein Python-Programm erstellen und dann die Lösung ansehen. Wir benötigen dazu auch die Funktion `reversed(str)`, `.join()` und `list()`, die uns die Reihenfolge umdreht.

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

Lösung Palindrom mit casefold()

- Als Erstes drehen wir die Reihenfolge um und lassen uns zur Kontrolle den Inhalt ausgeben:

```
text = "Rentner"  
text = text.casefold()  
rueckwarts = reversed(text)  
print(text)  
print(rueckwarts)
```

- Wir erhalten bei unserem reversed() ein „reversed object at ...“.

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

- Als Objekt können wir es nicht mit einem String vergleichen. Also erstellen wir über .join einen String.

```
text = "Rentner"  
text = text.casefold()  
rueckwarts = "".join(reversed(text))
```

Variablen und Strings

aggressive Umwandlung in Kleinbuchstaben .casefold()

- Und jetzt können wir bequem vergleichen:

```
text = "Rentner"
text = text.casefold()
rueckwarts = "".join (reversed(text))

if text == rueckwarts:
    print(text, " ist ein Palindrom")
else:
    print("KEIN Palindrom")
```

- Ach ja – das Wort „Rentner“ ist ein Palindrom.

Variablen und Strings

Weitere Methoden zur Umwandlung zwischen Groß- und Kleinschreibung

Alle String-Methoden für die Umwandlung bei Klein- und Großschreibung:

- `casefold()`, `upper()`, `lower()`, `capitalize()`, `title()`, `swapcase()`

Variablen und Strings

Erster Buchstaben in Großschreibung über `capitalize()` – Rest klein!

Nur der erste Buchstabe wird in Großschreibung, der Rest wird aber in Kleinbuchstaben umgesetzt bei der Python Methode `capitalize()`!

- Schauen wir uns unsere Beispiele an:

```
vornachname = "Rolf von und zu Maier-Müller"  
umgewandelt = vornachname.capitalize()  
print(umgewandelt)
```

- Und als Ergebnis erhalten wir:
- Rolf von und zu maier-müller
- Die Schreibweise vom Text ist gleichgültig. Alle Zeichen, egal wie diese im Ursprungstext geschrieben wurden, werden bis auf das erste Zeichen in Kleinbuchstaben umgesetzt.

Variablen und Strings

Erster Buchstaben in Großschreibung über `capitalize()` – Rest klein!

Die Schreibweise vom Text ist gleichgültig.

- Alle Zeichen, egal wie diese im Ursprungstext geschrieben wurden, werden bis auf das erste Zeichen in Kleinbuchstaben umgesetzt.
- Nur der erste Buchstabe wird als Großbuchstaben ausgegeben:

```
zeichenkette = "hleR kOmMt TeXT"  
print(zeichenkette.capitalize())
```

- Als Ergebnis erhalten wir:

```
Hier kommt text
```

Variablen und Strings

Erster Buchstaben in Großschreibung über `capitalize()` – Rest klein!

Sonderfälle bei Großbuchstaben

- Sofern möglich wird das erste Zeichen in einen Großbuchstaben umgewandelt.
- Das funktioniert natürlich nur bei Buchstaben – wobei es auch hier Sonderfälle gibt.
- Probieren wir es bei Zahlen und Vorzeichen.
- Hier haben wir keinen Unterschied bei Groß- und Kleinschreibung.
- Das wäre jetzt nicht weiter Erwähnenswert, wenn es nicht den Buchstaben „ß“ (Eszett) gäbe.
- Aber der Reihe nach.

Variablen und Strings

Erster Buchstaben in Großschreibung über `capitalize()` – Rest klein!

Schauen wir uns das Verhalten von `capitalize()` bei Zahlen und Vorzeichen an:

```
zeichenkette = "123 Text"  
print(zeichenkette.capitalize())
```

Als Ausgabe erhalten wir:

```
123 text
```

Variablen und Strings

Erster Buchstaben in Großschreibung über `capitalize()` – Rest klein!

Ober bei Vorzeichen:

```
zeichenkette = "+123 Text"  
print(zeichenkette.capitalize())
```

Auch bei Vorzeichen bleibt das Ursprungszeichen einfach bestehen:

```
+123 text
```


Variablen und Strings

Erster Buchstaben in Großschreibung über `capitalize()` – Rest klein!

Sonderfall „ß“ und `capitalize()`!

- Interessant ist die Umwandlung bei `capitalize()` von dem Sonderfall mit dem deutschen „ß“:

```
zeichenkette = "ß Text"  
print(zeichenkette.capitalize())
```

- Als Ergebnis erhalten wir:

```
Ss text
```

- Wobei das Große Eszett es seit 2017 auch offiziell als Großbuchstaben gibt!
- Und dieses hat definitiv nicht die Umsetzung in „Ss“ –
- wobei hier schätzungsweise wieder nationale Sonderwege gibt.
- Vielleicht ist es bei den Schweizern oder Österreichern in dieser Form üblich.

Variablen und Strings

Erster Buchstaben in Großschreibung über `capitalize()` – Rest klein!

Bei Umlauten, die über eine Großschreibung verfügen, funktioniert es problemlos:

```
zeichenkette = "äöü Text"  
print(zeichenkette.capitalize())
```

Bisher ist mir keine direkte Anwendung beim Programmieren unter die Finger gekommen.

Aber wer weiß, irgendwann und irgendwer wird die Methode `capitalize()` schon sinnvoll benutzen können.

```
Äöü text
```

Variablen und Strings

Jeder Anfangsbuchstaben in Großschreibung über title()

Jeder Anfangsbuchstabe wird in Großschreibung umgesetzt. Schauen wir uns unser Beispiel an:

```
vornachname = "Rolf von und zu Maier-Müller"  
umgewandelt = vornachname.title()  
print(umgewandelt)
```

Und als Ergebnis erhalten wir:

```
Rolf Von Und Zu Maier-Müller
```

Das könnte man bei „normalen“ Vor- und Nachnamen machen, aber sobald ein „von“ oder „van“ dazukommt, läuft es schief.

Variablen und Strings

Groß- und Kleinschreibung wird vertauscht: `swapcase()`

Alles was ursprünglich ein Großbuchstabe war wird zum Kleinbuchstaben und andersherum.

- Am Beispiel wird es deutlich:

```
vornachname = "Rolf von und zu Maier-  
Müller"  
umgewandelt = vornachname.swapcase()  
print(umgewandelt)
```

- Und als Ergebnis erhalten wir:

```
rOLF VON UND ZU mAIER-mÜLLER
```

Variablen und Strings

Zeichen ersetzen/austauschen mit Python: String replace()

String replace('alt', 'neu', [Anzahl])

- Bei der Arbeit mit Strings ist es oft notwendig, Zeichenketten bzw. Teile des Inhalts auszutauschen.
- Python bietet mit der Funktion `variable.replace("alt", "neu")` eine einfache Möglichkeit.
- Die Funktion zum Austauschen bzw. Ersetzen von Zeichenfolgen ist nicht auf einzelne Buchstaben begrenzt.

Variablen und Strings

Zeichen ersetzen/austauschen mit Python: String replace()

- Im folgenden Beispiel wollen wir in dem String alle DM durch Euro ersetzen (wenn das auch schon ein paar Tage her ist).

```
ausgabertext = "Der Preis für 2 Socken beträgt 5 DM und 5 Paar kosten 10 DM"  
print(ausgabertext)  
ausgabertext = ausgabertext.replace("DM", "Euro")  
print("Nach dem Austauschen über replace():")  
print(ausgabertext)
```

- Als Ergebnis erhalten wir:

```
Der Preis für 2 Socken beträgt 5 DM und 5 Paar kosten 10 DM  
Nach dem Austauschen über replace():  
Der Preis für 2 Socken beträgt 5 Euro und 5 Paar kosten 10 Euro
```

Variablen und Strings

Zeichen ersetzen/austauschen mit Python: String replace()

- Parameter von `replace('x', 'y', [Anzahl])`
- Der dritte Parameter für die Anzahl ist Optional. Hier können wir festlegen, wie viele Vorkommen des alten Wertes ersetzt werden soll. Wird dieser Parameter nicht angegeben, werden alle Vorkommen ersetzt.
- Beispiel: es sollen nur die erste 3 Vorkommen ersetzt werden:

```
ausgabertext = "Der Preis für 2 Socken beträgt 2 DM und 2 Paar kosten 3.50 DM"  
ausgabertext = ausgabertext.replace("DM", "Euro")  
ausgabertext = ausgabertext.replace("2", "zwei", 2)  
print("Nach dem Austauschen über replace():")  
print(ausgabertext)
```

- Als Ergebnis erhalten wir:

```
Der Preis für zwei Socken beträgt zwei Euro und 2 Paar kosten 3.50 Euro
```

Variablen und Strings

Zeichen ersetzen/austauschen mit Python: String replace()

Mehrere Ersetzungen durchführen

- Bei der Methode replace() können wir nur eine Ersetzung mitgeben.
- Was aber, wenn mehrere Ersetzungen gewünscht sind.
- In unserem Beispiel sollen die Zahl 1,2 und 3 ausgeschrieben werden.
- Es soll also jede „1“ durch „eins“ und jede „2“ durch „zwei“ und jede „3“ durch „drei“ ersetzt werden.
- Welche Möglichkeiten haben wir dafür?

```
ausgabertext = "1 1 2 2 3 3 4 4"  
ausgabertext = ausgabertext.replace("1", "eins")  
ausgabertext = ausgabertext.replace("2", "zwei")  
ausgabertext = ausgabertext.replace("3", "drei")  
print("Nach dem Austauschen über replace():")  
print(ausgabertext)
```

- Als Ergebnis erhalten wir die gewünschten Ersetzungen:

```
eins eins zwei zwei drei drei 4 4
```


Variablen und Strings

Zentrierte Ausgabe über center()

zweite Variante für mehrfache Ersetzungen

- Die zweite Möglichkeit ist, einfach unsere Aufrufe hintereinander durchzuführen:

```
ausgabertext = "1 1 2 2 3 3 4 4"  
ausgabertext =  
ausgabertext.replace("1","eins").replace("2","zwei").replace("3","drei")  
print("Nach dem Austauschen über replace():")  
print(ausgabertext)
```

- Als Ergebnis erhalten wir wieder die gewünschten Ersetzungen:

```
eins eins zwei zwei drei drei 4 4
```

Variablen und Strings

Zählen von bestimmten Vorkommen über die String-Methode .count()

- Wenn man wissen möchte, wie oft etwas in einem String vorkommt, hilft die Methode .count() in Python weiter.
- In den Klammern gibt man das Gesuchte ein.
- Will man zum Beispiel von unserem String mit den Variablennamen „inhalt“ wissen, wie viele „i“ in diesem vorhanden sind bekommt man das im Handumdrehen über folgenden Code:

```
inhalt = "Hier kommt ein String-Inhalt"  
print ( inhalt.count("i") )
```

- Als Rückmeldung erhält man „3“.
- Der Buchstabe „i“ kommt also 3-mal in unserem String vor.

Variablen und Strings

Zählen von bestimmten Vorkommen über die String-Methode .count()

- Wir können auch nach mehr als einem Buchstaben suchen.
- Natürlich gehen ganze Wörter oder auch Wortteile wie z.B. „in“.

```
inhalt = "Hier kommt ein String-Inhalt"  
print ( inhalt.count("in") )
```

- Als Rückmeldung kommt:

```
2
```

- Das Wort „in“ kommt 2-mal vor in unserem String:

```
Hier kommt ein String-Inhalt
```

Variablen und Strings

Zählen von bestimmten Vorkommen über die String-Methode `.count()`

- Bei dem Wort „Inhalt“ haben wir ein weiteres „in“, das allerdings mit Großschreibung beginnt.
- Mit der Großschreibung wird es allerdings nicht gefunden, sprich das „In“ von dem Wort „Inhalt“ wurde nicht gefunden und somit auch nicht mitgezählt.
- Es gibt bei der Methode `.count()` keinen optionalen Parameter, aber eine entsprechende Methode zur Umwandlung in Kleinbuchstaben haben wir bereits kennengelernt, mit der wir dies erreichen.

Variablen und Strings

Zählen von bestimmten Vorkommen über die String-Methode .count()

- Gemeint ist die Methode .lower().
- Hier unser Beispiel in Teilschritten:

```
inhalt = "Hier kommt ein String-Inhalt"  
kleinbuchstaben = inhalt.lower()  
print ( kleinbuchstaben )  
print ( kleinbuchstaben.count("in") )
```

Variablen und Strings

Zählen von bestimmten Vorkommen über die String-Methode .count()

Optionale Parameter bei .count("teilstring", index_anfang=..., index_ende=...)

- Wollen wir nur innerhalb der ersten 15 Zeichen den String überprüfen lassen, können wir dies über die optionalen Parameter für den Bereich (also Anfangspunkt und Endpunkt) erreichen:

```
inhalt = "Hier kommt ein String-Inhalt"  
print ( inhalt.count("in", 0, 15) )
```

- Es wird dann nur der Teil des Strings mit dem Inhalt „Hier kommt ein“ ausgewertet und dort wird dann einmal der gesuchte Teilstring „in“ gefunden im letzten Wort „ein“.

Weitere Nutzungsmöglichkeiten

- Die Methode .count() kann genauso bei Listen eingesetzt werden!

Variablen und Strings

erstes Vorkommen bestimmen über die String-Methode .find()

- Wollen wir das erste Vorkommen bestimmen, können wir die Methode find() nutzen:

```
inhalt = "Hier kommt ein String-Inhalt"  
print ( inhalt.find("e") )
```

- Wir bekommen als Feedback:

```
2
```

- Die Zählung beginnt bei 0, daher ist die dritte Stelle dann die Nummer 2.

Variablen und Strings

erstes Vorkommen bestimmen über die String-Methode `.find()`

- Wollen wir das nächste Vorkommen von „e“ finden, können wir den Start mitgeben als weitere Parameter:

```
inhalt = "Hier kommt ein String-Inhalt"  
print ( inhalt.find("e", 3) )
```

- Als Rückmeldung erhalten wir:

```
11
```


Variablen und Strings

erstes Vorkommen bestimmen über die String-Methode `.find()`

- Bereiche können auch definiert werden. Wird die letzte Zahl negativ angegeben, erfolgt die Zählung von hinten:

```
inhalt = "Hier kommt ein String-Inhalt"  
print ( inhalt.find("i", 5, -10) )
```

Variablen und Strings

String auf Bedingung testen

- Es gibt verschiedene Abfragen um zu testen, ob ein bestimmtes Kriterium für einen String vorliegt. Diese String-Methoden starten immer mit is.....
- Im Folgenden die Übersicht aller Methoden des Datentyp Strings und is...:

Variablen und Strings

String auf Bedingung testen

Methode	Beschreibung
<code>string.isalnum()</code>	Überprüft auf alphanumerische Zeichen (a-zA-Z0-9). Leerzeichen ist kein alphanumerisches Zeichen!
<code>string.isalpha()</code>	Überprüft auf alphabetische Zeichen (a-zA-Z). Leerzeichen ist kein alphanumerisches Zeichen!
<code>string.isdecimal()</code>	Überprüft auf Zahlen – wenn alle Zeichen Dezimalzahlen sind, wird True zurückgeliefert
<code>string.isdigit()</code>	Überprüft auf Numerische und digitale Zeichen z.B. '123' oder '3\u00B2' (was 3 hoch 2 entspricht!)
<code>string.isidentifier()</code>	Ob ein Identifier vorliegt (siehe https://docs.python.org/3.3/reference/lexical_analysis.html#identifiers)
<code>string.islower()</code>	Überprüft, ob alles in Kleinschreibung vorliegt
<code>string.isnumeric()</code>	Überprüft auf numerische Zeichen (z.B. 1/2, 3hoch2 etc.)
<code>string.isprintable()</code>	Überprüft, ob Druckbar ist
<code>string.isspace()</code>	Überprüft, ob nur Leerzeichen (u.ä.) vorhanden sind
<code>string.istitle()</code>	Überprüft, ob es sich um eine Überschrift handelt
<code>string.isupper()</code>	Überprüft, ob alles in Großschreibung vorliegt

Im Folgenden werden zwei Methoden beispielhaft vorgestellt:

Variablen und Strings

String auf Bedingung testen

isalnum() Methode: Test auf alphanumerische Zeichen

- Liegen nur alphanumerische Zeichen vor? Wenn das zutrifft, wird „True“ als Rückgabewert zurückgegeben. Alphanumerische Zeichen sind die Buchstaben („Alpha“) des Alphabets und Zahlen („numerische“) von 0 bis 9 – daher Alphanumerisch.

```
inhalt = "Beispieltext"  
ergebnis = inhalt.isalnum()  
print(ergebnis)
```

- Ergebnis:

```
True
```

Variablen und Strings

String auf Bedingung testen

- Sobald Leerzeichen vorkommen, sind nicht mehr alle Zeichen alphanumerisch und somit kommt beim folgenden Beispiel False zurück:

```
inhalt = "Beispieltext mit Leerzeichen"  
ergebnis = inhalt.isalnum()  
print(ergebnis)
```

- Ergebnis:

```
False
```

Variablen und Strings

String auf Bedingung testen

isalpha() Methode: Test auf Buchstaben (Alphabet)

- Test, ob nur Buchstaben vorliegen, sprich das Alphabet verwendet wird.
- Zahlen und Leerzeichen führen zu einem „False“, da diese nicht dazu gehören!

- Ergebnis:

```
inhalt = "Beispieltext"  
ergebnis = inhalt.isalpha()  
print(ergebnis)
```

True

- Und hier mit Zahl und somit trifft es nicht zu:

```
inhalt = "123Beispieltext"  
ergebnis = inhalt.isalpha()  
print(ergebnis)
```

False

- Ergebnis:

Variablen und Strings

String aufteilen mit `split()`

- Oft liegen uns Daten vor, die durch Komma getrennt sind.
- Beispielsweise ein Export von Excel im Format CSV (englisch „comma separated values“).
- Diesen String können wir einfach „aufspalten“ über `split()`
- Die Methode `split(Trennzeichen, Anzahl_Aufteilungen_maximal)` hat 2 Parameter, die beide Optional sind.
- Schauen wir uns den ersten Parameter an.
- Über diesen geben wir das gewünschte Trennzeichen mit.

Variablen und Strings

String aufteilen mit `split()`

- Die Methode `split(Trennzeichen, Anzahl_Aufteilungen_maximal)` hat 2 Parameter, die beide Optional sind. Schauen wir uns den ersten Parameter an. Über diesen geben wir das gewünschte Trennzeichen mit.

```
daten = "vorname, nachname, alter"  
einzeldaten = daten.split(",")  
print(einzeldaten)
```

- Als Ergebnis erhalten wir eine Liste. Listen lernen wir im Kapitel Listen kennen.

```
['vorname', ' nachname', ' alter']
```


Variablen und Strings

String aufteilen mit `split()`

- Achtet man nun genau auf den zurückgelieferten Inhalt,
 - sieht man vor ' nachname' und ' alter' jeweils ein Leerzeichen.
- Diese Leerzeichen sind oft unerwünscht, können aber sehr einfach mit der Methode `strip()` entfernt werden.
- Oder man achtet bereits beim Ausgangsmaterial darauf, dass keine Leerzeichen nach den Kommas vorhanden sind.
- Wenn man allerdings sicher weiß, dass immer im Ausgangsmaterial nach dem Komma ein Leerzeichen kommt, kann man dies auch als Parameter nutzen!
- Der Parameter kann also aus einer beliebigen Zeichenkombination bestehen.

Variablen und Strings

String aufteilen mit `split()`

Wir übergeben der Methode bei unserem Beispiel neben dem Komma auch das Leerzeichen:

```
daten = "vorname, nachname, alter"  
einzeldata = daten.split(", ")  
print(einzeldata)
```

- Als Ausgabe erhalten wir:

```
['vorname', 'nachname', 'alter']
```

Variablen und Strings

String aufteilen mit `split()`

erste Parameter bei `split()`

- Bei der Methode `split()` sind zwei Parameter möglich und beide sind optional!
- Im letzten Beispiel haben wir als ersten Parameter das gewünschte Trennzeichen vorgegeben.
- Diese Angabe können wir auch weglassen.
- Schauen wir uns an, was passiert, wenn wir das letzte Beispiel ohne Parameter ausführen lassen.
- Wir ändern nichts am Beispiel außer bei `split()`

```
daten = "vorname, nachname, alter"  
einzeldaten = daten.split()  
print(einzeldaten)
```

- Als Ergebnis erhalten wir nun:

```
['vorname,', 'nachname,', 'alter']
```

Variablen und Strings

String aufteilen mit `split()`

- Als Ergebnis erhalten wir nun:

```
['vorname,', 'nachname,', 'alter']
```

- Wird also `split()` ohne Parameter aufgerufen, erfolgt eine Trennung bei jedem Leerzeichen! Jetzt werden die Kommas als Inhalt angesehen und sind bei der Liste in „vorname,“ und „nachname,“ gelandet.
- Interessant ist noch, dass mehrere Leerzeichen (falls vorhanden) als eines angesehen werden.
- Wir erhalten das gleiche Ergebnis wie oben bei folgenden String:

```
daten = "vorname,  nachname, alter"
```

Variablen und Strings

String aufteilen mit `split()`

zweiter Parameter: `Anzahl_Aufteilungen_maximal`

- Beim zweiten Parameter von `split(Trennzeichen, Anzahl_Aufteilungen_maximal)` können wir festlegen, wie viele Aufteilungen wir gerne maximal bekommen möchten.
- Wird nichts angegeben (was dem Standard von -1 entspricht) erhalten wir alle möglichen.
- Wären 2 möglich (wie bei unseren vorherigen Beispielen) und wir geben 1 an, erhalten wir auch nur noch eine Aufspaltung:

```
daten = "vorname,nachname,alter"  
einzeldata = daten.split(",", 1)  
print(einzeldata)
```

- Als Ergebnis bekommen wir auch genau eine Teilung:

```
['vorname', 'nachname,alter']
```

- Wir bekommen also als Anzahl von Listenelemente unsere Anzahl von Trennungen + 1.

Variablen und Strings

String auf Bedingung testen

Anzahl Wörter in einem Text über split()

- Über die Methode split() ist es sehr einfach, die Anzahl der Wörter in einem Text zu bestimmen.
- Wir wissen, dass Leerzeichen die Trennung zwischen Wörtern in einem Text darstellen.
- Also nutzen wir das Leerzeichen als Trennzeichen in split() und können danach über len() die Anzahl der Elemente (sprich Wörter) zählen.

```
inhalt = "Anzahl Wörter in einem Text zählen!"  
woerter = inhalt.split()  
print("Anzahl der Wörter: ", len(woerter))
```

Variablen und Strings

„Endet mit“-Methode: Strings auf Suffix überprüfen mit `.endswith()`

- Die Methode `.endswith()`
 - liefert „True“ zurück, wenn das gesuchte Ende vorhanden ist.
 - Ansonsten kommt als Rückgabewert „False“.
- Die englische Zusammenziehung liest sich ungewohnt.
- Übersetzt man es ein wenig „holprig“ ins deutsche, ist die Funktion der Methode klar: ends = wird beendet, endet / with = mit
- Also einfach die Methode „endet-mit“.

Variablen und Strings

„Endet mit“-Methode: Strings auf Suffix überprüfen mit `.endswith()`

Syntax von `endswith()`

- `string.endswith(Suffix[, Startposition[, Endposition]])`
- Wir haben also 3 Parameter:
 - Suffix (muss angegeben werden): zu überprüfende Zeichenfolge
 - Startposition (optional): Startpunkt, ab dem überprüft wird
 - Endposition (optional): Endpunkt, bis zu dem überprüft wird

Variablen und Strings

„Endet mit“-Methode: Strings auf Suffix überprüfen mit `.endswith()`

- Anhand eines Beispiels wird die Anwendung klarer.

```
inhalt = "https://cctrainer.ddnss.de"  
ergebnis = inhalt.endswith(".de")  
print(ergebnis)
```

- Wir bekommen als Ergebnis zurückgeliefert:

```
True
```

- Wir können also überprüfen, ob es zutrifft (hier in unserem Fall, ob es eine deutsche Domainendung ist).
- Damit können wir auch in eine if-Abfrage gehen oder in eine while-Schleife.

Variablen und Strings

„Endet mit“-Methode: Strings auf Suffix überprüfen mit `.endswith()`

Anfangs- und Endposition einsetzen

- Je nach Fall ist es manchmal geschickt, die Anfangs- und Endposition für die Überprüfung festzulegen.
- Geben wir hier 28 als Ende an, bekommen wir weiterhin „True“ zurück, da unser String 28 Zeichen lang ist.

```
inhalt = "https://cctrainer.ddnss.de"  
ergebnis = inhalt.endswith(".de", 0, 26)  
print(ergebnis)
```

- Sobald wir 25 eingeben haben wir zur Überprüfung nur noch „...cctrainer.ddnss.d“ – und somit kein „.de“ mehr.

Variablen und Strings

„Endet mit“-Methode: Strings auf Suffix überprüfen mit `.endswith()`

Mehrere Fälle überprüfen

- Diese Methode kann auch mit Tupel's eingesetzt werden. Hört sich kompliziert an, ist aber in der Praxis sehr einfach und wird öfters benötigt. Was ist ein Tupel? Ein Tupel ist eine Wertesammlung.
- Nehmen wir an, wir wollen unseren String überprüfen, ob die URL mit einer dieser Endungen endet:
 - `.de`
 - `.com`
 - `.net`

Variablen und Strings

„Endet mit“-Methode: Strings auf Suffix überprüfen mit `.endswith()`

Also nehmen wir alle 3 Fälle (unsere Wertesammlung) auf und es soll, wenn einer der Fälle zutrifft „True“ zurückgeliefert werden.

- Wir erstellen aus unseren Fällen ein Tupel (siehe entsprechendes Kapitel <https://www.python-lernen.de/tupel.htm>)
- Unser Tupel aus den 3 Domainendungen:

```
datentyp_tupel = (".de", ".com", ".net")
```

- Dieses Tupel wird nun in unsere Methode `.endswith()` eingesetzt:

```
inhalt = "https://cctrainer.ddnss.de"  
datentyp_tupel = (".de", ".com", ".net")  
ergebnis = inhalt.endswith(datentyp_tupel)  
print(ergebnis)
```

- Jetzt kann die URL auf „.de“ oder auf „.com“ oder auf „.net“ enden und wir erhalten ein „True“ zurück.

Variablen und Strings

„Endet mit“-Methode: Strings auf Suffix überprüfen mit `.endswith()`

- Jetzt kann die URL auf „.de“ oder auf „.com“ oder auf „.net“ enden und wir erhalten ein „True“ zurück.
- Öfters wird man diese Konstruktion sehen. Hier ist schneller ersichtlich, dass ein Tupel eingesetzt wird:

```
inhalt = "https://cctrainer.ddnss.de"  
ergebnis = inhalt.endswith((".de", ".com", ".net"))  
print(ergebnis)
```

Variablen und Strings

„Beginnt mit“-Methode: Stringanfang überprüfen mit `.startswith()`

Die Methode `.startswith()`

- liefert „True“ zurück, wenn das Gesuchte am Anfang des Strings vorhanden ist.
- Ansonsten kommt als Rückgabewert „False“.
- Die englische Zusammenziehung liest sich ungewohnt. Übersetzt man es ein wenig „holprig“ ins deutsche, ist die Funktion der Methode klar:
 - starts = startet, beginnt
 - with = mit
- Also einfach die Methode „Beginnt-mit“.
- Wir können also Überprüfen, ob bei einem String der Anfang einer von uns bestimmten Zeichenfolge entspricht.
- Nehmen wir an, wir haben als String als Text eine URL und wollen überprüfen, ob diese mit „https://“ startet.

Variablen und Strings

„Beginnt mit“-Methode: Stringanfang überprüfen mit `.startswith()`

Nehmen wir an, wir haben als String als Text eine URL und wollen überprüfen, ob diese mit „https://“ startet.

```
inhalt = "https:// cctrainer.ddnss.de "  
ergebnis = inhalt.startswith("https://")  
print(ergebnis)
```

Als Rückmeldung erhalten wir „True“, da die Überprüfung korrekt war. Würde die URL allerdings mit „http://“ starten, kommt ein „False“ zurück:

```
inhalt = "http:// cctrainer.ddnss.de "  
ergebnis = inhalt.startswith("https://")  
print(ergebnis)
```

Variablen und Strings

„Beginnt mit“-Methode: Stringanfang überprüfen mit `.startswith()`

- Wie schon bei der Methode „`.endswith()`“ gibt es weitere Parameter.
 - Suffix (muss angegeben werden): zu überprüfende Zeichenfolge
 - Startposition (optional): Startpunkt, ab dem überprüft wird
 - Endposition (optional): Endpunkt, bis zu dem überprüft wird
- Wir können also auch festlegen, ab welcher Position angefangen wird, die Überprüfung zu starten und bis zu welcher Position.

Variablen und Strings

„Beginnt mit“-Methode: Stringanfang überprüfen mit `.startswith()`

Mehrere Fälle überprüfen mit `.startswith()`

- Im obigen Beispiel haben wir schon gesehen, dass es mehrere Fälle geben kann.
- In unserem Beispiel kann eine Internetadresse mit „https://“ oder mit „http://“ beginnen.
- Klar wäre es jetzt einfach nur auf „http“ zu überprüfen.
- Wenn wir allerdings sicherstellen wollen, dass auch der „://“ für eine korrekte URL vorhanden ist, dann haben wir diese beiden Fälle:
 - `https://`
 - `http://`
- Und diese beide Fälle können wir als Tupel (siehe entsprechendes Kapitel) schreiben:

```
urlanfang_als_tupel = ("https://", "http://")
```

Variablen und Strings

„Beginnt mit“-Methode: Stringanfang überprüfen mit `.startswith()`

- Und genau dieses Tupel können wir als Kontrollwert unserer Methode `.startswith()` übergeben und erhalten ein „True“ als Rückgabewert, wenn einer der Werte des Tupels zutrifft.

```
inhalt = "https:// cctrainer.ddnss.de "  
urlanfang_als_tupel = ("https://", "http://")  
ergebnis = inhalt.startswith(urlanfang_als_tupel)  
print(ergebnis)
```

- Öfters ist auch die Methode in der folgenden Schreibweise anzutreffen (auf die 2 runden Klammern achten!):

```
inhalt = "https:// cctrainer.ddnss.de "  
ergebnis = inhalt.startswith(("https://", "http://"))  
print(ergebnis)
```

Variablen und Strings

Python String Methode .expandtabs() zum Umwandeln von Tabs in

Leerzeichen

Über die Methode `expandtabs()` werden alle in einem Strings enthaltene Tabs (`\t`) in Leerzeichen umgewandelt. Es werden 8 Leerzeichen für einen Tab genutzt, sofern man keine andere Angabe als Parameter mitgibt:

```
string.expandtabs([Anzahl_Leerzeichen])
```

- Schauen wir es uns als Beispielcode an:

```
inhalt = "Textinhalt\t1234567890\tmehr Inhalt"  
ergebnis = inhalt.expandtabs()  
print(ergebnis)
```

- Als Ergebnis erhalten wir folgende Ausgabe:

```
Textinhalt    1234567890    mehr Inhalt
```

Variablen und Strings

Python String Methode `.expandtabs()` zum Umwandeln von Tabs in Leerzeichen

- Parameter bei `expandtabs()`
 - Über den Parameter kann man die gewünschte Anzahl an Tabs angeben, wenn man eine andere Anzahl als 8 (was der Standardeinstellung entspricht), gerne hätte.
 - Beispiele:

Variablen und Strings

Python String Methode .expandtabs() zum Umwandeln von Tabs in Leerzeichen

```
inhalt = "Textinhalt\t1234567890\tmehr Inhalt"
print("01234567890123456789012345678901234567890123456789")
print(inhalt.expandtabs(), " (Standardeinstellung 8)\n")
print(inhalt.expandtabs(2), " (Tabstopp bei 2)\n")
print(inhalt.expandtabs(3), " (Tabstopp bei 3)\n")
print(inhalt.expandtabs(4), " (Tabstopp bei 4)\n")
print(inhalt.expandtabs(5), " (Tabstopp bei 5)\n")
print(inhalt.expandtabs(6), " (Tabstopp bei 6)\n")
print(inhalt.expandtabs(7), " (Tabstopp bei 7)\n")
print(inhalt.expandtabs(8), " (Tabstopp bei 8)\n")
print(inhalt.expandtabs(9), " (Tabstopp bei 9)\n")
print(inhalt.expandtabs(10), " (Tabstopp
```

Variablen und Strings

Python String Methode .expandtabs() zum Umwandeln von Tabs in Leerzeichen

- Und als Ergebnis erhalten wir:

```
01234567890123456789012345678901234567890123456789
Textinhalt  1234567890  mehr Inhalt (Standardeinstellung 8)
Textinhalt 1234567890  mehr Inhalt (Tabstopp bei 2)
Textinhalt 1234567890  mehr Inhalt (Tabstopp bei 3)
Textinhalt 1234567890  mehr Inhalt (Tabstopp bei 4)
Textinhalt  1234567890  mehr Inhalt (Tabstopp bei 5)
Textinhalt 1234567890  mehr Inhalt (Tabstopp bei 6)
Textinhalt  1234567890  mehr Inhalt (Tabstopp bei 7)
Textinhalt  1234567890  mehr Inhalt (Tabstopp bei 8)
Textinhalt   1234567890  mehr Inhalt (Tabstopp bei 9)
Textinhalt    1234567890  mehr Inhalt (Tabstopp bei 10)
```

- Auch wenn es so wirkt, als würde sich bei der Einstellung für den Tabstopp für 2, 3 wie auch bei 4 nichts ändern, passt das Verhalten durchaus.
- Im Beispiel liegt bei allen 3 Einstellungen der Beginn nach dem ersten umgewandelten Tab bei 12. Und 12 ist ein Vielfaches von 2, 3 und 4.

Variablen und Strings

Python String Methode .expandtabs() zum Umwandeln von Tabs in Leerzeichen

- Bis zur Position 10 ist alles belegt durch den vorherigen Text plus eines Leerzeichens. Also kann erst nach 10 der Tab „wirken“. Schauen wir uns die Reihen an:
 - Bei der 2er-Reihe:
 - 2,4,6,8,10 (alles nicht möglich), 12 (und dort startet dann auch unser Text)
 - Bei der 3er-Reihe:
 - 3,6,9 (alles nicht möglich), 12 (und dort startet dann auch unser Text)
 - Bei der 4er-Reihe:
 - 4,8 (alles nicht möglich), 12 (und dort startet dann auch unser Text)
 - Bei der 6er-Reihe:
 - 6 (nicht möglich), 12 (und dort startet dann auch unser Text)
 - Die 5er-Reihe ist anders,
 - da hier unser Text bei 15 startet 5,10 (alles nicht möglich), 15 (startet unser Text)

Variablen und Strings

String Methode partition() – String in Einzelteile zerlegen

- Die Methode partition() erhält als Parameter den Suchtext, anhand der komplette String zerlegt werden soll.

```
string.partition("Suchtext")
```

- Wir bekommen 3 Teile als Rückgabewerte:
 - alles vor dem Suchtext
 - den Suchtext
 - alles nach dem Suchtext

Variablen und Strings

String Methode partition() – String in Einzelteile zerlegen

- Schauen wir es am Beispiel an. Wir haben den Satz „Python ist einfach zu lernen“. Jetzt wollen wir diesen Text zerlegen, und zwar bei dem Wort „ist“.

```
satz = "Python ist einfach zu lernen"  
ergebnis = satz.partition("ist")  
print(ergebnis)
```

- Das Ergebnis ist als Datenform ein Tupel und wir erhalten:

```
('Python ', 'ist', ' einfach zu lernen')
```

- Bitte ein Augenmerk darauf, dass nichts verloren geht!
- Es werden auch alle Leerzeichen um der Suchwert „ist“ behalten.
- Dieser finden sich nach dem ersten Wert des Tupels nach „Python “ und vor dem dritten Wert „ einfach zu lernen“.

Variablen und Strings

String Methode `partition()` – String in Einzelteile zerlegen

Suchtext öfters vorhanden – was passiert?

- Was passiert nun eigentlich, wenn unser Suchtext öfters vorhanden ist? Erweitern wir unseren Satz auf: „Python ist einfach zu lernen und ist cool“.

```
satz = "Python ist einfach zu lernen und ist cool"  
ergebnis = satz.partition("ist")  
print(ergebnis)
```

- Im Ergebnis sieht man schön, dass nur das erste Auftreten des Suchtextes berücksichtigt wird. Das zweite „ist“ in unserem Beispiel endet im dritten Rückgabewert des Tupels.

```
('Python ', 'ist', ' einfach zu lernen und ist cool')
```

Variablen und Strings

String Methode partition() – String in Einzelteile zerlegen

Suchtext am Anfang sofort vorhanden

- Was passiert, wenn der Suchtext sofort am Anfang unseres Strings vorhanden ist?

```
satz = "ist Python einfach zu lernen?"  
ergebnis = satz.partition("ist")  
print(ergebnis)
```

- Hier wird das Ergebnis für das Verständnis der Funktion von .partition() wichtig.
 - Als Rückgabe-Tupel erhalten wir:

```
("", 'ist', ' Python einfach zu lernen?')
```

- Unser Suchtext ist in dem Tupel also immer der zweite Wert! Tritt er im Text in der ersten Position auf, kommt vor ihm also nichts.
 - Somit bleibt der erste Wert unseres Rückgabetupels leer und im dritten Wert steht alles nach dem „ist“.

Variablen und Strings

String Methode partition() – String in Einzelteile zerlegen

Suchtext nicht vorhanden

- Wenn unser Suchtext nicht im Text vorkommt, ist die Reaktion der partition()-Methode, dass im Ergebnis alles im ersten Bereichs unseres Tupels landet und der zweite Wert des Tupels leer bleibt. Unsere Suche war ja nicht erfolgreich und im zweiten Wert steht immer das erfolgreich gesuchte Wort:

```
satz = "Ist Python einfach zu lernen?"  
ergebnis = satz.partition("ist")  
print(ergebnis)
```

- Ergebnis:

```
('Ist Python einfach zu lernen?', '', '')
```

- Man sieht an diesem Beispiel auch, dass Groß- und Kleinschreibung einen Unterschied macht.

Variablen und Strings

String Methode `partition()` – String in Einzelteile zerlegen

Fehlermeldung „`ValueError: empty separator`“

- Die Methode `partition()` benötigt einen Parameter (für den Suchtext), ansonsten erhält man als Fehlermeldung
 - „`ValueError: empty separator`“.
 - Das passiert, wenn ein leerer Suchtext übergeben wird:
 - `ergebnis = satz.partition("")`.
- Gibt man anstelle des Suchtextes keinen Parameter ein
 - `ergebnis = satz.partition()`,
 - erhält man die Fehlermeldung: „`TypeError: partition() takes exactly one argument (0 given)`“

Variablen und Strings

String Methode partition() – String in Einzelteile zerlegen

Für das Aufspalten muss also ein sinnvoller Wert mitgegeben werden. Wobei ein Leerzeichen auch ein sinnvoller Wert ist:

```
satz = "Ist Python einfach zu lernen?"  
ergebnis = satz.partition(" ")  
print(ergebnis)
```

- Er wird dann einfach beim ersten Leerzeichen aufgeteilt. Unser Ergebnis:

```
('Ist', ' ', 'Python einfach zu lernen?')
```

- Die Methode partition() bietet viele Möglichkeiten im praktischen Einsatz. Einfach im Hinterkopf behalten, dass es diese Methode gibt.

Variablen und Strings

Zusammenfügen von Zeichenketten über join()

- Die Methode join() ist extrem hilfreich, um Zeichenketten zusammenzufügen.
- Wir erhalten als Rückgabe ein String.
- Was sich so beiläufig anhört, ist extrem wichtig.
- Denn wir können join() mit verschiedenen Datentypen „füttern“ und bekommen eine Zeichenkette zurück!

Variablen und Strings

Zusammenfügen von Zeichenketten über join()

Befehlsaufbau:

```
str = trennzeichen.join(Aufzählung)
```

- Beispiel: Einsatz von join() mit dem Datentyp Liste

```
wortliste = ['Axel', 'Elke', 'Martin']  
trennzeichen = '#'  
ergebnis = trennzeichen.join(wortliste)  
print(ergebnis)
```

- Als Ergebnis erhalten wir zurück:

```
Axel#Elke#Martin
```

- Wir erhalten aus dem Datentyp Liste eine String. Die einzelnen Elemente sind durch „#“ getrennt.

Variablen und Strings

Zusammenfügen von Zeichenketten über join()

- Die Nutzung von join() ist anhand von dem Datentyp Listen einfacher verständlich und die Mächtigkeit der Methode schnell klar.
- Es funktioniert genauso mit Zeichenketten („Strings“).
- Allerdings wird jedes Zeichen des Textes getrennt durch das Trennzeichen vom nächsten Zeichen:

```
zeichenkette = "abcd"  
trennzeichen = '#'  
ergebnis = trennzeichen.join(zeichenkette)  
print(ergebnis)
```

- Als Ergebnis erhalten wir:

```
a#b#c#d
```

- Wir können die Datentyp „List“, „Tupel“, „String“, „Dictionary“ und „Set“ nutzen und join() übergeben.

Variablen und Strings

Zusammenfügen von Zeichenketten über join()

beliebige Anzahl von Trennzeichen

- Dabei kann auch mehr als 1 Trennzeichen angegeben werden.
- Die Nutzung von trennzeichen = ' #123# ' führt beispielsweise zu „Axel #123# Elke #123# Martin“.

```
wortliste = ['Axel', 'Elke', 'Martin']  
trennzeichen = ' #123# '  
ergebnis = trennzeichen.join(wortliste)  
print(ergebnis)
```

- Und als Ergebnis erhalten wir:

```
Axel #123# Elke #123# Martin
```

Variablen und Strings

Zusammenfügen von Zeichenketten über join()

Datentyp Dictionary und Fallstricke bei join()

- Der Datentyp Dictionary (auf deutsch „Wörterbuch“ bzw. assoziative Liste) kann als Inhalte sowohl Strings wie Werte haben.
- Besteht das Wörterbuch nur aus Strings, haben wir kein Problem. Wichtig ist nur zu wissen, dass immer bei dem Datentyp Dictionary der „key“ verwendet wird.

```
deutschenglisch = { 'null': 'zero', 'eins': 'one' }  
trennzeichen = '#'  
print(trennzeichen.join(deutschenglisch))
```

- Und als Ergebnis erhalten wir:

```
null#eins
```

- Ist der Inhalt allerdings numerisch, bekommen wir eine Fehlermeldung!

```
woerterbuch = {0: 'null', 1: 'eins' }  
trennzeichen = '#'  
print(trennzeichen.join(woerterbuch))
```

- Anstelle eines Ergebnisses erhalten wir die Fehlermeldung: „TypeError: sequence item 0: expected str instance, int found“

Variablen und Strings

Operatoren für Strings

- Lustigerweise kann man in Python auch mit Operatoren (+-*/) auf Zeichenkettenausgaben losgehen.
- Was passiert, wenn man folgende Anweisung schreibt?
- Nun erfolgt als Ausgabe

```
print( 3 * 'mi' );
```

- Ein Operator ist eine mathematische Vorschrift.
- So steht das „*“ wie üblich in der Mathematik für die Multiplikation
 - – wenden wir diese Multiplikation in Python auf einen Text an, wird dieser entsprechend oft wiederholt.

```
mimimi
```

Variablen und Strings

Operatoren für Strings

Jetzt können wir auch noch dahinter ein Plus packen:

```
print( 3 * 'mi' + 'mo' );
```

- Nun kommt als Ausgabe

```
mimimimo
```

Variablen und Strings

Operatoren für Strings

Gibt man 2 Strings hintereinander an:

```
print( 'mi' 'mo' );
```

- Werden bei zusammen hintereinander ausgegeben.
- Im Vergleich zu anderen Programmiersprachen benötigen wir kein zusätzliches Zeichen, um mehrere Zeichenketten miteinander zu verketten.
- In JavaScript würde man mit einem Pluszeichen arbeiten, in PHP mit einem Punkt verketten. Python macht es einfacher.

Variablen und Strings

Operatoren für Strings

Allerdings funktioniert das bei 2 Variablen nicht und es gibt eine Fehlermeldung.

- Versucht man bei Variablen folgende Programm:
- Es kommt nun die Fehlermeldung "SyntaxError: invalid syntax".

```
variable1 = "ich"  
variable2 = "du"  
print( variable1 variable2 )
```

Variablen und Strings

Operatoren für Strings

- Es klappt dann wieder, wenn wir ein Pluszeichen mitgeben. Daher bietet es sich an, das Pluszeichen immer zu machen (auch bei Zeichenketten, wo es eigentlich ohne gehen würde).

```
variable1 = "ich"  
variable2 = "du"  
print( variable1 + variable2 )
```


Variablen und Strings

Operatoren für Strings

Beispielanwendung: Funktionsgrafen ausgeben ohne Grafik

- Natürlich kann man sich fragen, wofür man die Wiederholung von Ausgaben benutzen könnte.
 - Eine einfache (und manchmal ausreichende) Variante ist die Ausgabe einer Kurve.
- Im Folgendem ein kleines Beispiel: Nicht besonders schön programmiert, aber es funktioniert.
 - Schöner geht es dann, wenn wir im Kurs Schleifen kennengelernt haben.

Variablen und Strings

Operatoren für Strings

```
print( 'Ausgabe Kurve ohne Grafik' );  
print( 3 * '*' );  
print( 5 * '*' );  
print( 8 * '*' );  
print( 9 * '*' );  
print( 10 * '*' );  
print( 9 * '*' );  
print( 8 * '*' );  
print( 5 * '*' );  
print( 3 * '*' );
```

Ausgabe Kurve ohne Grafik

```
***  
  
*****  
  
*****  
  
*****  
  
*****  
  
*****  
  
*****  
  
***
```