

Chatbot

Autor:	Basile Alessandro
Erstell Datum:	10. Februar 2020
Lehrjahr	1.Lehrjahr
Semester:	2.Semester
Berufsbildner	Herr Fructuoso
Firma	Syntegon

Inhaltsverzeichnis

1	Was ist ein Chatbot	6
1.1	Regelbasierter Chatbot	6
1.2	Selbstlernender Chatbot	6
1.2.1	Suchbots	6
1.2.2	generative Bots.....	6
2	Python vs. JavaScript	7
2.1	Python	7
2.2	JavaScript.....	7
2.3	Fazit	7
3	Python	8
3.1	Was ist Python?	8
3.2	Windows Installationen.....	8
3.2.1	Anaconda.....	8
3.2.2	Spyder.....	9
3.2.3	PyCharm	10
3.2.3.1	In PyCharm schreiben.....	12
3.3	Grund-Befehle	16
3.4	Variablen	16
3.4.1	Statische Variable	17
3.5	Datentypen.....	17
3.6	Listen	18
3.7	Listen Teil 2.....	19
3.7.1	Eintrag Löschen	19
3.7.2	List slicing.....	20
3.7.3	List Comprehension.....	20
3.7.4	Tuple	20
3.7.5	Dictionaries.....	21
	items.....	22
3.7.6	Listen Verschachteln	22
3.7.7	Liste in Dictionarie	22
3.8	Boolesche Werte	22
3.9	Kontrollstruktur	23
3.9.1	if, else	23
3.9.2	Vergleichsoperatoren.....	24
3.9.3	Booleans (and und or)	24
3.9.4	not-Operator	26
3.9.5	Elif.....	27

3.9.6	While – Schleife	27
3.9.7	For – Schleife	27
3.9.8	Welche Schleife?	27
3.9.9	Break & Continue	28
3.10	Funktionen.....	28
3.10.1	Warum Funktionen?.....	28
3.10.2	Funktionen.....	28
3.10.3	Eigene Funktionen.....	28
3.10.4	return.....	29
3.11	Dateien Öffnen, Lesen, Schreiben.....	29
3.11.1	Datei lesen.....	29
3.11.2	\n Befehl	29
3.11.3	Datei schreiben.....	29
3.11.4	close()	29
3.11.5	write()	30
3.11.6	Daten in Dateien hinzufügen.....	30
3.11.7	With Konstrukt	30
3.11.8	CSV Dateien öffnen.....	30
3.12	Grafiken	30
3.12.1	Grafik zeichnen.....	30
3.13	Objektorientierung.....	32
3.13.1	Klassen und Methoden.....	32
3.13.2	Constructor und Methoden erstellen	32
3.13.3	Methoden und Eigenschaften	33
3.13.4	Private Eigenschaften und Methoden.....	33
3.13.5	Besondere Methoden.....	33
3.13.6	Vererbung.....	34
3.13.7	Type feststellen	34
3.13.8	Alles ein Objekt.....	35
3.13.9	Wie Klassen/Variablen benennen?	35
3.13.9.1	Wann wird was verwendet?.....	35
3.14	Module in Python	35
3.14.1	Das einbinden.....	36
3.14.2	Module und Ordner.....	36
3.14.3	Python Module.....	37
3.15	Exceptions	37
3.15.1	Fehler Lösung mit Exceptions.....	37
3.15.2	Eigene Fehlerklassen	37

4	Der Crawler.....	38
4.1	Tools	38
4.1.1	Beispiele	38
4.2	Requests	38
4.3	Beautifulsoup	38
4.4	Elemente Finden.....	39
4.5	Generatoren	39
5	Flask.....	41
5.1	Installation Flask.....	41
5.1.1	Über Anaconda.....	41
5.1.2	Über CMD	42
5.2	Neues Projekt	42
5.3	Grundlagen	42
5.4	Webseite aufrufen.....	43
5.5	HTML-Code mit Flask generieren	45
5.5.1	Unterseiten.....	45
5.5.2	HTML im return	46
5.5.3	Wichtige Unterordner	46
5.6	Flask richtig einbinden.....	46
5.6.1	Wichtige Dateien und Inhalte.....	46
5.6.2	Start Seite/Chatbot Seite mit Nav-Bar	47
5.6.3	Inhalt der Start und Chatbot Seite	Fehler! Textmarke nicht definiert.
5.6.3.1	start.html.....	Fehler! Textmarke nicht definiert.
6	Chatbot erstellen Python	51
6.1	Natürliche Sprachverarbeitung(NLP)	51
6.2	Natural Language Toolkit(NLTK).....	51
6.3	Installation.....	51
6.4	NLTK-Textvorverarbeitung	51
6.5	Word-Set	51
6.6	TF-IDF-Verfahren	51
6.7	Otiar-Koeffizient.....	52
6.8	Chatbot-Training.....	52
6.9	Daten Lesen.....	52
6.10	Quellvorverarbeitung	53
6.11	Keyword-Auswahl.....	53
6.12	Antwortgenerierung.....	53
6.13	Erste Fragen einbinden (ohne KI)	55
6.13.1	Lehrstellen Fragen antworten	55

6.14	Wieso andere Sprache einbinden	56
7	Testfälle	57
8	Arbeitsjournal.....	59
9	Tabelle	73
10	Abbildungsverzeichnis.....	74
11	Index.....	75
12	Glossar	77

1 Was ist ein Chatbot?

Ein Chatbot soll die gewünschten Fragen der Benutzer beantworten können und einen bestmöglichen Support darstellen. Dabei gibt es grundlegend zwei verschiedene Arten von Chatbots, und zwar den Regelbasierten Chatbot und den Selbstlernenden Chatbot.

1.1 Regelbasierter Chatbot

Diese Chatbots haben eine Liste mit Befehlen und Schlüssel Wörtern, um die Fragen der Benutzer zu beantworten, ist die Frage des Benutzers jedoch nicht vorprogrammiert, kann der Chatbot nicht mehr helfen.

1.2 Selbstlernender Chatbot

Diese Chatbots funktionieren nach der Logik der Künstlichen Intelligenz. Sie haben somit keine vorbereiteten Antworten. Dabei kann man wieder zwischen zwei Arten unterscheiden, und zwar den Suchbots und den generativen Bots.

1.2.1 Suchbots

Diese Bots verwenden Methoden, um eine Antwort aus einer Bibliothek von vordefinierten Kopien auszuwählen. Dabei verwenden sie den Text der Frage und den Kontext des Dialogs und wählen dann eine der vorgefertigten Antworten aus.

1.2.2 generative Bots

Diese Bots generieren ihre eigenen Antworten. Sie sind enorm intelligent da sie jedes Wort der Frage studieren und daraus eine Antwort abgeben.

2 Python vs. JavaScript

Grundlegend sind beide schwere Sprachen und sie haben beide ihre Vorteile und Nachteile.

2.1 Python

Vorteile	Nachteil
+Ist schnell zu lernen	-Wird schwer auf Website einzubinden
+ Intelligente Chatbots	-Braucht viel Geduld und noch mehr Zeit

Tabelle 1 - Vorteile und Nachteile Python

2.2 JavaScript

Vorteile	Nachteil
+Ist schnell zu lernen	-Ist nicht intelligent, kann nur vorbestimmte Antworten geben
+Schnelle Einbindung in Website	-Ist nur in bestimmten Gebieten nützlich

Tabelle 2 - Vorteile und Nachteile JavaScript

2.3 Fazit

Ich werde mich zuerst am Python Chatbot versuchen da es mein Ziel, war und ist einen intelligenten Chatbot zu entwickeln, jedoch werde ich sobald ich fertig bin mich dran machen ihn auf einer Website einzubinden. Ich werde auch in dieser Dokumentation noch erklären wie der Chatbot in JavaScript programmiert wird.

3 Python

3.1 Was ist Python?

Python ist eine Programmiersprache, welche zu Anfang nicht sehr einfach wirkt aber mit der Zeit immer logischer und einfacher wird. Ich werde auf die Sprache selber noch in Späteren Kapiteln darauf eingehen.

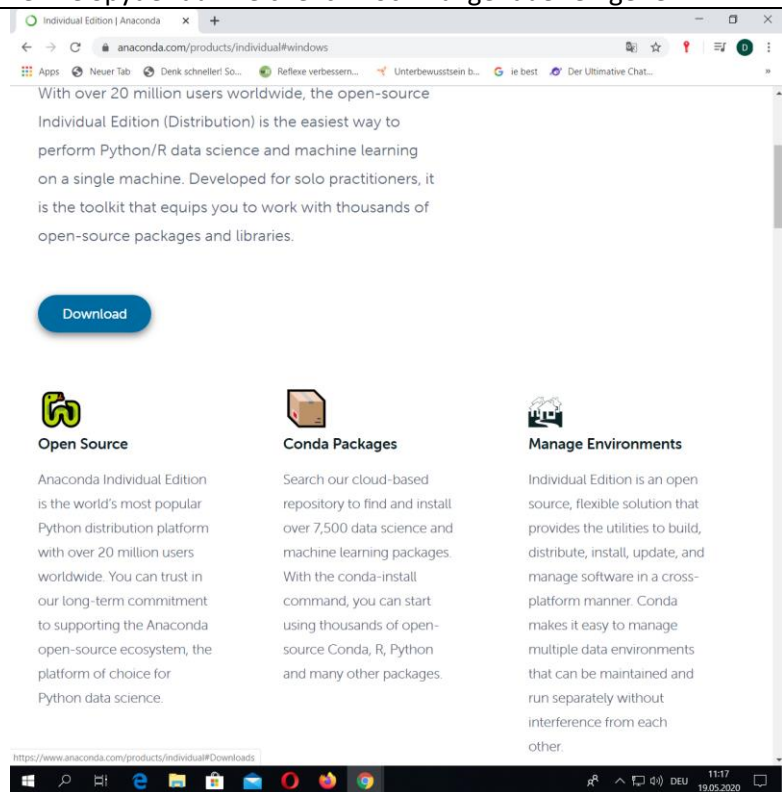
3.2 Windows Installationen

3.2.1 Anaconda

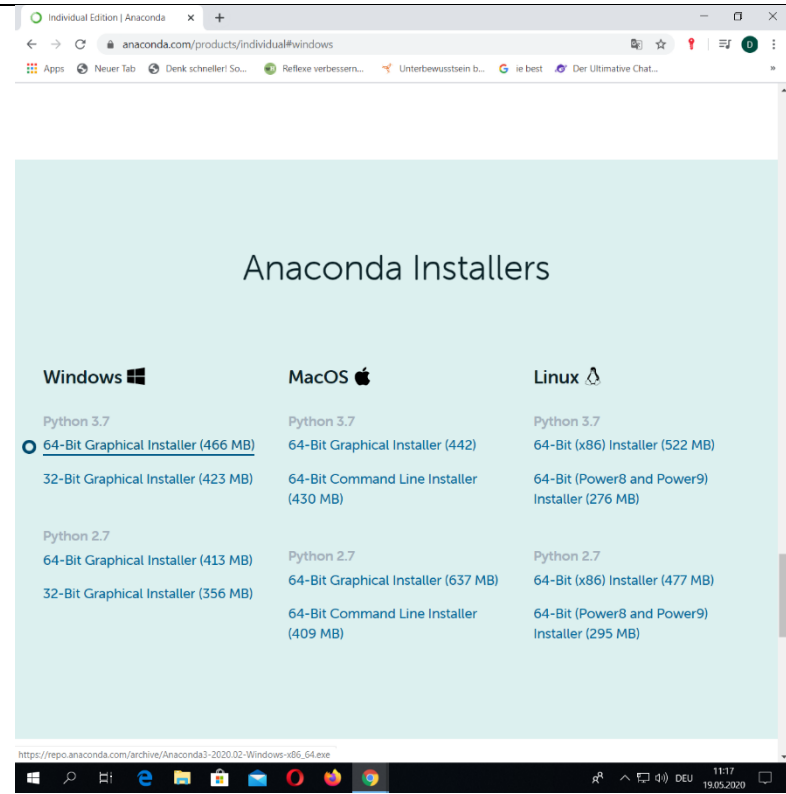
Zuerst was ist Anaconda. Anaconda ist ein Programm, in welchem es verschiedene Editoren und Hilfen. Darunter sind auch Programme wie Spyder auf welche ich nochmal genauer eingehe.

Als erstes geht man auf den folgenden Link und drückt dort auf den Download Button.

[Anaconda](https://www.anaconda.com/products/individual#windows)



Jetzt wählt man sein jeweiliges Betriebssystem, aus. In meinen Fall ist es *64-Bit Graphical Installer*. Nach dem Download führt man die jeweilige Datei aus



Jetzt klickt man sich durch die Installation durch, darunter akzeptiert man die Lizenz Bedingungen, den Installations Ordner und am Ende klickt man auf *Install*.

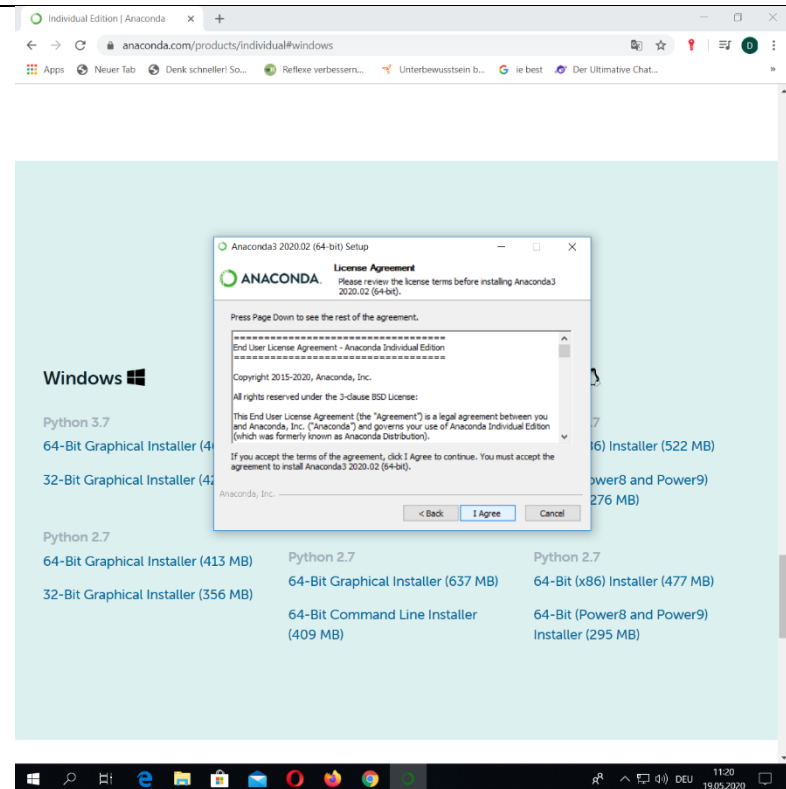


Tabelle 3 - Anaconda Installation

3.2.2 Spyder

Spyder ist ein Editor mit welchem man ganz einfach und mit Hilfen Python Code schreiben kann. Dieser wird super einfach, über das Anaconda Programm gedownloadet.

Spyder ist super einfach zu installieren. Dazu muss man nur den Anaconda Navigator öffnen. Diesen findet man ganz einfach, wenn man auf das Suchfeld von Windows klickt und dann Anaconda Navigator sucht.

Nach der Installation kann man ganz einfach auf Launch drücken und schon öffnet sich Spyder. In Spyder kann man dann direkt beginnen zu schreiben.

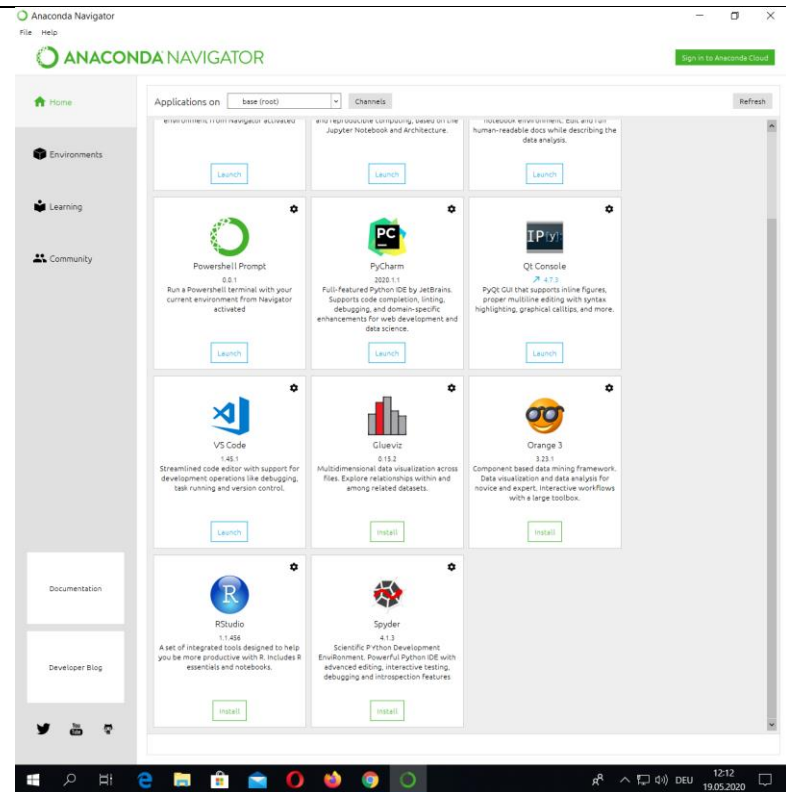


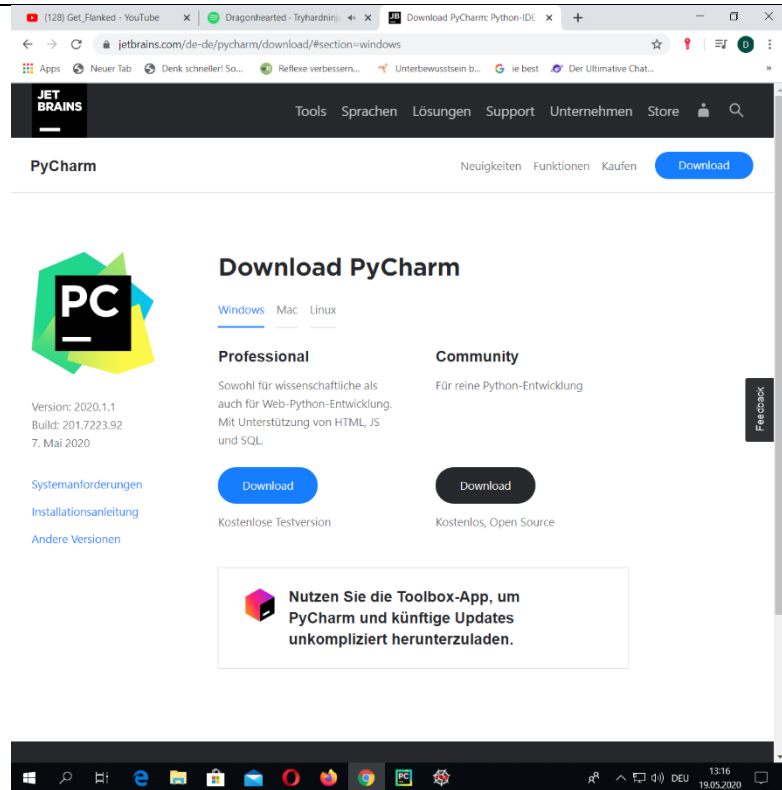
Tabelle 4 - Spyder Installation

3.2.3 PyCharm

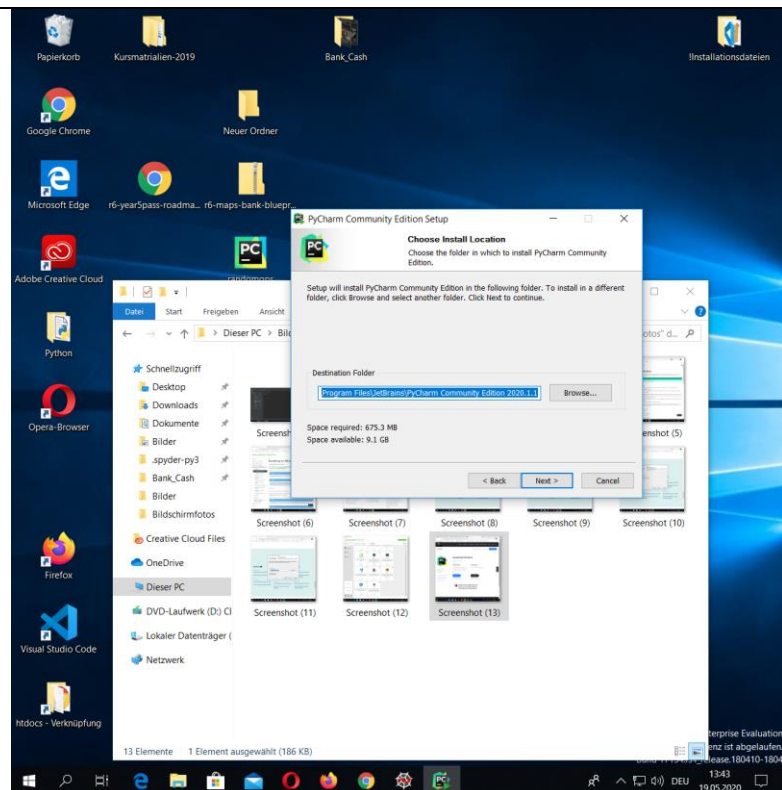
PyCharm ist auch ein Editor wie Spyder, jedoch finde ich ihn ein wenig besser, da er vielseitiger und effizienter ist als Spyder. Er gibt einem wie Spyder Vorschläge was man schreiben will. Und gibt auch meistens auch sehr gute Informationen was die Fehlerbehebung angeht. Aber was ihn vor allem für mich sehr gut macht ist das er Dateien verschiedener Programmiersprachen einfach erstellen kann. Dazu zählen auch HTML oder CSS, was für spätere Module wie Flask sehr wichtig sein kann.

Zuvor wichtig zu wissen ist, dass man PyCharm auch ganz einfach über den Anaconda Navigator herunterladen kann, wie Spyder. Jedoch kann es sein, dass Anaconda nicht die aktuellste Version gerade zur Verfügung stellt. Deshalb würde ich was das angeht eher denn Internet Download bevorzugen.

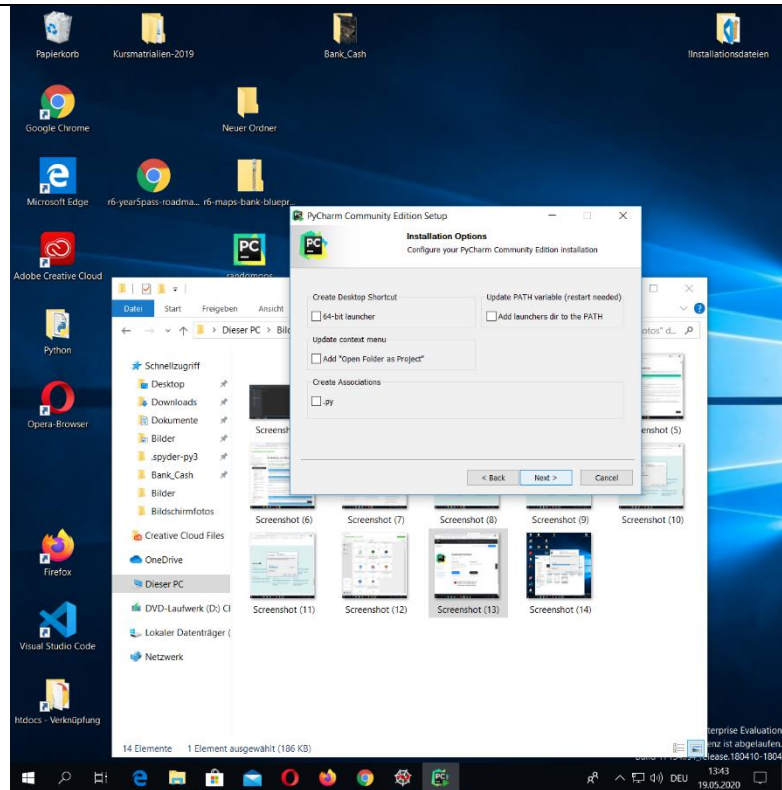
Als erstes geht man auf den folgenden Link und klickt dort auf Download und zwar die Community Version. Danach sollte es automatisch bereits die .exe Datei herunterladen.



Als nächstes führt man diese .exe Datei aus. Jetzt klickt man auf *Next* und wählt dann den Installationsordner aus.



Als nächstes wählt die jeweiligen Kästchen aus welche man will, für das Projekt oder die gesamte Dokumentation muss man keines der Kästchen anwählen.



Beim nächsten wählt man *JetBrains* aus. Dieses sollte aber eigentlich in den meisten Fällen bereits ausgewählt sein. Danach klickt man weiter und PyCharm wird bereits installiert.

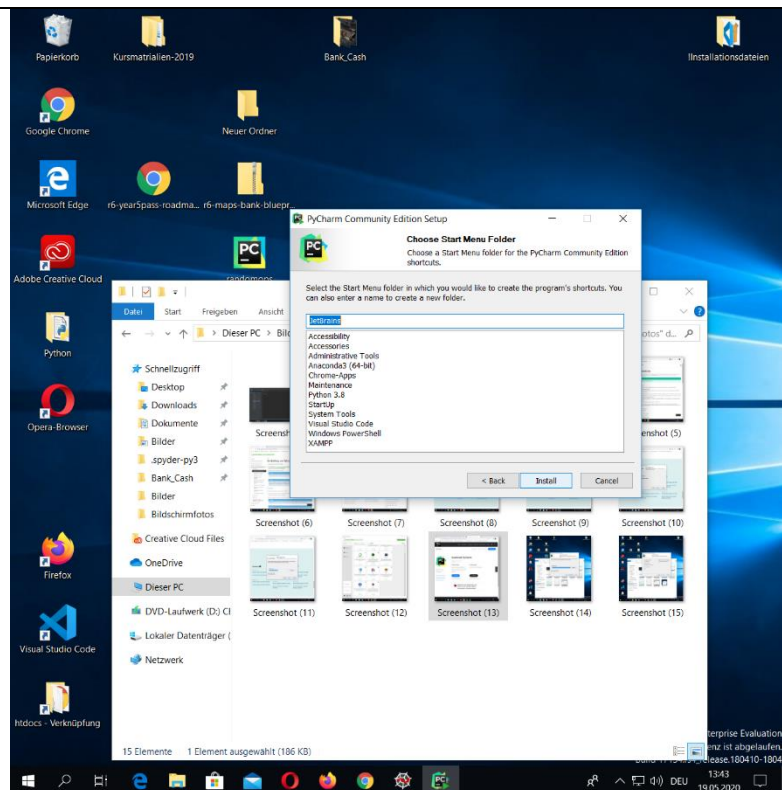
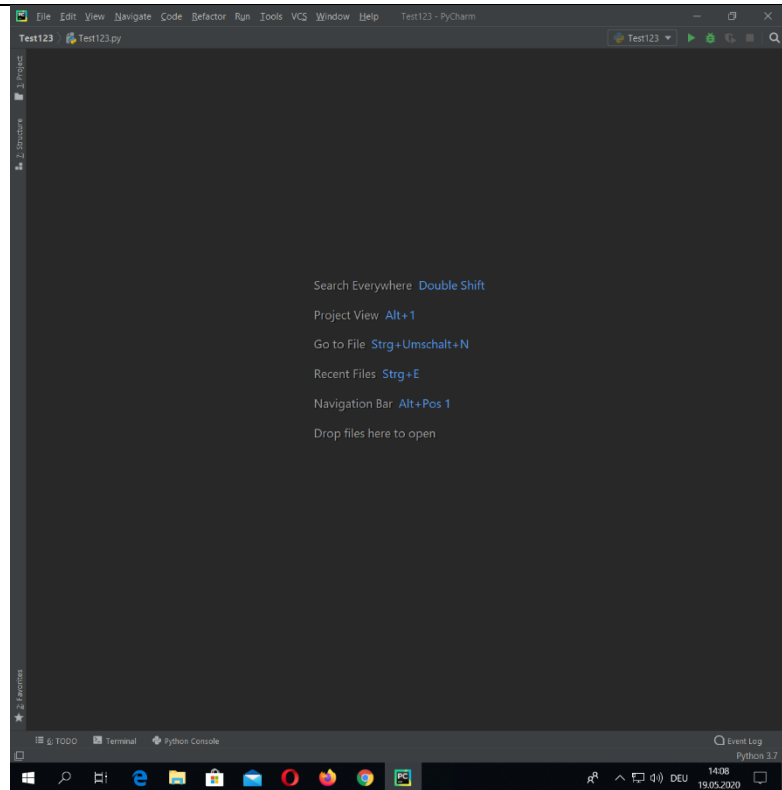


Tabelle 5 - PyCharm Installation

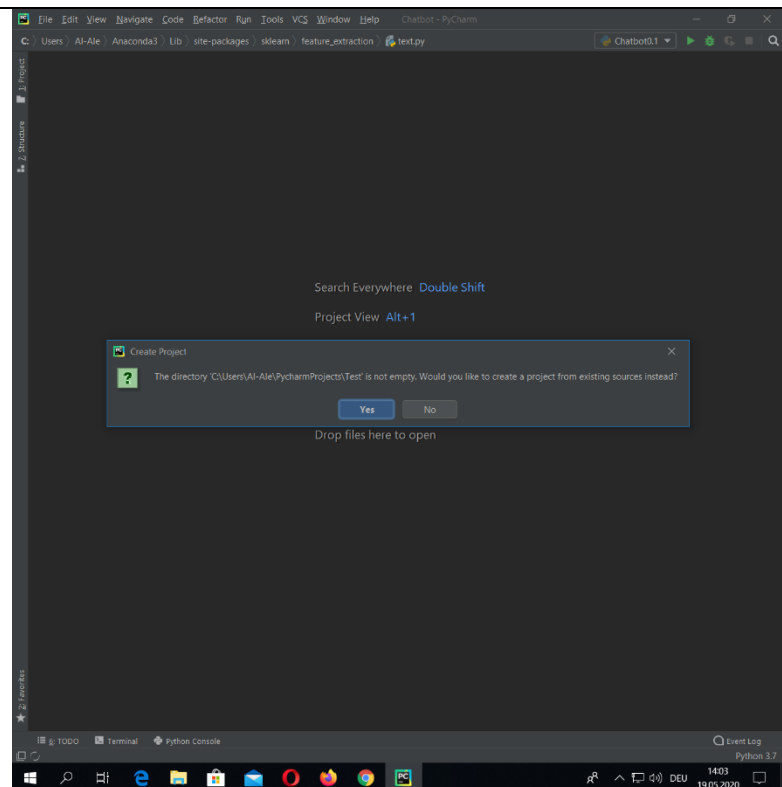
3.2.3.1 In PyCharm schreiben

Nun noch kurz wie man jetzt in PyCharm seine eigene Datei erstellt.

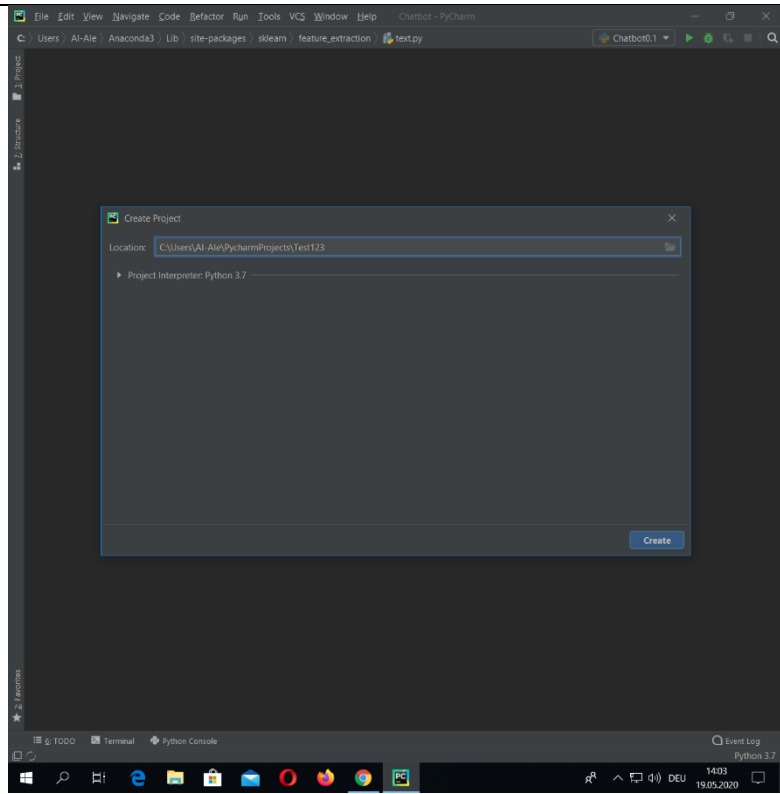
Als erstes Start man PyCharm ganz normal wie jede andere App, nach dem Start sollte dann eine Fenster, wie rechts zu sehen ist, aufgehen.



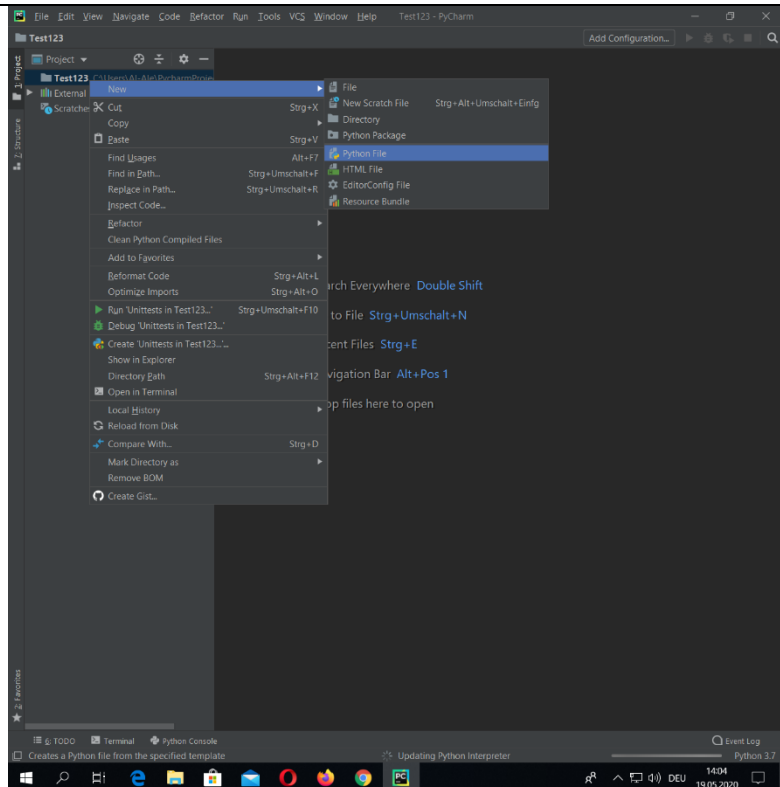
Als nächstes klickt man oben recht auf *File* und dann auf *New Project*.



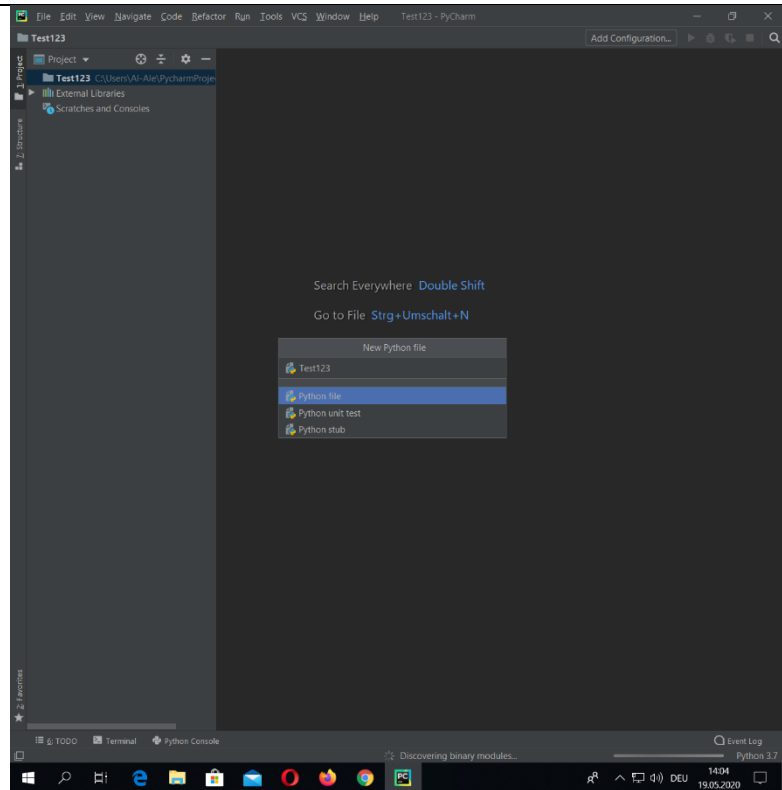
Im nächsten Fenster kann man dann den Namen des neuen Projektes angeben und danach auf *Create* drücken damit das Projekt erstellt wird.



Dann führt man oben Links einen links Klick auf den Projektnamen aus und dann findet man ganz oben den Tab New und dann wählt man die jeweilige Datei variante aus. In meinem Fall wähle ich natürlich die Python Datei.



Im nächsten Fenster kann man ganz schnell denn Namen der jeweiligen Datei noch angeben.



Jetzt kann man seinen Code im Feld einschreiben. Um es Auszuführen, muss man einen rechts Klick auf die Datei ausführen und dann auf *Run* drücken. Nach dem ersten Mal wird ab dann immer rechts oben ein Grünes Dreieck angezeigt, welches für *Run* steht angezeigt.

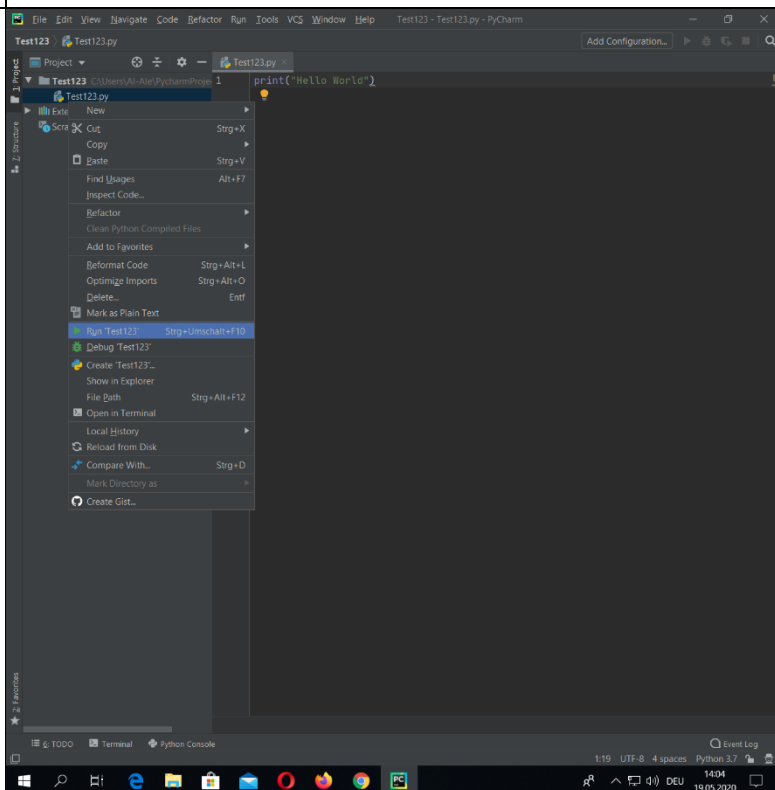


Tabelle 6 - PyCharm neue Datei/Projekt

3.3 Grund-Befehle

Print

Um in Python einen Text darzustellen nutzt man den Befehl `print()`.

Syntax
<code>print(„Text“)</code>

Tabelle 7 - Print Syntax

Comments

Für Comments gibt es drei verschiedene Schreibweisen.

Codezeile	Ausgabe
<code>#Das ist ein Kommentar</code>	
<code>""" Das ist ein Kommentar """</code>	

Tabelle 8 - Comments Syntax

3.4 Variablen

Variablen werden mit einem einfachen Gleichheitszeichen geschrieben.

Syntax
<code>variable = wert</code>

Tabelle 9 - Variablen Syntax

Dazu kann man noch mehrere Variablen auf einmal schreiben und verschiedene Variablen gleichsetzen. Schreibt das Wort `global` vor die Variabel, so gilt diese dann für die gesamte Code Seite.

Input	Output
<code>x = 5 print(x)</code>	5
<code>x, y, z = Blau, Rot, Gelb print(x) print(y) print(z)</code>	Blau Rot Gelb
<code>x = y = z = Blau print(x) print(y) print(z)</code>	Blau Blau Blau

Tabelle 10 - Variabel Varianten

Bei Variablen gibt es aber auch Regeln:

- Variablen dürfen nur mit einem Buchstaben oder unterstrich beginnen.
- Eine Variable darf nur A-z, 0-9 und unterstriche beinhalten.
- Variablen unterscheiden sich auch in Groß und Kleinschreibung

Man kann Variablen auch direkt in einen Satz einbinden

Input	Output
<code>x = Welt Print(„Hallo “ + x)</code>	Hallo Welt

Tabelle 11 - Variabel in Satz

3.5 Daten Typen

In Python gibt es viele verschiedene Datentypen, diese kann man ganz einfach mit diesem Befehl herausfinden. Dazu kann man einen Datentyp auch anpassen, dies geht mit dem zweiten Befehl. Somit kann man aus einem float ein gerundetes int machen.

Codezeile	Ausgabe
X= 'Hallo Welt' Print(type(x))	<class 'str'>
Print(int(20.5))	21

Tabelle 12 - Type Syntax

Oder wenn man zwei Zahlen in Strings hat werden die nicht ausgerechnet, sondern zusammengestellt. Möchte man sie rechnen kann man sie in ein int umwandeln

Codezeile	Ausgabe
a = „3“ b= „6“ print(a + b)	36
print(int(a)+int(b))	9

Tabelle 13 - Datentyp Umwandeln

3.5.1 Statische Variable

Als, erstes wenn man in einer Klasse eine Variable erstellt kann man diese über zwei Arten abrufen, die erste ist, wenn man die Klasse öffnet und dann mithilfe der gleichgesetzten Variable und der erstellten Variable auf den Inhalt zugreift.

Syntax
class Klassenname: variablenname = Inhalt c = Klassenname c.variablenname

Tabelle 14 - Syntax Statische Variable

Das c kann hierbei jede andere Art von Namen haben, solange es als Variable funktioniert. Oder man kann auch einfach direkt über den Klassennamen und Variablenname den Inhalt aufrufen.

Syntax
Klassenname.Variablenname

Tabelle 15 - Syntax Statische Variable mit Klassennamen

Jetzt kann man ganz einfach auch den Inhalt verändern.

Syntax
Klassenname.Variablenname = Neuer Inhalt

Tabelle 16 - Inhalt verändern

Dieser Inhalt ist jetzt allgemein verändert, das heißt, wenn jetzt nochmals, nach dieser Eingabe versucht auf diese Variable zuzugreifen, bekommt man den neuen Inhalt heraus.

3.6 Datentypen

Datentyp	Erklärung	Beispiel
str	Text Datentyp	X = 'Hallo Welt'
int	Alle Ganzzahlen	X = 54
float	Alle Dezimalzahlen	X = 23.5
Complex	Zahlen mit j am Ende	X = 1j
List	Liste mit mehreren Datensätzen	X = [„Apfel“, „Banane“, „Kirsche“]
Tuple	Liste mit mehreren Datensätzen	X = („Apfel“, „Bananen“, „Kirsche“)
Bool	Nur True or False	X = True

Tabelle 17 - Datentypen

Es gibt zwar noch mehr Datentypen, diese sind jedoch jetzt noch nicht von Bedeutung.

Mathe

Nummern

Es gibt 3 verschiedene Datentypen für Zahlen, und zwar:

Int:

- $X = 1$
- $Y = 436374537374553$
- $Z = -25326344$

Float:

- $X = 4.75$
- $Y = 5.0$
- $Z = -79.64$

Float mit für das 10fache:

- $X = 65e3$
- $Y = 45E2$
- $Z = -95e33$

Complex:

- $X = 6+7j$
- $Y = 8j$
- $Z = -3j$

Operatoren

Mathe ist in Python sehr einfach anwendbar. Zum Beispiel, wenn man etwas berechnen will. Dabei gibt verschiedene Zeichen für verschiedene Berechnungen.

Zeichen	Funktion	Aktiver Einsatz	Endergebnis
+	Addition	Print(4+2)	6
-	Subtraktion	Print(4-2)	2
*	Multiplikation	Print(4*2)	8
/	Division	Print(4/2)	2
**	Potenz Berechnung	Print(4^2)	16
%	Rest Berechnung	Print(4%2)	0
//	Teilt und rundet ab auf die nächste ganze Zahl.	Print(15//2)	7

Tabelle 18 - Mathe Operatoren

Random Modul

Das Random Modul importiert einen Zufalls Generator.

Syntax
<pre>Import random random.randrange(min. Zahl, max. Zahl))</pre>

Tabelle 19 - Random Modul

3.7 Listen

In Python gibt es sogenannte Listen, in welchen man mehrere Daten eingeben kann.

Syntax
<pre>liste = [Inhalt 1, Inhalt 2, etc.]</pre>

Tabelle 20 - Listen

Append()

Mit der Funktion `append()` kann man auch ein weiteres Element hinzufügen.

Syntax
<code>liste.append(Inhalt Input)</code>

Tabelle 21 - Append Funktion

Len()

Mithilfe von der `len()` Funktion, kann man anzeigen lassen wie viele Einträge die Liste hat. Dazu, wenn man einen String angibt, gibt es die Anzahl Zeichen aus.

Syntax
<code>len(liste)</code>

Tabelle 22 - len Funktion

Mit dem folgenden Befehl greift man auf eine bestimmte Eingabe zu. Man sollte hierbei beachten das man immer bei 0 und nicht bei 1 startet, dies hat historische Gründe(C).

Syntax
<code>liste[Index]</code>

Tabelle 23 - Index

Mithilfe der Minus Schreibweise kann man auch von hinten beginnen den Index abzufragen. Schreibt man zum Beispiel -1 so gibt es das letzte Element einer Liste aus.

Syntax
<code>liste[-Index]</code>

Tabelle 24 - Index von hinten

pop()

Mithilfe der `pop()` Funktion kann man ganz einfach den neusten Eintrag entfernen. Man kann diese Funktion auch mehrmals hintereinander ausführen, dabei wird immer das letzte Element entfernt.

Syntax
<code>liste.pop()</code>

Tabelle 25 - pop Funktion

Dazu kann man den Befehl einer Variabel zuweisen und dies dann ausgeben, dabei wird dann das entfernte Element angezeigt.

Liste -> String

Mithilfe eines Separators und der `.join()` Funktion kann man eine Liste in einen, einzelnen String umwandeln. Dabei kann man dies direkt mit `print` ausgeben oder in eine Variable einfügen.

Syntax
<code>(„Separator“ .join(liste))</code>

Tabelle 26 - join Funktion

String -> Liste

Mithilfe eines Separators und der `.split()` Funktion kann man einen String in eine Liste umwandeln.

Syntax
<code>liste.split(„Separator“)</code>

Tabelle 27 - split Funktion

3.8 Listen Teil 2

3.8.1 Eintrag Löschen

Mit `del` kann man bestimmte Teile der Liste löschen.

Syntax

del liste[indexnummer]

Tabelle 28 - del Funktion

Mit remove() kann man auch Teile der Liste Lösche, der Vorteil hierbei ist, dass man denn Inhalt angeben kann.

Syntax

liste.remove(„Inhalt“)

Tabelle 29 - remove Funktion

3.8.2 List slicing

Bei List Slicing wählt man ein Gebiet aus welches ausgegeben wird, als ersten Parameter wählt man den Start Punkt, als zweiten den Endpunkt, der zweite Parameter selber wird nicht angezeigt.

Syntax

liste[indexstart:indexende]

Tabelle 30 - List Slicing

Gibt man keinen zweiten Parameter an, so zeigt es alles vom ersten Startpunkt bis zum Ende.

Das List Slicing funktioniert auch auf String, nur dabei sind es die Buchstaben welche man mit den Zahlen auswählt.

3.8.3 List Comprehension

Bei List Comprehension geht es darum Listen einfach in andere umzuwandeln. Beispiel: Du hast Liste x und willst jeden Wert in verdoppeln und diese Werte in y einfügen.

Input	Output
<pre>xs = [1, 2, 3, 4, 5] ys = [x * 2 for x in xs] print(xs) print(ys)</pre>	<pre>[1, 2, 3, 4, 5] [2, 4, 6, 8, 10]</pre>

Tabelle 31 - List Comprehension

Oder wenn man eine Liste mit Namen hasst und du willst von jedem die Buchstabenlänge wissen.

Input	Output
<pre>students = [„Max“, „Monika“, „Erik“, „Franziska“] lengths = [len(student) for student in students] print(lengths)</pre>	<pre>[3, 6, 4, 9]</pre>

Tabelle 32 - List Comprehension mit len

3.8.4 Tuple

Funktionieren ähnlich wie Listen.

Syntax

variable = (Wert, Wert, etc.)

Tabelle 33 - Tuple

Der erste unterschied ist das Tuple unveränderlich sind, das heißt Code wie append und weitere funktioniert bei Tuple nicht. Ein Vorteil dabei ist, das man diese Listen nicht mehr überschreiben kann.

Dazu kann den Werten in einer Tuple Tabelle noch Objekte hinzufügen. Also zum Beispiel:

Input	Output
<pre>student = („Max Mustermann“, 22, „Informatik“) name, age, subject = student</pre>	<pre>Max Mustermann Informatik</pre>

<code>print(name)</code> <code>print(subject)</code>	
---	--

Tabelle 34 - Tuple Beispiel

Man kann auch mehrere Tuple in eine Liste einfügen.

Syntax
<pre>variable = [(„Wert1“, Wert2) („Wert1“, Wert2)]</pre>

Tabelle 35 - Mehrere Tuple in einer Liste

Man kann auch die obige Liste in eine For schleife einbinden, zum Beispiel:

Input	Output
<pre>students = [(„Max“, 22) („Monika“, 23)] for name, age in students: print(name) print(age)</pre>	<pre>Max 22 Monika 23</pre>

Tabelle 36 - Liste mit for Schleife

3.8.5 Dictionaries

In Dictionaries kann man Elementen Werten hinzufügen.

Syntax
<code>variable = {„Wert1“ : „Wert2“, etc.}</code>

Tabelle 37 - Dictionaries Syntax

Als zum Beispiel, Städten deren Flughäfen.

Input	Output
<pre>d = {„Berlin“: „BER“, „Helsinki“: „Hel“} print(d[„Berlin“])</pre>	<pre>BER</pre>

Tabelle 38 - Dictionaries Beispiel

Man kann auch nachträglich Werte einfügen.

Syntax
<code>variable[„Wert1“] = „Wert“</code>

Tabelle 39 - Dictionaries nachträgliche Einträge

Dabei ist jedoch nicht eine korrekte Einsortierung gewährt, jedoch braucht man diese in Dictionaries auch nicht.

Man kann Werte auch wieder entfernen.

Syntax
<code>del d[„Wert1“]</code>

Tabelle 40 - Del Funktion bei Dictionaries

Man kann auch mit `.get` denn Wert ausgeben, dabei ist der Vorteil gegenüber der anderen Ausgabe dass, falls der gesuchte Wert nicht vorhanden ist, gibt die andere Ausgabe einen Error an, `.get` gibt nur `none` aus.

Syntax

```
variable.get(„Wert1“)
```

Tabelle 41 - get Funktion bei Dictionaries

Aber der Error kann in diesem Fall sagen das etwas nicht funktioniert, im anderen Fall würde man den Fehler nicht bemerken.

Ich persönlich empfehle die erste Schreibweise.

items

Mithilfe der Funktion items kann man die Werte von einer Dictionarie als Tuple ausgeben.

Syntax

```
variable.items()
```

Tabelle 42 - items Funktion

Somit kann man jetzt auch mit einer Schleife die Werte Einzel ausgeben, hierzu ein Beispiel.

Input	Output
<pre>d = {„München“: „MUC“, „Budapest“: „BUD“} for key, value in d.items(): print(key + „: “ + value)</pre>	<pre>München: MUC Budapest: BUD</pre>

Tabelle 43 - Schleife und Dictionaries Beispiel

3.8.6 Listen Verschachteln

Dabei geht es ganz einfach darum mehrere Listen in einer Liste zu haben oder noch mehr.

Syntax

```
variable = [
    [„Wert1“, „Wert2“, etc.]
    [„Wert1“, „Wert2“, etc.]
]
```

Tabelle 44 - Listen Verschachteln Syntax

Um jetzt zum Beispiel auf den zweiten Wert der zweiten Liste zuzugreifen muss man folgendes eingeben.

Syntax

```
variable[1][1]
```

Tabelle 45 - Index bei Verschachtelung

3.8.7 Liste in Dictionarie

Hierbei kann man eine Liste in ein Dictionarie einfügen.

Syntax

```
variable = {
    „Objekt1“: [„Wert1“, „Wert2“, etc.]
    „Objekt2“: [„Wert1“, „Wert2“, etc.]
}
```

Tabelle 46 - Listen in Dictionaries

3.9 Boolesche Werte

Bei Booleschen Werten wird am Ende einfach nur ausgegeben ob die Aussage, des Input stimmt.

Input	Output
Print(12 < 2)	False
Print(12 == 2)	False
Print(12 > 2)	True

Tabelle 47 - Boolesche Werte

Dieser Datentyp wird am meisten bei if und else genutzt.

3.10 Kontrollstruktur

3.10.1 if, else

If, else und elif dienen dazu Abfragen zu erstellen, dabei wird eine bedingung nachdem if erstellt, wichtig bei dieser Bedingung ist, dass die Ausgabe True oder False ist.

Syntax
if Bedingung: Aufgabe wenn Bedingung True

Tabelle 48 - if Syntax

Hierzu ein kleines Beispiel:

Input	Output
x = 30 if n < 42: print(„Die Zahl n ist kleiner als 42“)	Die Zahl n ist kleiner als 42

Tabelle 49 - if Beispiel

Wie man sieht wird der print ausgeführt da die Nachfrage bei if stimmt. Der print Befehl ist eingerückt da es das Tochter Element von if ist.

Beim oberen Beispiel wird jedoch bis jetzt nicht ausgegeben wenn n grösser als 42 ist, um das zu ändern braucht man noch else.

Input	Output
x = 56 if n < 42: print(„Die Zahl n ist kleiner als 42“) else: print(„Die Zahl n ist grösser als 42“)	Die Zahl n ist grösser als 42

Tabelle 50 - if und else Beispiel

Bei else wird keine Bedingung wie $n > 42$ festgelegt, da es nur eingesetzt wird, wenn der if und alle elif befehle falsch sind.

3.10.2 Vergleichsoperatoren

Vergleichsoperatoren können zum einen bei if, else eingesetzt werden oder auch ganz normal. Bei ihnen wird immer entweder True oder False ausgegeben. True oder False sind vom Datentyp Boolean.

Input	Output
print(2 < 4)	True
print(2 > 4)	False

Tabelle 51 - Vergleichsoperator Beispiele

Dabei gibt es noch mehr Vergleichsoperatoren.

Operator	Funktion
x < y	x ist kleiner als y
x > y	x ist grösser als y
x <= y	x ist kleiner oder gleich groß wie y
x >= y	x ist grösser oder gleich groß wie y
x == y	x ist gleich grösser wie y
x != y	x und y sind nicht gleich

Tabelle 52 - Vergleichsoperatoren

3.10.3 Booleans (and und or)

Wenn man einen Bereich abfragen will wie zum Beispiel von bis, braucht man 2 abfragen.

Input	Output
age = 26 if age >= 20: if age <= 29: print(„Diese Person ist in ihren 20-ern.“)	Diese Person ist in ihren 20-ern.

Tabelle 53 - 2 if Abfragen

Dies verbraucht aber Platz und ist unübersichtlicher, somit gibt es zum Glück and und or.

And

And wird eingesetzt, wenn man zwei oder mehr Bedingungen gleichstellen will, das heißt alle Bedingungen mit einer and Verbindung müssen True sein, sonst wäre das Endergebnis False.

Input	Output
age = 26 if age >= 20 and age <= 29: print(„Diese Person ist in ihren 20-ern.“)	Diese Person ist in ihren 20-ern.

Tabelle 54 - And

Somit muss diese Person 20 oder älter sein und 29 oder jünger.

Or

Hierbei muss nur eine der angegebenen Bedingungen True ergeben damit das Endergebnis auch True ergibt.

Input	Output
age = 32 if age < 20 or age >= 30: print(„Diese Person ist nicht in ihren 20-ern.“)	Diese Person ist nicht in ihren 20-ern.

Tabelle 55 – Or

Beispiel für And und Or

Input	Output
<pre>country = „US“ age = 20 if (country = „US“ and age >= 21) or (country != „US“ and age >= 18): print(„Diese Person darf Alkohol konsumieren.“) else: print(„Diese Person darf kein Alkohol konsumieren.“)</pre>	Diese Person darf kein Alkohol konsumieren.

Tabelle 56 - And und Or Beispiel

Hierbei werden Klammern verwendet da diese immer als erstes „Berechnet“ wird.

in-Operator

Der in-Operator ist ein Operator wie = oder >. Dieses sucht ein Element, Begriff, etc. in einer Menge.

Input	Output
<pre>students = [„Moritz“, „Max“, „Maira“] print(„Moritz“ in students) print(„Matilda“ in students)</pre>	True False

Tabelle 57 - in-Operator

Da der in-Operator auch True oder False ausgibt kann man ihn auch bei if Statements nutzen.

Hier ein kleines Beispiel:

Input	Output
<pre>if „Moritz“ in students: print(„Moritz studiert.") else: print(„Moritz studiert nicht. if „Matilda“ in students: print(„Matilda studiert.") else: print(„Matilda studiert nicht.“)</pre>	<pre>Moritz studiert. Matilda studiert nicht.</pre>

Tabelle 58 – in-Operator Beispiel 1

Man kann sogar mit in innerhalb eines Satzes nach Zeichen, Wörtern, etc. suchen.

Input	Output
<pre>sentence = „Ich mag Python sehr!“ if „?“ in sentence: print(„JA“) else: print(„NEIN“)</pre>	<pre>NEIN</pre>

Tabelle 59 - in-Operator Beispiel 2

3.10.4 not-Operator

Der not-Operator sagt einer Bedingung dass es nur True ist wenn die Bedingung nicht stimmt.

Input	Output
<pre>age = 43 if not age < 40: print(„Älter als 40“) else: print(„Jünger als 40“)</pre>	<pre>Älter als 40.</pre>

Tabelle 60 - not-Operator

Dazu kann man noch den not-Operator vor den in-Operator setzten um zu schauen ob etwas nicht vorhanden ist.

Input	Output
<pre>students = [„Max“, „Moritz“] if „Matilda“ not in students: print(„Matilda ist nicht vorhanden.“) else: print(„Matilda ist vorhanden“)</pre>	<pre>Matilda ist nicht vorhanden</pre>

3.10.5 Elif

Elif soll ganz einfach helfen, dass man nicht mehr super viele if und else benutzt. Und zwar nutzt man als erstes immer noch if und für alles was nicht abgefragt wurde else, will man jedoch noch weitere Bedingungen festlegen nutzt man ab jetzt einfach elif. Es funktioniert gleich wie if nur das nach dem ersten if man so viele elif erstellen kann wie man will.

Input	Output
<pre> currency = „€“ if currency = „\$“: print(„Dollar“) elif currency = „£“: print(„Pfund“) elif currency = „€“: print(„Euro“) else: print(„ldk“) </pre>	Euro

3.10.6 While – Schleife

Die While Schleife ist wie das if Statement, jedoch wiederholt es seine Aufgabe bis ein vorgegebenes Limit erreicht ist.

Input	Output
<pre> counter = 0 while counter < 10 : print(counter) counter = counter + 1 </pre>	0 1 2 3 4 5 6 7 8 9

3.10.7 For – Schleife

Die For Schleife funktioniert ähnlich wie die while schleife, sie endet einfach sobald sie die vorgegebene Liste abgearbeitet hat.

Input	Output
<pre> liste = ["Max", "Moritz", "Matilda", "Maira"] for i in liste: print(i) </pre>	Max Moritz Matilda Maira

Syntax: for (variabel) in (liste/zahlenrange)

Die variabel spricht hierbei immer die liste an.

3.10.8 Welche Schleife?

While Schleife:

Lässt man das +1 ausversehen weg so entsteht eine Endlosschleife, welche sehr schnell für Probleme sorgt, da sie Endlos ist...

For Schleife

Besser da man direkt zu Anfang die Länge der der Wiederholungen angibt und somit es nicht sehr schnell zu Problemen kommt.

3.10.9 Break & Continue

Continue

Continue wird innerhalb einer For-Schleife genutzt und zwar in einem if. Dabei wird der jeweilige abschnitt übersprungen. Setzt man also zum Beispiel `i == 3` als Bedingung fest, so wird die 3 übersprungen.

Input	Output
<pre>for i in range(0, 10) : if i == 3 : continue print(i)</pre>	<pre>0 1 2 4 5 6 7 8 9</pre>

Break

Break wird gleich eingebunden wie das continue, nur wird an der Stelle die gesamte Schleife abgebrochen.

Input	Output
<pre>liste = [4, 6, 7, 2, 4, 6, 7] s = 0 for e in liste: s = s + e if s >= 10: break print(s)</pre>	<pre>17</pre>

Das heißt Continue und Break sind sehr wichtig, da man nach Bedingung etwas überspringen oder beenden kann.

Listeninhalt von 2 Listen abfragen

<pre>continents = [„Amerika“, „Südamerika“, „Australien“] others = [„Amerika“, „Asien“, „Afrika“] for element in continents: if element in others: print(element)</pre>	<pre>Amerika</pre>
--	--------------------

3.11 Funktionen

3.11.1 Warum Funktionen?

Bisher hat man seinen Code nur runtergeratert jedoch sobald man selbst oder andere den Code nochmal anschauen merkt man sehr schnell das er sehr unübersichtlich ist. Funktionen sind dafür die perfekte Lösung, mit ihnen kann man Code in Logische Blöcke aufteilen und somit wird, dass Programm um einiges übersichtlicher.

3.11.2 Funktionen

Funktionen habe ich schon verschiedene vorgestellt dazu gehört auch die `print()` Funktion. Hierbei ist `print` die Funktion und in den klammern wird die Funktion aufgerufen.

3.11.3 Eigene Funktionen

Um eine eigene Funktion zu erstellen muss man es wie folgt ein Syntax machen.

Syntax
<pre>def (Name der Funktion)(Parameter): Aufgaben und Eigenschaften der Funktion(print, etc.)</pre>

Jetzt muss man nur noch wie folgt die Funktion aufrufen.

Syntax
<pre>(Name der Funktion)()</pre>

Jetzt wird das ausgegeben was man der Funktion oben zur Aufgabe gegeben hat. Der obere Syntax so gesehen das Rezept und der unter führt dieses aus.

Beispiel mit Parameter

Input	Output
<pre>def printer(name, count): for in range (0, count): print(name) printer(„Hallo“, 5)</pre>	<pre>Hallo Hallo Hallo Hallo Hallo</pre>

Dazu können funktionen auch in anderen Funktionen eingesetzt werden.

3.11.4 return

Mit dem Befehl return gibt man die Lösung oder das Endergebnis als die Funktion aus. Es definiert so gesehen das Ende der Funktion.

Input	Output
<pre>def max(a, b): if a > b: return a else: return b result = max(6, 9) print(result)</pre>	<pre>9</pre>

3.12 Dateien Öffnen, Lesen, Schreiben

3.12.1 Datei lesen

Die Datei wird ganz einfach mit der Funktion open geöffnet. Der zweite Parameter ist jetzt ein r, damit Python Weiß, dass die Datei zum lesen da ist.

Syntax
<pre>variable = open(„Datei.txt“, „r“)</pre>

Um sie jetzt auszulesen kann man zum Beispiel die For-Schleife nutzen. Wichtig dabei ist alle \n zu entfernen da es sonst zu viel White Space kommt. Das entfernt man mit .split().

Input	Output
<pre>file = open(„datei.txt“, „r“) for line in file: print(line.strip())</pre>	<pre>Zeile 1 von datei Zeile 2 von datei Zeile 3 von datei</pre>

3.12.2 \n Befehl

Wenn immer Python auf den Befehl \n stößt erstellt Python einen Zeilenumbruch.

3.12.3 Datei schreiben

Nun öffnet man wieder eine Datei mit dem Befehl open, nur jetzt ist der zweite Parameter w, damit Python Weiß, dass die Datei zum Schreiben ist.

Syntax
<pre>variable = open(„Datei.txt“, „w“)</pre>

3.12.4 close()

Es ist auch wichtig, wenn man die Datei öffnet sie auch wieder zu Schließen und zwar mit `.close()`.

Syntax

```
variable.close()
```

3.12.5 write()

Mit `.write()` wird ein Text in die geöffnete Datei eingefügt.

Syntax

```
variable.write(„Hallo Welt“)
```

3.12.6 Daten in Dateien hinzufügen

Als letzten Parameter gibt es noch `a`, dieser steht für `append` und wird zum Daten hinzufügen genommen. Unterschied zu `write` ist dass er bei jedem mal ausführen die Daten nochmal hinzufügt.

Syntax

```
variable = open(„Datei.txt“, „a“)
```

3.12.7 With Konstrukt

Mit dem `With` Konstrukt, Weiß Python dass es eine Datei offen hat und diese, wenn sie nicht gebraucht wird, geschlossen wird.

Syntax

```
with open(„dateiname.txt“, „parameter“) as variable:
```

3.12.8 CSV Dateien öffnen

Mit dem folgenden Befehl kann man den Inhalt einer CSV Datei, Dateiformat für strukturierte Daten, in eine Python Tabelle umwandelt.

Syntax

```
data = (variable1.strip()).split(„;“)
```

Wenn man jetzt aber nach Zahlen sucht oder Zahlen aus der CSV nimmt sollte man beachten das diese als string importiert werden. Das heißt man sollte sie vorher noch in einen integer umwandeln.

3.13 Grafiken

3.13.1 Grafik zeichnen

Um Grafiken überhaupt zu erstellen muss man zuerst die folgenden zwei Zeilen code einfügen. Die obere sorgt dafür das die Grafiken direkt erscheinen und die untere fügt das Modul ein welches uns überhaupt erst ermöglicht Grafiken zu machen. Auf Module werde ich später noch eingehen.

Syntax

```
%matplotlib inline
import matplotlib.pyplot as plt
```

Als nächstes gibt man die `x` und `y` Werte an.

Syntax

```
xs = [x werte...]
ys = [y werte...]
```

Als letztes lässt man es noch mit folgendem Code darstellen

Syntax

```
plt.plot(xs, ys)
plt.show()
```

Beispiel Geburtstatistiken USA

Input

```
xs = []
ys = []

name = "Kevin"      #<---- Name
gender = "M"        #<---- Geschlecht
state = "TX"        #<---- Staat

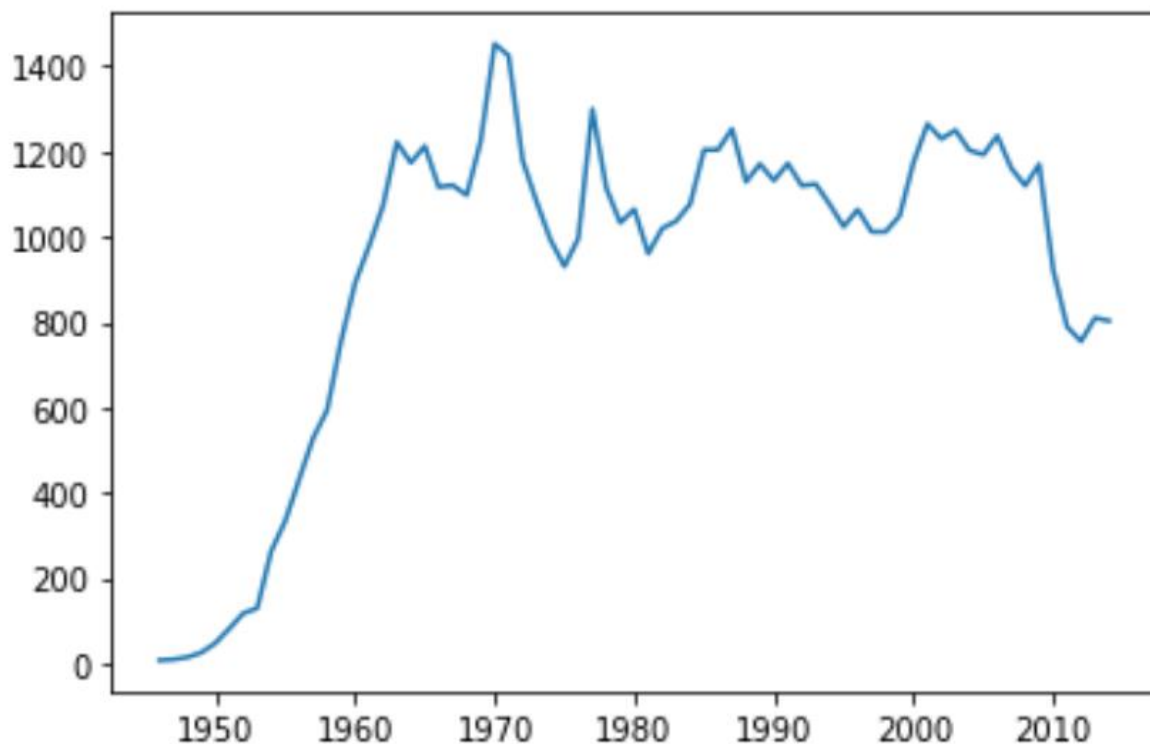
with open("names.csv", "r") as names:
    for line in names:
        data = line.strip().split(",")
        if data[1] == name and data[3] == gender and data[4] == state:
            xs.append(int(data[2]))
            ys.append(int(data[5]))

print(xs)
print(ys)

%matplotlib inline
import matplotlib.pyplot as plt

plt.plot(xs, ys)
plt.show()
```

In diesem Beispiel lässt man alle männlichen Kinder aus Texas mit dem Namen Kevin anzeigen.
Und so sieht das Diagramm dann aus:



3.14 Objektorientierung

Objektorientierung wird eigentlich bereits ausgeführt, wenn man `liste.append(...)` eingibt, denn man orientiert sich an dem Objekt welches in diesem Fall die Liste ist und dann die Methode `.append` ausführt.

3.14.1 Klassen und Methoden

Als erstes erstellt man eine Klasse und zwar wie folgt:

Syntax
<pre>class Classname(): pass</pre>

Es wird empfohlen den Classname immer mit einem Großbuchstaben zu beginne. Der Code `pass` sorgt dafür, dass Python weiß, dass man es okay ist, dass die Class leer ist. Jetzt kann man noch Eigenschaften definieren für die obige Klasse.

Syntax
<pre>variable = classname() variable.beschreib = „Wert1“ variable.beschreib = „Wert2“</pre>

Jetzt ist die Frage wozu braucht man, das? Ich zeige dies am besten in einem Beispiel:

Input	Output
<pre>class Student(): pass def get_name(student): print(student.firstname + " " + student.lastname) erik = Student() erik.firstname = "Erik" erik.lastname = "Mustermann" get_name(erik) monika = Student() monika.firstname = "Monika" monika.lastname = "Müller" get_name(monika)</pre>	Erik Mustermann Monika Müller

In diesem Beispiel sieht man das man mehrere Variablen nimmt und zwar `erik` und `monika` und weil man diese als Variablen von `Student` nimmt kann man noch viel mehr Namen so hinzufügen und diese nur mit `student` alle ansprechen. Somit muss man weder super viele `print` befehle machen, noch große Schritte machen, wenn man einen neuen `Student` hinzufügen will.

3.14.2 Constructor und Methoden erstellen

Was macht man jetzt aber, wenn man eine Liste hat mit zwei werten, der erste ist der `firstname` und der zweite ist der `lastname`. Jedoch versteht Python nicht welcher Wert `first` und welcher `lastname` ist. Dafür gibt es die Funktion `__init__` von Python, welche man aber ganz normal zuerst Aktivieren/erstellen muss. Beispiel:

Input	Output
<pre>class Student(): def __init__(self, firstname, lastname): self.firstname = firstname self.lastname = lastname def get_name(student): print(student.firstname + " " + student.lastname) erik = Student(„Erik“, „Mustermann“) erik.name()</pre>	Erik Mustermann Monika Müller


```
monika = Student(„Monika“, „Müller“)
monika.name()
```

Hierbei wird mit dem Self immer die Klasse angesprochen, in welcher man sich befindet. Dieser init wäre jetzt ein Constructor.

3.14.3 Methoden und Eigenschaften

Die nächsten Abschnitte lassen sich ganz einfach durch ein reales Beispiel erklären, der Supermarkt aus der Sicht des Kunden. Was will der Kunde vom Supermarkt? Er will dort einkaufen und dafür braucht man ganz einfach die Methode einkaufen (). Aber damit diese Methode funktioniert werden noch mehr Dinge funktionieren, die Kassierer müssen beschäftigt werden, der Strom muss bezahlt werden und auch neue Produkte müssen bestellt werden. Das sind jedoch alles Dinge, die vor dem Kunden versteckt werden. Das heißt wenn der Kunde zum Beispiel Äpfel kauft und der Lagerbestand niedriger wird, soll der Supermarkt automatisch neue Bestellen. Auf diese kann/darf ich nicht zugreifen. In Python ist es ziemlich ähnlich, wenn man zum Beispiel eine Klasse schreibt welche eine Website herunterlädt, will man ja nicht wissen wie die Verbindung aufgestellt wird und die einzelnen Datenpakete heruntergeladen werden, man will nur, dass man am Ende die Website heruntergeladen hat. In den nächsten Kapiteln geht es genau eben um diese Sichtbarkeit.

3.14.4 Private Eigenschaften und Methoden

Um eine Private Eigenschaft zu erstellen zu erstellen braucht es, gar nicht viel Aufwand.

Syntax

```
__Eigenschaft
```

Zwei Bodenstrich vor die Eigenschaft und schon ist sie Privat, das heißt eben das man nicht mehr auf sie zugreifen kann. Aber man kann auch nur einen Bodenstrich vor die Eigenschaft setzen, denn das bedeutet für jeden Entwickler, dass man diese nur im Notfall nutzen soll, aber das ist nur ein Hinweis und keine Einschränkung.

3.14.5 Besondere Methoden

Wenn man eine solche Klasse erstellt und ihr eine Variable zuweist kann man mit dieser Variable noch genauere Infos abnehmen.

Als erstes kann man einfach nur die variable im code block eingeben. Dabei sollte in etwa folgendes dargestellt werden:

```
<__main__.student at 0x1b3f1af7588>
```

Diese Ausgabe ist aber nicht sehr hilfreich und deshalb kann man die auch konfigurieren und zwar:

Syntax

```
def __repr__(self):
    return "text"
```

Das Self steht wiederum wieder für die Klasse.

Jedoch kann man auch print(variable) eingeben und dabei kommt wieder dieser komische Code von oben doch auch diesen kann man mit folgender Methode verändern.

Syntax

```
def __str__(self):
    return "text"
```

Als letztes kann man auch mit print(len(variable)) die anzahl eingaben oder sonst die Länge angeben. Aber damit das funktioniert muss man wieder eine Methode erstellen.

Syntax

```
def __len__(self):
    return länge
```

3.14.6 Vererbung

Bei der Vererbung geht es ganz einfach darum das, wenn man bereits eine Methode geschrieben hat, dass man diese übernehmen kann und sie auf einen Schlag überall verändern kann. Um den Sinn und die Effektivität gut zu beschreiben, kommt jetzt ein Beispiel.

Beispiel

```
class Student():
    def __init__(self, firstname, surname):
        self.firstname = firstname
        self.surname = surname
        self.term = 1

    def name(self):
        return self.firstname + " " + self.surname

class WorkingStudent(Student):
    def __init__(self, firstname, surname, company):
        super().__init__(firstname, surname)
        self.company = company
    def name(self):
        return "Working Student: " + self.firstname + " " + self.surname

erik = Student("Erik", "Mustermann")
print(erik.name())
erik = WorkingStudent("Erik", "Mustermann", "Bosch GmbH")
print(erik.name())

students = [
    WorkingStudent("Max", "Müller", "GF"),
    Student("May", "Parotti"),
    WorkingStudent("Oliver", "Queen", "SIG"),
    Student("Laurel", "Lance")
]
```

Als erstes erstellt man ganz oben eine Klasse, in diesem Fall die Klasse Student. In diese wird die Methode `__init__` gemacht, welcher später dann noch einmal gebraucht wird. Als zweite Methode wird Name erstellt. Jetzt kommt die zweite Klasse und zwar WorkingStudent. Jetzt ist wichtig, um alles richtig zu vererben muss man in die Klammer dahinter die obere Klasse, welche man nutzen will, in diesem Fall Student. Als nächstes gibt man `super().` ein und nachdem Punkt gibt man den Namen der Methode ein die man nehmen will, in diesem Fall wäre das `__init__`. Somit sind jetzt alle Eigenschaften von `__init__` aus der Klasse Student übernommen. Würde man jetzt nicht nochmal die Methode Name erstellen würde sie einfach von der Klasse Student übernommen werden. Erstellt man diese aber kommen sie sich, obwohl sie gleich heißen, nicht in die Quere. Jetzt kann entweder die Namen einzeln zuweisen oder sogar alle Daten in eine Liste verpacken. Und solange man vor die Klammer den Namen der Klasse schreibt, weiß Python auch immer welche name Methode es benutzen soll.

3.14.7 Type feststellen

Wenn man eine Klasse erstellt so werden spätere eingaben dieser Klasse zu einem Typ. Beim oberen Beispiel wäre Max Müller zum Beispiel vom Typ her ein WorkingStudent und May Parotti hätte denn Typ Student. Man kann abfragen von welchem Typ jemand ist, mit der folgenden Abfrage.

Syntax

```
type(Name des Werts Bsp. erik)
```

Mithilfe dieser Abfrage kann man auch wieder ein if statment erstellen und schauen wer wohin kommt. Man kann auch eine isinstance Abfrage machen, diese geht wie folgt.

Syntax
isinstance(Name des Werts Bsp. erik, gesuchter Typ)

Das kann man auch bei listen machen, nur da braucht man eine for schleife um jede Zeile Einzel durchzumachen.

3.14.8 Alles ein Objekt

Das einfachste Beispiel hierbei ist, die Plus rechnung in Python. Dabei wird nämlich auch eine Methode genutzt um auf die Lösung zu kommen.

Vorne Durch	Im Hintergrund
12.5 + 5	(12.5).__add__(5)

Bei beiden Rechnungen kommt man auf die Lösung 17.5.

Auch bei der Funktion len, passiert etwas Anderes im Hintergrund.

Vorne Durch	Im Hintergrund
len(„Hallo“)	(„Hallo“).__len__()

Aber hier sowie bei weiteren Funktionen wurde es für den Nutzer schnell und einfacher gemacht. Somit kann man auch die verschiedenen Objekte definieren.

Input	Output
<pre>class A: def __len__(self): return 6 instance = A() len(instance)</pre>	6

In diesem Beispiel wird die len Funktion, verändert und gibt in diesem fall immer diese Sechs aus, auch wenn man ein längeres Wort einbindet.

3.14.9 Wie Klassen/Variablen benennen?

Es gibt drei verschiedene Schreibweisen bei der Benennung:

- PascalCase (IchBesteheAusMehrerenWörtern)
 - Hierbei werden jeweils der Erste Buchstabe und ab dann jeder erste Buchstabe in einem Wort großgeschrieben. Alles wird zusammengeschrieben.
- camelCase (ichBesteheAusMehrerenWörtern)
 - Hierbei wird jeweils der erste Buchstabe klein geschrieben, alle darauffolgenden werden großgeschrieben. Auch hier wird alles wieder zusammen geschrieben.
- sneak_case (ich_bestehe_aus_mehreren_wörtern)
 - Bei dieser Schreibweise wird alles klein geschrieben. Zwischen den Wörtern befindet sich dazu keine normale Leerstelle, sondern ein unterstrich.

3.14.9.1 Wann wird was verwendet?

Für Klassen wird meistens PascalCase genutzt, für Variablen und Funktionen wird normalerweise sneak_case genutzt. camelCase wird in Python gar nicht genutzt, das sind jedoch alles nur Empfehlungen, man sollte jedoch immer eine diese Drei Schreibweisen nutzen, um eine grössere Übersichtlichkeit zu haben. Dennoch sollte man am Ende immer versuchen den gesamten Namen so kurz wie irgend möglich zu halten.

3.15 Module in Python

Das Grundziel der Module ist es den Code auf verschiedenen Dateien zu verteilen und somit den Code übersichtlicher zu machen. Eigentlich ist es ähnlich wie ein Puzzle, man erstellt einen Baustein und setzt diese dann richtig, ein. Am Ende funktioniert alles und macht Sinn.

3.15.1 Modul erstellen

Das einbinden oder besser gesagt das Erstellen eines Moduls und dessen Import sind super einfach. Als erstes erstellt man eine Datei und fügt dort den Code für dieses Modul ein. Wichtig dabei ist das der Name der Datei einfach und Logisch ist. Der Dateityp ist .py. Jetzt muss man nur noch in der Hauptdatei folgendes eingeben.

Syntax

```
import Dateiname
```

Nun kann man alle erstellten Funktionen, Variablen und weiteres nutzen. Man kann auch bestimmte Funktionen aus dem Modul laden und zwar mit folgendem Syntax.

Syntax

```
from Dateiname import Funktion/en
```

Der Unterschied ist vor allem, dass man beim Importieren der gesamten Datei, man immer wenn man eine Funktion, etc. ausführen will folgendes eingeben muss.

Syntax

```
dateiname.funktion()
```

Beim importieren einzelner Funktionen kann man sie ganz normal nutzen.

Syntax

```
Funktion()
```

Dabei gibt es noch eine Möglichkeit, alles einzufügen und trotzdem die Funktionen normal zu nutzen.

Syntax

```
from Dateiname import *
```

Das Sternchen steht hierbei, für alles. Das heißt man fügt alles von der jeweiligen Datei ein. Nun kann man alle Funktionen normal nutzen. Damit das bei allen editoren funktioniert muss man dies zuerst erlauben. Das erreicht man indem man eine `__init__.py` erstellt und in dieser dann vollgenden Syntax eingibt.

Syntax

```
__all__ = [„Dateiname“]
```

Das braucht man aber nur für die Module, welche man selber erstellt hat.

Oder man kann auch die Datei ganz normal einfügen und dazu ihr einen kürzel zuweisen.

Syntax

```
import Dateiname as kürzel
```

```
kürzel.funktion()
```

Auch vorgefertigte Module aus dem Internet funktionieren so.

3.15.2 Module und Ordner

Am besten erstellt man einen Unterordner mit allen Modulen, wichtig dabei ist aber, dass es alles .py Dateien sind, denn sonst erkennt Python diese nicht. Man kann Dateien aus einem Ordner auf zwei Arten einfügen. Entweder man importiert einfach alles aus einem Ordner. Hierbei muss man dann nur den Dateinamen und die Funktion angeben.

Syntax

```
from Ordnername import *  
Dateiname.Funktion()
```

Oder man importiert den Ordner ganz einfach, muss dafür extra vor die Datei nochmals den Ordnernamen schreiben.

Syntax

```
import Ordnername  
Ordnername.Dateiname.Funktion()
```

3.15.3 Python Module

Bei Python gibt es Module welche bereits bei der Python installation, mit installiert wurden. Diese kann man alle auf folgender Website finden: <https://docs.python.org/3/py-modindex.html>
 Dabei ist dort wichtig das man oben Links die richtige Version auswählt. Die version findet man mit folgendem Syntax heraus.

Syntax

```
import sys
print(sys.version)
```

3.16 Exceptions

Bei Exceptions handelt es sich Problem Behandler welche unerwartete Probleme lösen. Ein Unerwartetes Problem, wäre zum Beispiel, wenn man Daten von einer Seite beziehen, welche gar nicht mehr existiert. Die einfachste Lösung für solche unerwarteten Probleme ist es den Code so zu Schreiben als würde dieses Problem nicht existieren, jedoch tritt das Problem trotzdem auf, so wird eine sogenannte Exception gemacht.

Wichtig ist das man die Exceptions nicht missbraucht, das heißt man sollte sie nur einsetzen wenn die Fehler wirklich Kritisch sind.

3.16.1 Fehler Lösung mit Exceptions

In Python gibt es viele Möglichkeiten, dass, das Programm nicht funktioniert. Man versucht durch 0 zu dividieren, gibt einen Fehler aus oder man versucht eine Datei zu öffnen welche gar nicht existiert, auch das gibt einen Fehler aus. Das sind eben die sogenannten Exception, das heißt wenn das Programm einen Fehler entdeckt, wird das Programm sofort beendet. Man kann dies auch umgehen mit dem Try Syntax:

Syntax

```
try:
    versuchs Programm
except Fehler:
    print(„Rückmeldung“)
```

Mit dem Try sagt man dem Programm das es den Code zuerst einmal testen soll. Wenn jetzt der bei Fehler angegebene Fehler auftritt, gibt es die eigen geschriebene Rückmeldung zurück. Aber der rest des Codes ist jetzt nicht mehr betroffen und kann ganz normal weiterlaufen.

Dabei gibt es auch einen enormen Vorteil in Funktionen, da man ganze Funktionen in einen Try Syntax einfügen und somit sehr viel mehr schnell und kurz testen.

3.16.2 Eigene Fehlerklassen

Man kann in Python auch eigene Fehler Klassen erstellen. Wenn man jetzt zum Beispiel eine Emaildienst Programmieren will, kann man überprüfen ob es wirklich eine E-Mail ist und wenn nicht eine Fehlermeldung zurückgegeben wird.

Syntax

```
class InvalidEmailError(Exception):
    pass
```

In einem ersten Schritt wird der Error als Klasse definiert, Inhalt benötigt diese Klasse nicht.

Syntax

```
def send_email(email, subject, content):
    if not "@" in email:
        raise InvalidEmailError()
```

Hier wird zuerst die Funktion selber definiert, danach sagt man dem Programm, wenn kein @ in der Email ist soll die Klasse InvalidEmailError ausgeführt werden. Das sorgt dann am Ende dafür das der Fehler ausgegeben wird. In die letzten Klammern der zweiten Syntax kann man noch eine Beschreibung für die Fehlermeldung eingeben. Diesen raise sollte man aber wirklich nur nutzen wenn der Fehler sehr selten auftritt und dieser auch sehr Kritisch ist, sonst kann man ganz normal mit einem Return arbeiten.

4 Der Crawler

In diesem Teil der Doku wird erklärt wie man einen Crawler programmiert. Was ist aber ein Crawler? Ein Crawler ruft eine Seite ab und extrahiert dann die jeweiligen Daten.

4.1 Tools

Jetzt kommen die relevanten Tools:

- Requests:
 - Damit können wir komfortabel an den HTML-Code der Seite kommen.
- BeautifulSoup
 - Damit können wir komfortabel aus dem HTML-Code die Daten extrahieren.

4.1.1 Beispiele

Den Crawler kann man für verschiedenes nutzen, darunter:

- Aktuelle Aktienkurse aus dem Internet holen.
- Überschriften einer Nachrichtenseite in eine csv einfügen.
- Infos aus einem Produktkatalog beziehen.

4.2 Requests

Zuerst das wichtigste, das Requests Modul, müsste eigentlich zuerst installiert werden, jedoch durch die Nutzung von Anaconda, ist dieses Modul vorinstalliert.

Als erstes sollte man das Modul Request importieren, weil es Hauptverantwortlich ist, für die Abfrage der Webseite. Der import sowie die request selber, funktioniert mit folgendem Syntax.

Syntax

```
import requests
r = requests.get(„Internet Adresse“)
```

Nun gibt es von diesem Modul noch eine Funktion welche auch wichtig sind oder einfach nur Infos über die Seite liefern. Das r steht oben für die Variable, welche man in den nachfolgenden Kapiteln auch nutzt, das heißt man kann frei wählen welche man Variable man nutzt, solange man alle verändert.

Status_code

Mit der status_code Funktion kann man den Status der Seite abfragen. Wenn dieser eine 200 ausgibt, so ist alles korrekt, dennoch kann es auch Fehler ausgeben wie 404, was jetzt zum Beispiel bedeutet das die Seite nicht erreichbar ist. Abrufen tut man den Status wie folgt:

Syntax

```
print(r.status_code)
```

headers

Bei dieser Funktion werden alle wichtigen Informationen der Seite gezeigt, wie Server art, Connection, charset oder das Datum. Diese Infos ruft man mit der folgenden Syntax ab:

Syntax

```
print(r.headers)
```

text

Mit dieser Funktion, kann man ganz einfach den gesamten HTML Code, der Webseite anzeigen.

Syntax

```
print(r.text)
```

4.3 Beautifulsoup

Zuerst das wichtigste, das Requests Modul, müsste eigentlich zuerst installiert werden, jedoch durch die Nutzung von Anaconda, ist dieses Modul vorinstalliert.

Beautifulsoup ist für die Zerteilung des HTML Dokuments zuständig. Das ist der Syntax des Imports:

Syntax

```
from bs4 import BeautifulSoup
```

Als erstes Setzt man dem eingefügten html code eine Variable gleich.

Syntax

```
doc = BeautifulSoup(html-code, „html.parser“)
```

```
for p in doc.find_all(“parameter“):
```

Mit dem oberen Befehl sagt man BeautifulSoup, dass die Folgende Seite, der folgende Input untersucht werden soll. Der untere code such für die Variable p alle Eingaben eines bestimmten Parameters, wie p oder h1.

text

Mit der Text Funktion kann man die ausgewählten Tags darstellen.

Syntax

```
print(p.text)
```

Das p ist hierbei die Variabel die oben bereits gesetzt wurde.

attrs

Mit dieser Funktion kann man Attribute strukturiert anzeigen lassen, wie zum Beispiel Klassen.

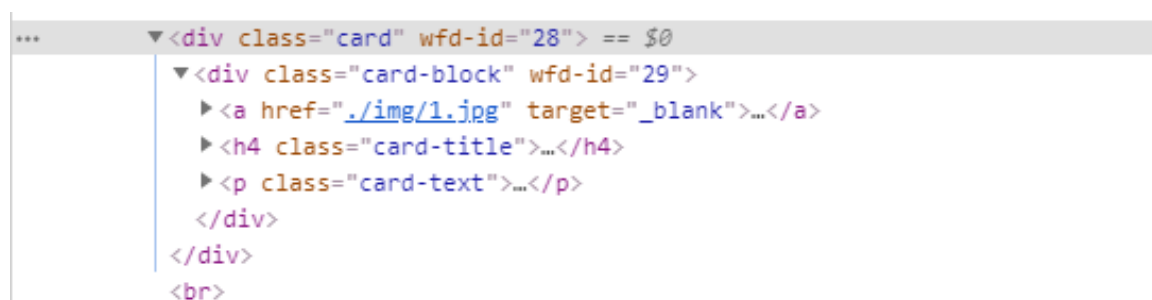
Syntax

```
print(p.attrs)
```

Auch hier steht das p wieder für die oben bereits gesetzte Variable.

4.4 Elemente Finden

Bevor man mit allem Beginnt, sollte man auf die Webseite gehen von welcher man die Daten beziehen will. Dort kann man nun mit Hilfe des Developer Tools heraus finden welchen Tag oder Klasse beziehungsweise Id die wichtigen Daten haben.



Nun kann man ganz normal zurück gehen in den Editor und dort folgenden Syntax eingeben:

Syntax

```
import requests
```

```
from bs4 import BeautifulSoup
```

```
r = requests.get(„Internetadresse“)
```

```
doc = BeautifulSoup(r.text, „html.parser“)
```

```
for card in doc.select(“gesuchter Tag/Class/Id“)
```

Das doc, das r und das card stehen hierbei für Variablen, welcher auch selber gewählt werden können. Jedoch immer alle ändern.

Nachdem man jetzt den Bereich eingesetzt hat, möchte man vielleicht einen bestimmten Tag, class, etc suchen, also:

Syntax

```
print(card.select(„spezifizierter Tag, Class, etc“).text)
```

Durch das .text wird nur der Inhalt des Tags ausgegeben und nicht noch der Tag und die Klasse/Ids.

4.5 Generatoren

Ein sogenannter Generator ist eine Art Schleife, welche einige Vorteile bringt. Um diese zu erklären hilft jedoch nur ein Beispiel:

Liste:

Input
Output

def gen_list():	liste: 0
liste = []	liste: 1
for i in range(0, 6):	liste: 2
print(„liste: „ + str(i))	liste: 3
liste.append(i)	liste: 4
return liste	liste: 5
	for: 0
for element in gen_list():	for: 1
print(„for: „ + str(element))	for: 2
	for: 3
	for: 4
	for: 5

Hier bei der for Schleife mit Liste sieht man das sie zuerst alle Daten(Liste:) durchgeht, diese nebenbei in der Liste gespeichert werden und erst danach die Infos/Liste aus der Schleife ausgibt. Das heißt wenn man einen Crawler hat, und ihm sag man will nur die ersten 8 Titel, durchsucht er dank der Listen zuerst die gesamte Website und stoppt dann sobald er die ersten 8 hat. Negativ daran ist, dass er zuerst sehr viel Zeit aufwendet, die anderen Seiten ein zu lesen.

Generator:

Input	Output
def gen_generator():	gen: 0
for i in range(0, 6):	for: 0
print(“gen: “ + str(element))	gen: 1
yield i	for: 1
	gen: 2
for element in gen_generator():	for: 2
print(“for : “ + str(element))	gen: 3
	for: 3
	gen: 4
	for: 4
	gen: 5
	for: 5

Bei einem Generator werden die Zahlen immer noch ganz normal in Reihe ausgegeben, jedoch wird jede Zahl direkt aus, weitergegeben. Das kann einem Crawler sehr helfen, denn wenn man zum Beispiel sagt man will nur die ersten 8 Titel, arbeitet er sich nur durch die ersten 8 durch und muss nicht zuerst alle Seiten abarbeiten.

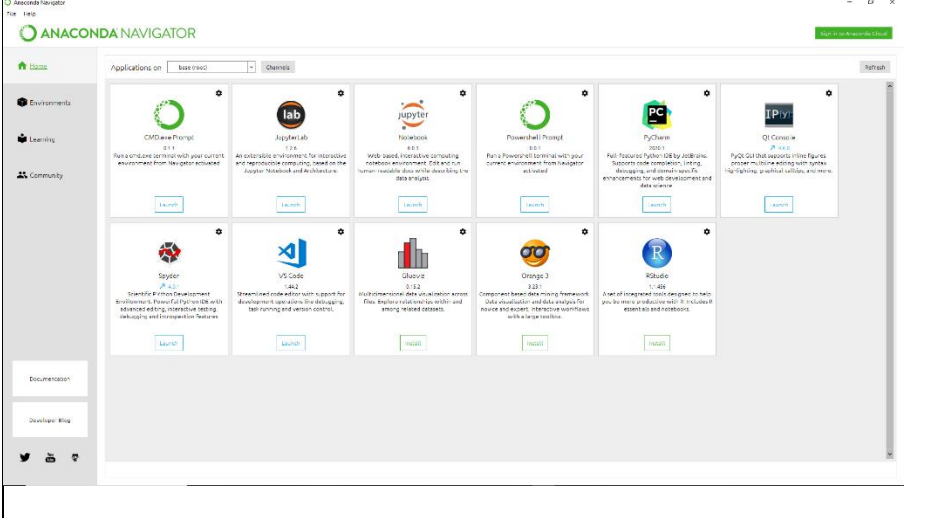
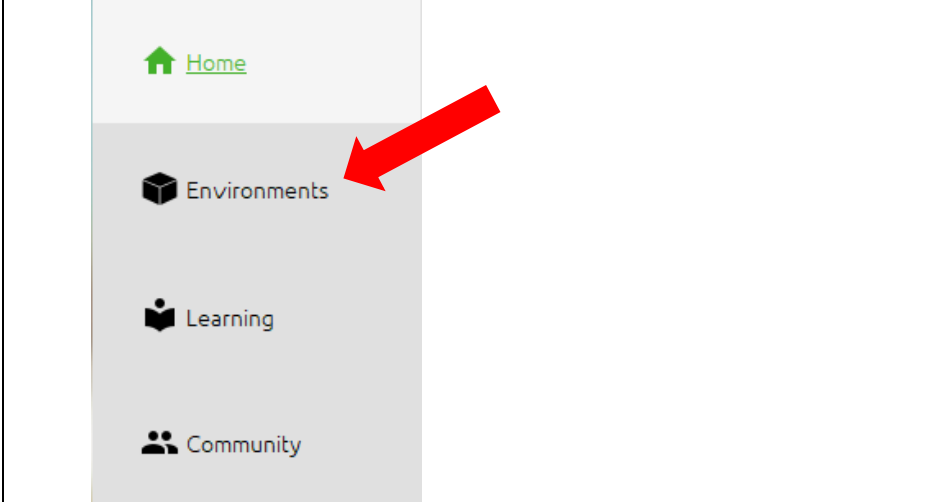
5 Flask

Was ist Flask? Flask ist so gesehen eine Schreibweise von Python mithilfe wesen Python ein Programm so schreiben/designen kann, dass es im Web angezeigt wird. Das heißt mit Hilfe dieses Schreibtyps kann ich einen Chatbot perfekt in die Website einpflanzen. Dieses Flask könnte man ganz normal über jeden Editor erstellen. Ich werde es hier mit dem bereits vorgestellten PyCharm erstellen.

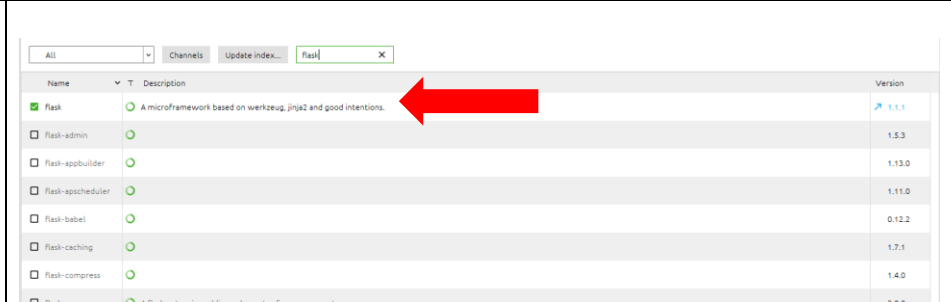
5.1 Installation Flask

5.1.1 Über Anaconda

Für alle welche Anaconda bereits installiert haben ist diese Methode empfehlener, diese geht nämlich schneller und ist einfacher.

Anaconda starten																									
Auf den Environments tab gehen.																									
Beim Feld <i>installed</i> auf <i>all</i> gehen und dann nach Flask suchen.	 <table border="1"> <thead> <tr> <th>Installed</th> <th>Description</th> <th>Version</th> </tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/></td> <td>A microframework based on Werkzeug, Jinja2 and gevent.</td> <td>1.1.1</td> </tr> <tr> <td><input type="checkbox"/></td> <td>admin</td> <td>1.5.3</td> </tr> <tr> <td><input type="checkbox"/></td> <td>flask-appbuilder</td> <td>1.13.0</td> </tr> <tr> <td><input type="checkbox"/></td> <td>flask-appscheduler</td> <td>1.11.0</td> </tr> <tr> <td><input type="checkbox"/></td> <td>flask-babel</td> <td>0.12.2</td> </tr> <tr> <td><input type="checkbox"/></td> <td>flask-caching</td> <td>1.7.1</td> </tr> <tr> <td><input type="checkbox"/></td> <td>flask-compress</td> <td>1.4.0</td> </tr> </tbody> </table>	Installed	Description	Version	<input checked="" type="checkbox"/>	A microframework based on Werkzeug, Jinja2 and gevent.	1.1.1	<input type="checkbox"/>	admin	1.5.3	<input type="checkbox"/>	flask-appbuilder	1.13.0	<input type="checkbox"/>	flask-appscheduler	1.11.0	<input type="checkbox"/>	flask-babel	0.12.2	<input type="checkbox"/>	flask-caching	1.7.1	<input type="checkbox"/>	flask-compress	1.4.0
Installed	Description	Version																							
<input checked="" type="checkbox"/>	A microframework based on Werkzeug, Jinja2 and gevent.	1.1.1																							
<input type="checkbox"/>	admin	1.5.3																							
<input type="checkbox"/>	flask-appbuilder	1.13.0																							
<input type="checkbox"/>	flask-appscheduler	1.11.0																							
<input type="checkbox"/>	flask-babel	0.12.2																							
<input type="checkbox"/>	flask-caching	1.7.1																							
<input type="checkbox"/>	flask-compress	1.4.0																							

Jetzt muss man
nur noch flask und
nur flask
installieren



5.1.2 Über CMD

CMD geht auch nicht sehr schwer jedoch muss man dafür pip installiert haben.

Als erstes öffnet man CMD	
Jetzt gibt den rechts angegebenen Befehl ein.	pip install -U Flask
Nur noch warten bis es fertig installiert ist	

5.2 Neues Projekt

Als erstes erstellt man ein neues Projekt. Diese Erklärung sollte sich im ersten Kapitel bei PyCharm befinden.

5.3 Grundlagen

So nun in diesem Abteil werden wir Grundlegend anschauen was man eingeben muss.

Als erstes fügt man Flask in die Datei mit dem Import Befehl ein.

Syntax
from flask import Flask

Danach fügt man einen Befehl ein welcher eine Flask Applikation erstellt damit der Code durchläuft.

Syntax
app = Flask(__name__)

Als nächstes kommt die Bestimmung des Wegs wenn man den Link eingibt.

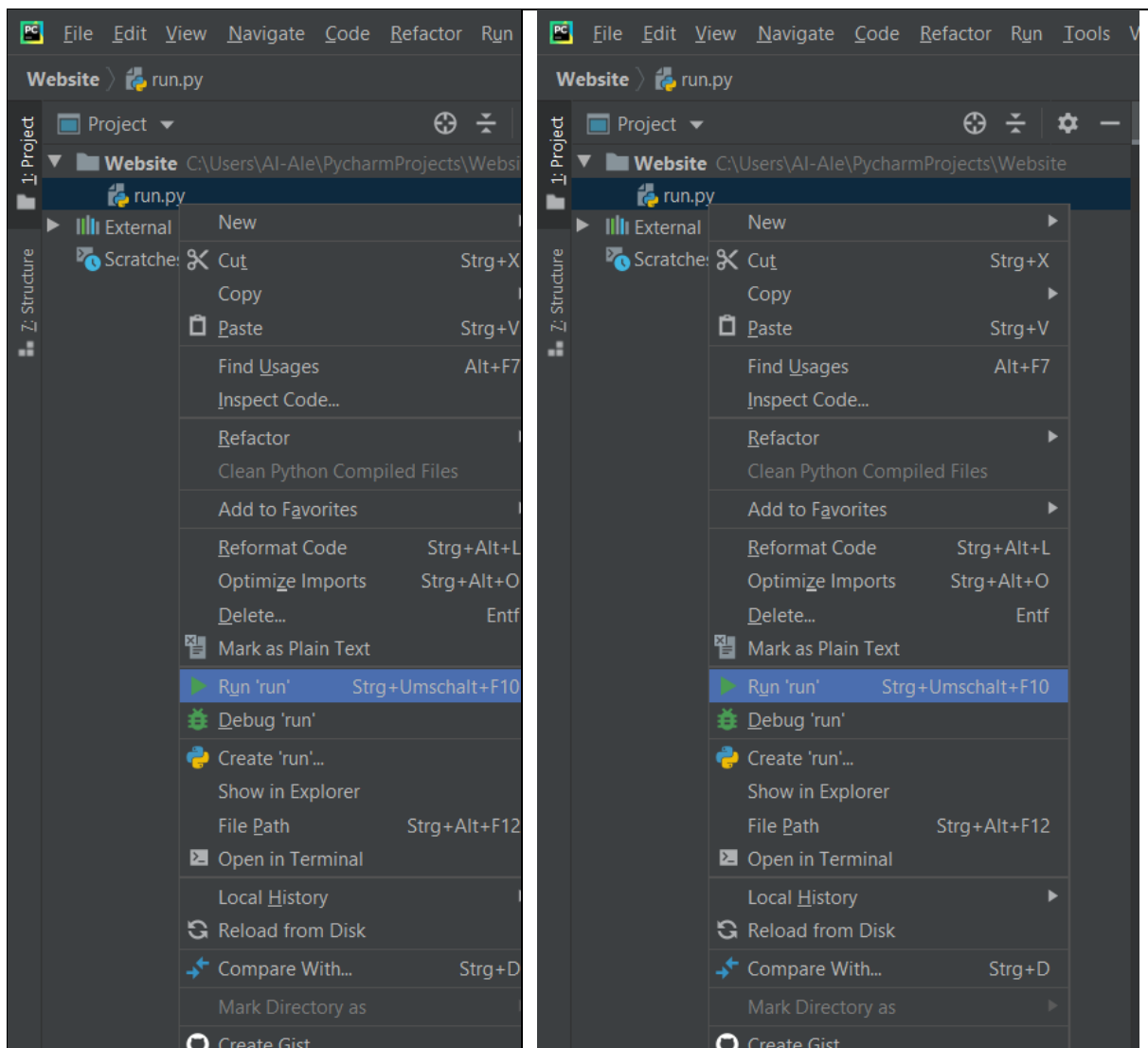
Syntax
app.route(„/“)

Jetzt will man noch zum Beispiel Hello World auf seiner Seite anzeigen, so muss man noch die folgende Funktion erstellen.

Syntax
def funktionsname(): return „Hello World“

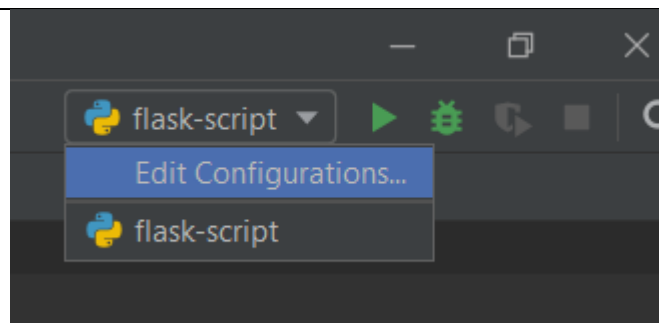
Schon wird auf seiner Seite ein Hello World angezeigt. Wie man diese Seite jetzt richtig aufruft kommt im nächsten Kapitel.

5.4 Webseite aufrufen.

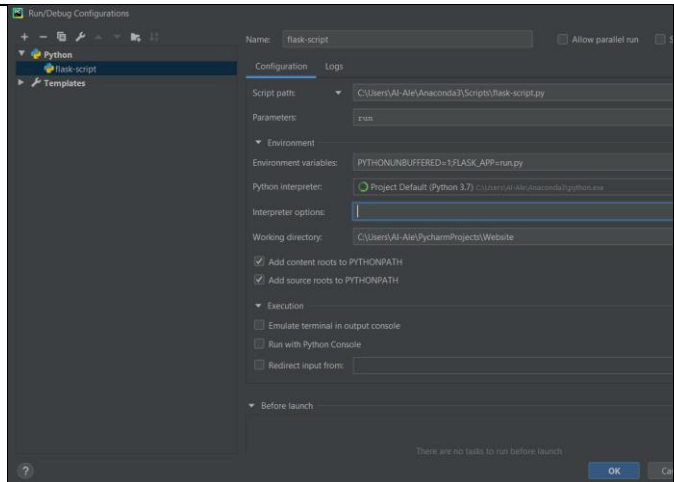


Nun wie ruft man die Webseite auf. Als erste führt man auf die zu ausführende Datei ein Rechtsklick aus und wählt dort den Tab *run* aus.

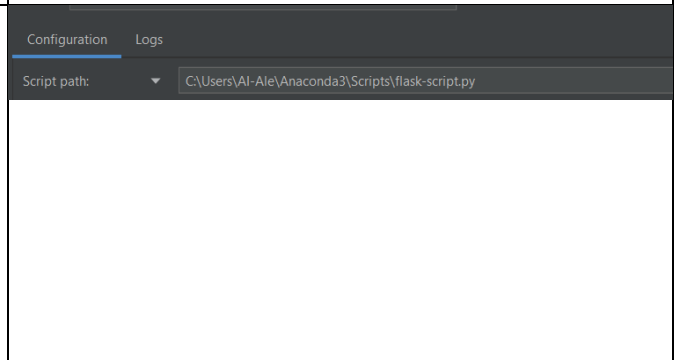
Jetzt sollte man oben rechts einen Tab haben, auf diesen führt man einen Link click aus.



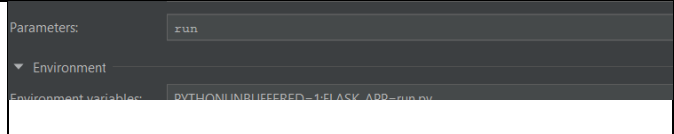
In diesem neu aufgetauchten Feld gibt es jetzt einiges einzustellen.



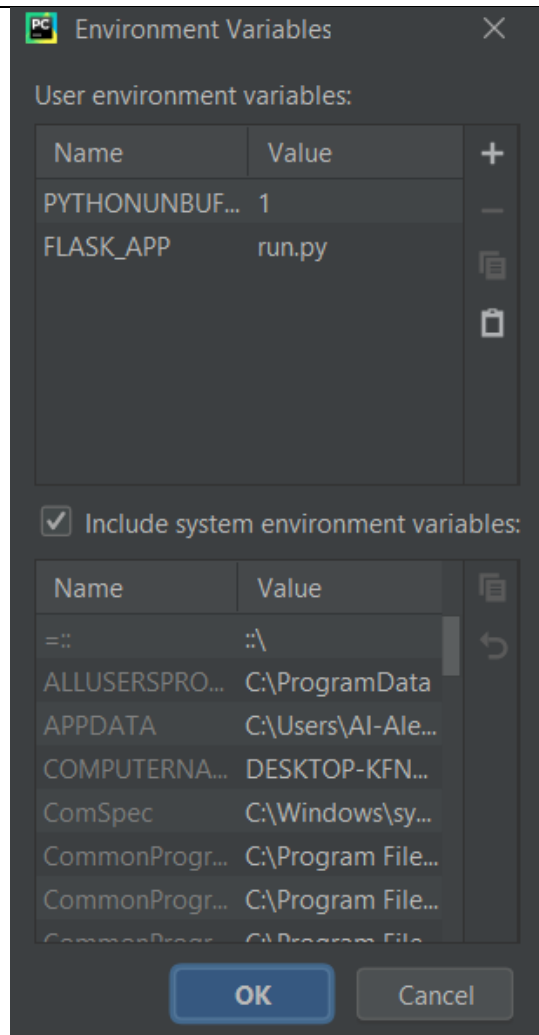
Als erstes wählt man im obersten Feld den Pfad zur Flask Datei welche wir mit Anaconda hinzugefügt haben. Diese Datei findet man am folgenden Ort, dabei ist wichtig dass die drei Punkte am Anfang für den Pfad zum Anaconda Speicherort, welcher jenach Person anders ist:
 .../Anaconda3/Scripts/flask-script.py
 Hierbei ist flask-script.py die gesuchte Datei.



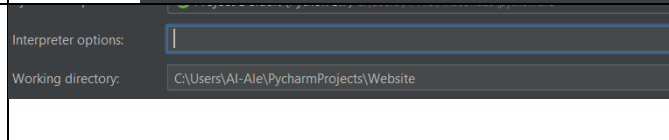
Im Darauf folgenden Feld *Parameters* gibt man einfach *run* ein.



Im nächsten Feld muss man noch eine sehr wichtige Variable eingeben. Als erstes klickt man hierbei mit der Linken Maustaste ganz rechts im Feld auf den Button. Jetzt klickt man im aufgepoppten Feld auf das Plus Zeichen und dort drin gibt man folgendes ein:
 Rechte Seite: FLASK_APP
 Linke Seite: dateiname.py.



Nun muss man nur noch im Feld Working directory den Pfad zum PyCharm Projects und in diesem wählt man dann noch das Projekt aus in welchem man arbeitet.



Nun muss man nur noch oben recht auf das Grüne Dreieck klicken und schon wird einem in der Konsole ein Link gegeben, mit wesen Hilfe man zur Webseite weitergeleitet wird.

5.5 HTML-Code mit Flask generieren

5.5.1 Unterseiten

Mit dem vorher bereits vorgestellten @app... kann man ganz einfach unter Seiten erstellen. Man muss nur den gewünschten Seiten Namen hinter das Slash schreibt.

Syntax

```
@app.route("/")
def hello():
    return „Hello World“

@app.route("/test")
def test():
    return „Hello Test“
```

Wenn man jetzt oben in der suchleiste hinter der Url das folgende aufschreibt gibt es jeweilige Unterschiede:

```
/:
Hello World
```

```
/test:
Hello Test
```

Und schon hat man eine Art unter Seite erstellt.

5.5.2 HTML im return

Man kann auch HTML innerhalb des return Befehles schreiben und zwar muss man nur folgendes machen und der html Code Funktioniert schon. Ich schreibe hier bewusst nicht mit allem, man sollte beachten das der Code nicht funktioniert ohne die Standard Angaben welche man in diesem Kapitel gesehen hat.

Input	Website
return „Hello World“	Hello World

5.5.3 Wichtige Unterordner

In diesem Kapitel geht es darum welche Unterordner wichtig sind. Diese werden in CSS und HTML unterteilt.

CSS

Als erstes erstellt man im Projekt Ordner, denn Unterordner static, für alle wichtigen CSS und damit man sie auch aufrufen kann.

HTML

Als zweites erstellt man einen templates Unterordner, in welchen man nochmal einen layouts Unterordner erstellt. In den templates Ordner kommen die Seiten. In den Layout Ordner kommen alle Layout welche man auf mehreren Seiten braucht, wie Nav-bar oder Footer.

5.6 Flask richtig einbinden

Als erstes befolgt man ganz einfach die regeln von oben und konfiguriert alles richtig in Pycharm damit danach alles funktioniert.

5.6.1 Wichtige Dateien und Inhalte

Nun sollte einfach eine run.py im Projektordner sein. Als erstes fügt man neben dieser noch eine index.html ein. In diese kopiert man dann den folgenden Code. Später ist diese Datei nicht relevant für das Aussehen der Seite, sondern sorgt nur dafür das alles im Hintergrund funktioniert.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Flask - Projekt</title>
  </head>
  <body>
    <div class="container">
      <h1>Überschrift #1</h1>
      <p>
        </p>
      <h2 class="important">Überschrift #2</h2>
      <p>
        </p>
    </div>
  </body>
</html>
```

5.6.2 Start Seite/Chatbot Seite mit Nav-Bar

In diesem Kapitel geht es darum wie man die zwei wichtigsten Seiten grundlegend mit einer Nav-Bar hinzufügt.

Layout

Als erstes erstellt man ein HTML-Dokument innerhalb des Layout Ordners. In diesem geben wir zuerst den folgenden Code ein:

Syntax

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>
    </body>
</html>
```

Nun hat man mal das Grundskelett des HTML-Dokuments.

CSS-Dateien

Nun kommen wir zu den CSS-Dateien, ich nutze hierbei auch ein vorgefertigtes Framework wie Bootstrap und zwar mit Skeleton. Skeleton ist einfach für kleinere Projekte somit, wenn man eine grössere Website macht kann man auch gerne Bootstrap nutzen.

Nun zum Einfügen:

Als erstes geht man auf die Website von Skeleton (<http://getskeleton.com/>) und dort lädt man direkt über den ersten Link Skeleton herunter.

A dead simple, responsive boilerplate.



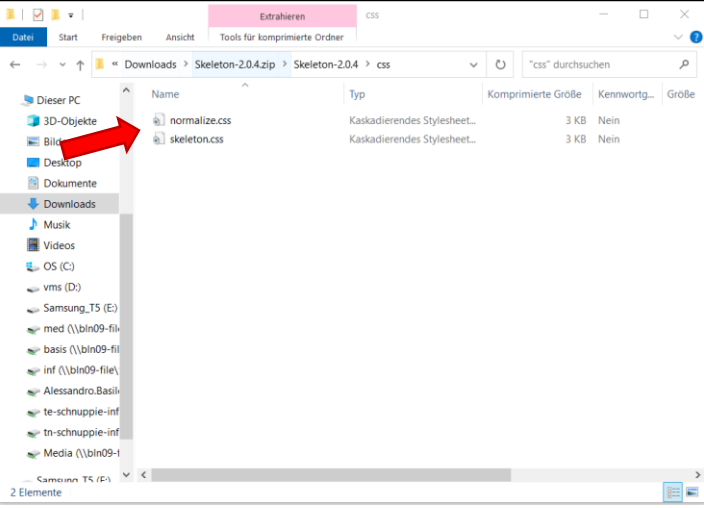
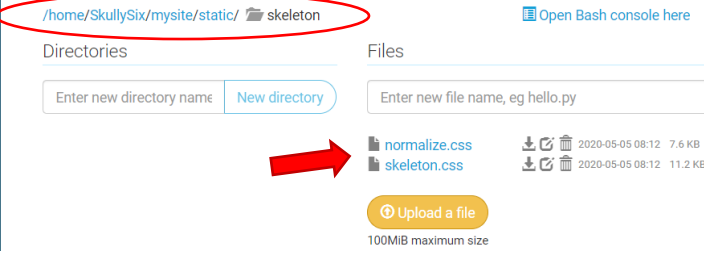
Light as a feather at ~400 lines & built with mobile in mind.



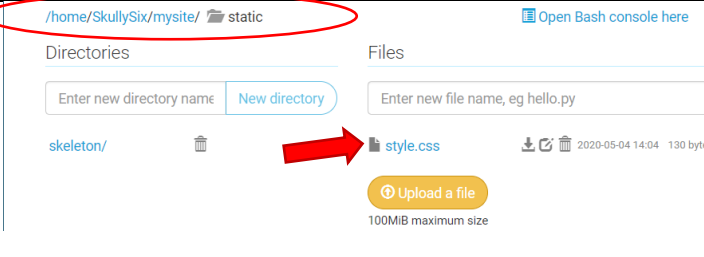
Styles designed to be a starting point, not a UI framework.



Quick to start with zero compiling or installing necessary.

<p>Im ZIP-Ordner befinden sich eigentlich nur zwei relevante Dateien und zwar normalize.css und skeleton.css, welche sich im Ordner CSS befinden. Diese kopiert man sich raus.</p>	
<p>Nun fügt man alles in den static Ordner ein ich habe nur einfach noch einen Extra Skeleton Unterordner erstellt damit es übersichtlicher ist.</p>	
<p>Jetzt muss man nur noch die jeweiligen Dateien in die Layout Datei und in die index Datei einfügen. Den rechts stehenden Code fügt man ganz normal im Header ein.</p>	<pre><link href="static/skeleton/skeleton.css" rel="stylesheet"/> <link href="static/skeleton/skeleton.css" rel="stylesheet"/></pre>

Nun fügt man noch für bestimmte Fälle eine eigene CSS ein.

<p>Als erstes erstellt man eine CSS Datei im static Ordner. Ich habe sie dabei style.css genannt.</p>	
<p>Nun müssen wir noch diese Datei in die Haupt Datei einfügen. Auch dieser Code wird wieder ganz normal im Header eingefügt.</p>	<pre><link href="static/style.css" rel="stylesheet"</pre>

Der einzufügende Code wird später noch erklärt.

Nav-Bar HTML

Jetzt wird der HTML Code gezeigt und erklärt, mit welchem man eine Simple Nav-Bar macht. Der Code wird dabei direkt zuoberst in das Body element gepackt.

<p>Syntax</p> <pre><div class="container"> <ul class="navigation"> <li class="navigation-item"></pre>

<pre> Startseite #Erstellt einen Link für die jeweilige Seite in der Nav-Bar. <li class="navigation-item"> Testseite #Erstellt einen Link für die jeweilige Seite in der Nav-Bar. <div style="clear: both;">&nbsp;</div> #Sorgt für den Abstand zwischen Nav-Bar und Text Anfang. {% block content %} {% endblock %} Mit dieser Hilfe kann man diese Nav-Bar auf jeder Seite nochmal erscheinen lassen. </div> </pre>

Alle Linien mit einem # zu Beginn, sind Erklärungslinien.

Nav-Bar CSS

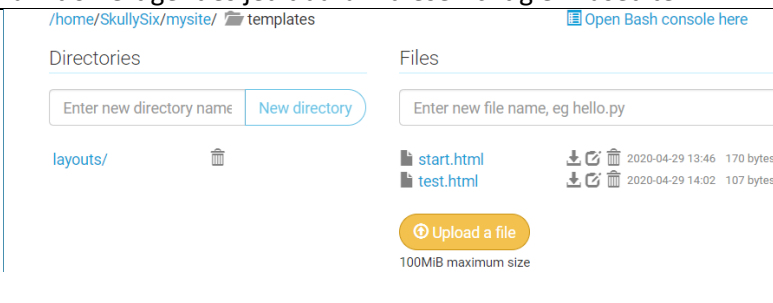
Nun kommen wir zur style.css da jetzt die Klassen navigation und navigation-item noch nicht funktionieren. Jedoch muss man nicht viel machen damit es einigermaßen anschaulich und funktional ist, man kann natürlich aber auch gern es schöner zu machen mit CSS.

Syntax
<pre> .navigation { list-style-type: none; #Sorgt dafür, dass es keinen nervigen oder störenden Listen Zeichen vor der Nav-Bar hat. } .navigation-item { float: left; #Alle Nav-Bar Inhalte gehen von links nach rechts. padding-left: 10px; #Sorgt für Abstand zwischen den einzelnen Nav-Bar Inhalten. padding-right: 10px; #Sorgt für Abstand zwischen den einzelnen Nav-Bar Inhalten. } </pre>

Alle Linien mit einem # zu Beginn, sind Erklärungslinien.

Restliche Seiten

So, da jetzt die Nav-Bar einwandfrei funktioniert geht es jetzt darum diese richtig einzusetzen.

<p>Als erstes erstellen wir im templates Ordner eine neue Seite. Man kann diese gerne nennen wie man will es sollte einfach nur eine HTML Datei sein.</p>	
<p>Jetzt wird rechts der code stehen.</p>	<pre>{% extends „layouts/main.html“ %}</pre>

	<p>#Hierbei verbindet man diese seite mit dem Layout, wichtig ist das jeder seinen Link zur Layout Datei hinterlegen sollte.</p> <p>{% block content %}</p> <p>#Defniert den beginn der Seite</p> <p>Seiten Inhalt</p> <p>{% endblock %}</p> <p>#Definiert dass ende der Seite</p>
--	--

So jetzt wird auf allen Seiten bei welchen man es genau so macht, die gleiche Nav-Bar an wie die, welche man in Layouts erstellt hat.

6 Chatbot erstellen Python

6.1 Natürliche Sprachverarbeitung(NLP)

NLP ist eine Möglichkeit für den Computer, die Menschliche Sprache auf eine vernünftige Art zu analysieren und zu verstehen.

6.2 Natural Language Toolkit(NLTK)

Dies ist eine Plattform für die Erstellung von Python Programmen, um mit einer natürlichen Sprache zu arbeiten. Es hilft vor allem bei der Übersetzung von der Menschlichen Sprache in die Computersprache.

6.3 Installation

Die Installation von NLTK ist sehr einfach man muss grundlegend nur in cmd `pip install nltk` eingeben. Jetzt gibt man im cmd noch `import nltk` ein und danach `nltk.download()`. Jetzt sollte sich ein Bootload öffnen, in diesem drückt man einmal auf *all* und danach auf *Download*.

6.4 NLTK-Textvorverarbeitung

Damit Probleme des Texts durch maschinelle Lernalgorithmen zu lösen, ist ein bestimmter grad an Eigenschaften nötig. Also bevor wir mit NLP starten können sollte man ein wenig vorverarbeiten.

Diese vor Verarbeitung umfasst:

- Konvertieren der Klein- und Großbuchstaben, damit der Algorithmus die gleichen Wörter nicht mehrmals verarbeitet.
- Tokenisierung wird verwendet, um den Prozess der Umwandlung von gewöhnlichen Textzeichenketten in eine Liste von Token, d.h. Wörter, zu beschreiben. Der Vorschlags-Tokenizer wird verwendet, um eine Liste von Vorschlägen zu erstellen. Der Wort-Tokenizer erstellt eine Liste von Wörtern.
- Entfernung von Rauschen, somit alles was keine Zahl oder kein Buchstabe ist.
- Entfernen aller Stoppwörter. Stoppwörter sind die Wörter, welche vom Wörterbuch ausgeschlossen werden da sie für die Bildung der Antwort nicht als wichtig angesehen werden. (Zwischenrufe, Artikel, einige einleitende Wörter)
- Stemming, das Wort zu Wuzelbedeutung zwingen. Wenn man zum Beispiel die Wörter „Stemes“, „Stemming“, „Stammed“ und „Stemization“ hat, wird das Ergebnis ein Wort sein – „Stamm“.
- Dazu braucht man noch die Lemmatisierung, da beim Stemming meist nichtexistierende Wörter hervorbringt, so bringt Lemma existierende Wörter hervor.

6.5 Word-Set

Nach der Vorverarbeitung sollte man den Text in einen Vektor von Zahlen konvertieren. Word-Set besteht aus einem Wörterbuch welches mit berühmten Wörtern und aus den Häufigkeiten , auf die jedes Wort im Text trifft.

Der Sinn hinter Word-Set ist, dass Texte inhaltlich ähnlich sind, wenn sie ähnliche Wörter enthalten. Wenn das Wörterbuch zum Beispiel die Wörter {Learning, is, the, not, great} enthält und wir den Satzvektor «Learning is great» machen wollen, erhalten wir den Vektor(1, 1, 1, 0, 0, 0, 1).

6.6 TF-IDF-Verfahren

Word-Set hat leider ein Problem, und zwar, dass der Text von häufig vorkommenden Wörtern dominiert werden kann, welche keine wichtigen oder wertvollen Informationen hat. Somit gibt auch «Word-Set» den langen texten mehr Bedeutung als den kurzen Texten.

Ein Ansatz zu Lösung dieses Problems besteht darin, die Häufigkeit des Wortes nicht in einem Text, sondern auf einmal zu berechnen. Somit wird der Beitrag z.B der Artikel «a» und «the ausgeglichen. Dieser Ansatz wird als TF-IDF (Term Frequency-Inverse Document Frequency) bezeichnet und besteht aus zwei Schritten.

- **TF** – Berechnung der Worthäufigkeit in einem Text.

- **IDF** – Berechnung, wie selten ein Wort in allen Texten vorkommt.

Beispiel

Zum Beispiel, man einen Text mit 100 Wörtern, in diesem kommt 5-mal das Wort „Telefon“ vor. Der TF-Parameter für das Wort „Telefon“ ist $(5/100) = 0,05$.

Nun stellen wir uns vor, wenn wir 10 Millionen Dokumente haben, und das Wort „Telefon“ erscheint in 1000 von ihnen. Der Koeffizient wird berechnet als $1 + \log(10\,000\,000/1\,000) = 4$. Somit ist der TD-IDF $0,05 * 4 = 0,20$.

TF-IDF kann in *scikit* so implementiert werden:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

6.7 Otiai-Koeffizient

TF-IDF ist eine Transformation die auf Texte angewendet wird, um zwei reale Vektoren in einen Vektorraum zu erhalten. Dann können wir denn Otiai-Koeffizienten eines beliebigen Vektorenpaars erhalten, indem wir ihr elementare Produkt berechnen und es in das Produkt ihrer Normen unterteilen. Somit wird der Winkelsinus zwischen den Vektoren erhalten. Der Otiai-Koeffizient ist ein Maß für die Ähnlichkeit zwischen zwei Nicht-Null-Vektoren. Mit dieser Formel können wir die Ähnlichkeit zwischen zwei beliebigen Texten d1 und d2 berechnen.

Beispiel

Cosine Similarity (d1, d2) = $\text{Dot product}(d1, d2) / ||d1|| * ||d2||$

6.8 Chatbot-Training

Als erstes nimmt man sich einen großen Text. Ich habe jetzt zum Beispiel die Wikipedia Seite von Chatbots genommen und den Inhalt in eine .txt Datei importiert.

Import der notwendigen Bibliotheken

```
import nltk
import numpy as np
import random
import string # to process standard python strings
```

6.9 Daten Lesen

Lesen Sie die Datei corpus.txt und konvertieren Sie den gesamten Text in eine Satzliste und eine Wortliste zur Weiterverarbeitung.

```
f=open('chatbot.txt','r',errors = 'ignore')

raw=f.read()

raw=raw.lower()# converts to lowercase

nltk.download('punkt') # first-time use only

nltk.download('wordnet') # first-time use only

sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences

word_tokens = nltk.word_tokenize(raw)# converts to list of words
```

Betrachten wir ein Beispiel für `sent_tokens` und `word_tokens` Dateien.

```
sent_tokens[:2]
['a chatbot (also known as a talkbot, chatterbot, bot, im bot, interactive agent, or artificial
conversational entity) is a computer program or an artificial intelligence which conducts a
conversation via auditory or textual methods.',
'such programs are often designed to convincingly simulate how a human would behave as a
conversational partner, thereby passing the turing test.']

word_tokens[:2]
['a', 'chatbot', '(', 'also', 'known']
```

6.10 Quellvorverarbeitung

Jetzt definieren wir die `LemTokens`-Funktion, die Token als Eingabeparameter übernimmt und normalisierte Token erzeugt.

```
lemmer = nltk.stem.WordNetLemmatizer()
#WordNet is a semantically-oriented dictionary of English included in NLTK.

def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

6.11 Keyword-Auswahl

Lassen Sie uns die Begrüßungslinie des Bots bestimmen. Wenn der Benutzer den Bot begrüßt, sagt der Bot hallo zurück. ELIZA verwendet einen einfachen Keyword-Vergleich für Begrüßungen. Wir werden die gleiche Idee verwenden.

```
GREETING_INPUTS = ("hallo", "hi", "hey", "morgen", "abend", "grüezi")

GREETING_RESPONSES = ["hi", "hey", "*nickt*", "hallo", "Ich bin sehr froh das du mit mir redest."]

def greeting(sentence):

    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
```

6.12 Antwortgenerierung

Um die Antwort unseres Bots auf die Eingabe von Fragen zu generieren, wird das Konzept der Ähnlichkeit verwendet. Deshalb importieren wir zunächst die notwendigen Module.

- Importieren Sie den TFidf-Vektorisierer aus der Bibliothek, um einen Satz von Rohdaten in eine TF-IDF-Eigenschaftsmatrix zu konvertieren.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

- Importieren Sie auch das Otiai-Koeffizientenmodul aus der scikit-Bibliothek. (Bitte ganz oben bei den anderen Import hinzufügen.)

```
from sklearn.metrics.pairwise import cosine_similarity
```

Dieses Modul wird verwendet, um nach Schlüsselwörtern in der Abfrage des Benutzers zu suchen.

Dies ist der einfachste Weg, um einen Chatbot zu erstellen.

Definieren Sie eine **Antwortfunktion**, die eine von mehreren möglichen Antworten zurückgibt. Wenn die Anfrage mit keinem der Keywords übereinstimmt, antwortet der Bot mit „Es tut mir leid aber, ich verstehe dich nicht.“

```
def response(user_response):
    robo_response=""

    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]

    if(req_tfidf==0):
        robo_response=robo_response+"Es tut mir leid aber ich verstehe sie nicht."
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

Schließlich spezifizieren wir Bot-Antworten am Anfang und Ende der Koversation, abhängig von den Antworten des Benutzers.

```

flag=True
print("Oliver: Mein Name ist Oliver. Ich werde dir so gut wie möglich bei deinen Fragen helfen. Wenn du
mich verlassen willst dann gib bitte bye ein!")

while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='danke' or user_response=='danke dir' ):
            flag=False
            print("Oliver: Bitte..")
        else:
            if(greeting(user_response)!=None):
                print("Oliver: "+greeting(user_response))
            else:
                sent_tokens.append(user_response)
                word_tokens=word_tokens+nlk.word_tokenize(user_response)
                final_words=list(set(word_tokens))
                print("Oliver: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False

print("ROBO: Auf wiedersehen...")

```

Wir haben es geschafft. Obwohl unser primitiver Bot kaum kognitive Fähigkeiten hat, war es eine gute Möglichkeit, mit NLP umzugehen und mehr über Chatbots zu erfahren.

6.13 Erste Fragen einbinden (ohne KI)

Da ich derzeit noch auf der Such bin nach einem Deutschen Einbindung für eine KI finde ich das ein nicht vollkommen KI funktionierender Chatbot eine gute Lösung ist.

6.13.1 Lehrstellen Fragen antworten

Als erstes geben wir möglich Inputs des Nutzers an in diesem Fall wäre das zum Beispiel Lehrstelle oder Lehrstellen. Diesen input sollte man alles klein schreiben da der Chatbot auch alle eingaben als klein ansieht.

```
apprentice_INPUTS = („lehrstelle“, „lehrstellen“)
```

Danach geben wir die Antworten an, die der Chatbot geben soll dabei kann man auch mehrere angeben. Auch sollte man alles klein schreiben.

```

apprentice_RESPONSES = [„Hier in der Wibilea haben wir die folgenden Lehrstellen: Informatiker,
Polymechaniker und Automatiker.“, "Die Wibilea bietet Lehrstellen im Bereich Automatik,
Informatik und Polymechanik.“]

```

Daraufhin erstellt man wieder die gleiche funktion wie vorhin bei den greetings. Bei dieser sagt man dem Bot einfach er soll alles klein schreiben und ein per Zufall gewählte Antwort nehmen.

```

def apprentice(sentence)

    For word in sentence.split()
        If word.lower() in apprentice_INPUTS:
            Return random.choice(apprentice_RESPONSES)

```

Wenn der Nutzer jetzt eine Frage eingibt in welcher das Wort Lehrstelle vorkommt, gibt der Bot eine der unter apprentice_RESPONSES angegebenen Antworten aus.

Als letztes ist jetzt noch wichtig die if Abfrage am Schluss des Codes einzufügen damit der Bot auch weiß wann er zum Einsatz kommt.

```
elif (apprentice(user_response) != None):  
    print("Oliver: " + apprentice(user_response))
```

Dieses elif muss unmittelbar unterhalb des if sein in welchem sich das greeting befindet. Nach diesem kann man noch so viele elif einfügen wie man will, solange man nie die gleichen über Titel nimmt (bsp. Apprentice).

Das kann man jetzt auch so ändern wie man will.

6.14 Wieso andere Sprache einbinden

Da der Bot auf einer Englischen Seite erklärt wurde hat er auch nur Englische Stop-Wörter, diese sind sehr wichtig da dank ihnen die Logik des Satzes besser analysiert werden kann und man somit eigentlich auch ein ganze Website einbinden kann. Mithilfe dieser Website kann man den Bot noch viel Intelligenter machen.

7 Testfälle

7.1 Testfall 1

Beschreibung	Start des Programms
Testvoraussetzung	PyCharm mit der Version 2020.1.1 installiert, Chatbot Datei
Testschritte	<ol style="list-style-type: none"> 1. Erstellen sie ein Neues PyCharm Projekt. 2. Innerhalb des Projekts öffnen sie die Chatbot.py Datei. 3. Führen sie jetzt einen rechts Klick auf die jetzt neu erschienene Chatbot.py Datei aus. 4. Jetzt klicken sie mit der Linken Maustaste auf den Tab Run welcher noch ein kleine Grünes Dreieck vorne dran hat.
Erwartetes Ergebnis	<p>Es sollte der folgende Text erscheinen:</p> <pre>C:\Users\AI-Ale\Anaconda3\python.exe C:/Users/AI-Ale/PycharmProjects/Chatbot/Chatbot0.1.py [nltk_data] Downloading package punkt to C:\Users\AI- [nltk_data] Ale\AppData\Roaming\nltk_data... [nltk_data] Package punkt is already up-to-date! [nltk_data] Downloading package wordnet to C:\Users\AI- [nltk_data] Ale\AppData\Roaming\nltk_data... [nltk_data] Package wordnet is already up-to-date! liste 1 liste 2 Oliver: Meine Name ist Oliver. Wenn du irgendwelche Fragen über die Wibilea, etc hast, beantworte ich dir diese gerne. Wenn du raus gehen willst gib bitte bye ein.</pre> <p>(An den Stellen von liste 1 und liste 2 sollte wirklich zwei Python Listen erscheinen, welche sich aber immer wieder ändern.</p>
Testresultat	

7.2 Testfall 1

Beschreibung	Vorprogrammierte Antwort „Begrüssung“
Testvoraussetzung	PyCharm mit der Version 2020.1.1 installiert, Chabot Datei
Testschritte	<ol style="list-style-type: none"> 1. Nach dem oberen Test sollte das Programm ja bereits laufen, wenn nicht, starten sie das Programm nochmal wie oben beschrieben. 2. Geben sie nun unterhalb von „Oliver: Meine Name ist Oliver. Wenn du irgendwelche Fragen über die Wibilea, etc hast, beantworte ich dir diese gerne. Wenn du raus gehen willst gib bitte bye ein.“, Hallo ein. 3. Geben sie jetzt nach und nach die folgenden Wörter einzeln ein: „hello, hi, greetings, sup, what's up, hey, hallo, moin, guten tag, hoi, guten abend, guten mittag“
Erwartetes Ergebnis	Jetzt sollte der Chatbot bei jedem einzelnen Wort, eine Antwort ausgeben. Es sollte aber immer eine der folgenden antworten sein: „Hi, Hey, *nickt*, Hallo, Ich bin froh das du mit mir redest“
Testresultat	

7.3

7.4

8 Arbeitsjournal

08.01.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Informations Video über KI angeschaut.	Erledigt	Wichtige Infos über Neuronale Netze	Keine	Video
Selber umgesetzt.	Erledigt	Einfachheit sowie Komplexität kennengelernt.	Einsetzung von Tensorflow funktionierte nicht ganz wegen Update.	
Mustererkennung einer KI	Erledigt			Video

13.01.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Mustererkennung verfeinert für ein fast perfektes Ergebnis.	Erledigt	Genauere Trainieren von KI.	Tensorflow, gleiches wie am 08.01.	
Python kennen gelernt und alles wichtige genauer angeschaut.	Erledigt	Python besser verstehen. Logik verständlicher.	Keine	Video

04-07.02.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
KI Verbessern	Eigenes Fenster Eigene Muster	Fenster öffnet sich	KI ist um einiges Komplizierter als gedacht	

10.02.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
KI Verbessern	Eigenes Fenster Eigene Muster	Fenster öffnet sich	KI ist um einiges Komplizierter als gedacht	

17.02.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
KI Verbessern	Eigenes Fenster Eigene Muster	Fenster öffnet sich	KI ist um einiges Komplizierter als gedacht	

02.03.2019

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
-----------	--------------------------------------	-----------------------	-----------------------	------------------

KI verändert, neue gadgets hinzufügen	Rundungs Größe anpassen alles genauer gemacht	Runden eigen eingeben	-	
Von Ubuntu zurück zu Windows	Gewechselt	-	Ubuntu zu alt für meine Infos	

09.03.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Tkinter lernen	Eigenes Fenster erstellen, KI fast ganz eingebunden	Tkinter wissen verbessert	-	Video
Aus Python file Exe gemacht	Python als Normales Programm starten	Wie man alle Python files in exe Dateien umwandelt	Schwer in HTML einbinden, vielleicht umwandeln ins Java/Java Script wegen Chatbot	

30.03.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Vorprogrammierter Chatbot	-Chatbot grundlegend begonnen und schon sehr weit	-Programmierung eines vorprogrammierten Chatbots.	-	Ai-United

31.03.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
-----------	--------------------------------------	-----------------------	-----------------------	------------------

Vorprogrammierter Chatbot	-Chatbot, funktioniert	-Fertigung und Verbesserung eines Chatbots	-Er kann bisher nur vorprogrammierte Antworten verwenden.	Ai-United
Udemy	-Udemy Kurs begonnen	-Kenntnisse über Grundlagen verbessert.		Udemy

01.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-Kontrollkonstrukte gelernt sowie Funktionen begonnen		Udemy

06.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-Funktionen abgeschlossen und Listen in Python begonnen		Udemy
Python to Exe	-Informationen über Pyinstaller und Auto-py-to-exe gesammelt	-Beste Python to Exe Möglichkeiten wären, Pyinstaller und Auto-py-to-exe.	-	PyPi Dataofish

07.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-Listen in Python abgeschlossen und Objektorientierung begonnen.	-	Udemy
Python to exe	-Pyinstaller und Auto-py-to-exe getestet	-Die Umwandlung funktioniert in bestimmten Gebieten mit Pyinstaller.	-Wegen eines bestimmten Moduls innerhalb des Chatbots funktioniert die Umwandlung nicht. -Wegen zu diesem Zeitpunkt unbekannten Version Problemen funktioniert die Installation von Auto-py-to-exe gar nicht	

08.04.2019

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-Objektorientierung sowie Module in Python abgeschlossen		Udemy

Python to exe	-Weitere Versuche im gebiet	-	-Weitere Fehlschläge	
---------------	-----------------------------	---	----------------------	--

14.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-Crawler begonnen		Udemy
Python to Web	-Im Internet, etc . nach Informationen gesucht über die Einführung von Python ins Web.		Keine derzeitige Möglichkeit Python direkt in HTML zu schreiben, da Python Server-Seitig ist und HTML Clientseitig	Udemy PSD-Tutorials

15.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-Crawler nun verstanden und Abgeschlossen.		Udemy

Python to Web	-Weiter Recherchen über Python to Web.	-Es wäre möglich jedoch nur über ein Plugin für Apache und dazu wäre diese Methode sehr veraltet	-Zum jetzigen Zeitpunkt keine ersichtliche Lösung	
---------------	--	--	---	--

16.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-Exceptions und Datenstrukturen. Abgeschlossen		Udemy

17.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy	-Udemy Kurs fortgesetzt	-PyCharm angeschaut		Udemy
Python to Web	-Dank neuer Webseiten von Herr Fructuoso, Fortschritte im Bereich Python to Web-	Enorme Fortschritte auf Wissen über Python to Web bezogen.		

27.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Python Anywhere	-Erste versuche und tests auf Python Anywhere mit Flask	-Flask entdeckt		Python Anywhere

28.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Flask in Python Anywhere besser kennen gelernt	-Eingabefelder erstellt -Weiteres im Bereich Flask	-Mehr Möglichkeiten für Flask		Python Anywhere
Udemy Kurs für Flask	-Anfangen mit einem Kurs über Flask	-Flask möglichkeiten		Udemy

29.04.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
-----------	--------------------------------------	-----------------------	-----------------------	------------------

Udemy Kurs für Flask	-Den Flask Udemy Kurs weitergemacht	-Flask möglichkeiten		Udemy
----------------------	-------------------------------------	----------------------	--	-----------------------

04.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Udemy Kurs für Flask	-Flask Kurs mehr angeschaut um den Chatbot richtig einzubinden	-Flask möglichkeiten		Udemy

05.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Neu gelerntes Testen	-Neu Gelerntes aus dem Kurs, für Chatbot ausprobiert	-Chatbot erstamlt einigermaßen eingebaut		Udemy

06.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
-----------	--------------------------------------	-----------------------	-----------------------	------------------

Letzte Python to Web versuche	-Chatbot versuche endlich ihn einzubinden	-	Chatbot sowie Flask laufen Parallel und geben/nehmen keine Informationen.	Python Anywhere
-------------------------------	--	---	---	---------------------------------

11.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Neue Versuche auf Chatbot bezogen	-Versuche, das Deutsche Stopwords akzeptiert werden.	-	Die Liste der Stop Wörter	

12.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Weitere Versuche	-Chatbot Stop Words liste endlich einfügen	-	Es gibt einige Tips doch die meisten funktionieren wegen der Version oder meinen Modulen nicht.	

13.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Letzte Python Versuche für Stop Word liste	-Endlich Stop Word list eingefügt	-Eigene Stop Wörter einfügen	-	
Tokenizer ins Deutsche übersetzen	-Das Programm welches die Schlüsselwörter aus dem Text pickt auf Deutsch übersetzten	-Tokenizer für die deutsche Sprache wappnen.	Auch wenn alles Funktionieren sollte, tut das Programm bei den Schlüssel und Stop Wörtern immer noch blöd.	

18.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Dokumentation überarbeiten	-Dokumentation überprüfen -Dokumentation verbessern -Wichtige Informationen noch hinzufügen			

19.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Dokumentation überarbeiten	-Dokumentation überprüfen -Dokumentation verbessern -Wichtige Informationen noch hinzufügen	-		

20.05.2020

Tätigkeit	Erledigte Arbeiten / Erreichte Ziele	Erfolge & neu gelernt	Aufgetretene Probleme	Genutzte Quellen
Dokumentation überarbeiten	<ul style="list-style-type: none"> -Dokumentation überprüfen -Dokumentation verbessern -Wichtige Informationen noch hinzufügen 	-		

9 Tabelle

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

10 Abbildungsverzeichnis

Abbildung 1- TEST..... Fehler! Textmarke nicht definiert.

11 Index

.

.close() · 29
 .join() · 19
 .split() · 19, 28

__Eigenschaft · 32
 __init__ · 31, 33, 35

A

Algorithmus · 50
 Anaconda · 2, 4, 8, 9, 10, 11, 37, 40, 43
 And · 23
 app.route(„/“) · 41, 44
 append() · 18, 20, 29, 30, 31, 39, 54

B

Beautifulsoup · 4, 37
 Beispiel · 71
 Bool · 17
 Booleschen Werten · 22, 23
 Break · 3, 27

C

camelCase · 34
 Chatbot · 1, 2, 4, 6, 7, 40, 46, 50, 51, 53, 54, 59, 60, 65, 66, 67
 Comments · 16
 Complex · 17, 18
 Comprehension · 2, 19, 20
 Constructor · 3, 31, 32
 Continue · 3, 27
 Crawler · 4, 37, 39, 62, 63
 CSS · 10, 45, 46, 47, 48

D

Datentypen · 2, 16, 17
 Developer Tools · 38
 Dictionaries · 2, 20, 21

E

Editor · 8, 9, 10, 38, 40
 Elif · 22, 26, 55
 Else · 2, 22, 23, 25, 26, 28, 53, 54
 10.02.2020 16:00:00

Exceptions · 3, 36, 63

F

Fehlerklassen · 3, 36
 Flask · 4, 10, 40, 41, 43, 44, 45, 64, 65, 66
 FLASK_APP · 44
 float · 16, 17, 18, 48
 For Schleife · 26, 39
 Funktionen · 3, 27, 28, 34, 35, 36, 60

G

Generator · 4, 18, 38, 39
 Grafiken · 3, 29

H

headers · 37
 HTML · 4, 10, 37, 44, 45, 46, 47, 48, 59, 62

I

IDF · 4, 50, 51, 52
 If · 2, 22, 23, 24, 25, 26, 27, 28, 30, 34, 36, 52, 53, 54, 55
 Index · 5, 19, 73
 in-Operator · 24, 25
 int · 16, 17, 30
 items · 2, 21

J

JavaScript · 7

K

KI · 4, 54, 57, 58, 59
 Klassen · 17, 31, 32, 33, 36, 38

L

LemTokens-Funktion · 52
 len() · 18
 List · 2, 17, 19, 20
 Listen · 2, 18, 19, 20, 21, 27, 39, 48, 60, 61

M

matplotlib · 29, 30
 Methoden · 3, 6, 31, 32

Module · 3, 10, 29, 34, 35, 36, 52, 62

N

n · 3, 28
Nav-Bar · 4, 46, 47, 48, 49
NLP · 4, 50, 54
NLTK · 4, 50, 52
not-Operator · 2, 25

O

Objektorientierung · 3, 31, 61, 62
Operatoren · 18
Or · 23
Otiai-Koeffizienten · 4, 51

P

Parameter · 19, 28, 29, 51
PascalCase · 34
plt.plot() · 29, 30
plt.show() · 29, 30
pop() · 19
print() · 16, 27
PyCharm · 2, 10, 11, 12, 13, 40, 41, 44, 63
Python · 7

R

Random · 18, 75
remove() · 19
Requests · 4, 37
return · 3, 4, 28, 32, 33, 34, 39, 41, 44, 45, 52, 53

S

sent_tokens · 51, 52, 53, 54
Skeleton · 46, 47
Slicing · 19
sneak_case · 34
Sprachverarbeitung · 4, 50
Spyder · 2, 8, 9, 10, 11
Status_code · 37
Stemming · 50
Stoppwörter · 50
str · 16, 17, 32, 39

T

TF · 4, 50, 51, 52
TFidf-Vektorisierer · 52
Token · 50, 52
Tokenisierung · 50
Tuple · 2, 17, 20, 21

V

Variablen · 2, 3, 16, 31, 34, 35, 38
Vektor · 50
Vererbung · 3, 33
Vergleichsoperatoren · 2, 23

W

While Schleife · 26
With Konstrukt · 3, 29
word_tokens · 51, 52, 54
Word-Set · 4, 50

12 Glossar

Abkürzung

Ram

edim

Random Access Memory