Hier könnt ihr eure Codesnippets austauschen.

WICHTIG: dieses Pad ist ÖFFENTLICH, jeder kann es einsehen und verändern!! Einmal geschrieben bleibt alles verfügbar, siehe Zeitleiste und "Gespeicherte Versionen".

Ausserdem handelt es sich um eine temporäre Lösung, d.h. kopiert alles wesentliche, damit ihr es später verwenden könnt.

Trennt am besten die einzelnen Blöcke ab:

```
import re
import urllib.request
import html
import csv

def get_website_content(url):
    page = urllib.request.urlopen(url)      # open URL
    page = page.read()                      # read URL source code
    return page.decode("utf-8")             # decode as Unicode text

def get_info_from_website_source(website_text):
    # TODO: extract title, url and first paragraph from website source code
    title = "title"
    url = "url"
    first_paragraph = "text"
    return ([url, title, first_paragraph])

def write_data_to_file(writer, data):
    writer.writerow(data)
    return

def make_corpus():
    with open('corpus.csv', 'w', newline='') as csvfile:
        corpuswriter = csv.writer(csvfile, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        corpuswriter.writerow(["URL", "Title", "First Paragraph"])
        for i in range(100):
            p = html.unescape(get_website_content("https://en.wikipedia.org/wiki/Special:Random"))
            #print(p)
            data = get_info_from_website_source(p)
            write_data_to_file(corpuswriter, data)
```

Automatisches Texten Beispielergebnisse (von Esther):
http://www.enigmabrot.de/new_dickens.txt
http://www.enigmabrot.de/new_buddenbrooks.txt

```
import re
```

```
"""
Liest eine Datei ein und schreibt den Inhalt der Zeilen in eine Liste
"""
def fileimport(filename):
    datei = open(filename, "r", encoding="utf8")
    ergebnis = []
    for line in datei:
        #Entfernen von newlines
        line = line.strip()
        ergebnis.append(line)
    return ergebnis

"""
Sucht ein Wort in einer Liste und gibt die Konkordanzen anhand der beiden Grenzen aus
"""
def do_work(word, liste, grenze1, grenze2):
    #Sucht matches in der Liste und schreibt die Indizes in eine neue Liste
    indexes = [i for i,x in enumerate(liste) if x.lower() == word.lower()]
    results=[]
    if indexes != []:
        for index in indexes:
            minus = grenze1
            plus = 0
            sub_list =[]
            #Schreibt jedes Wort links des Fundes bis zum angegebenen Ende in Subliste
            while minus!=0:
                #Benutze keine negativen Indizes
                if index-minus>0:
                    sub_list.append(liste[index-minus])
                minus-=1
            #Schreibt jedes Wort rechts des Fundes bis zum angegebenen Ende in Subliste
            while plus<=grenze2:
                #Gehe nicht Ã¼ber das letzte Element der Liste hinaus
                if index+plus<len(liste):
                    sub_list.append(liste[index+plus])
                plus+=1
            results.append(sub_list)
    #Gebe Meldung aus, falls Wort nicht in Datei vorhanden
    else:
        print ("Dieses Wort wurde nicht in der Datei gefunden")
    #Return ist eine Liste mit Konkordanz-Sublisten
    return results

if __name__ =="__main__":
    checked_string = "y"
    #Fuehre Programm immer wieder aus, solange gewÃ¼nscht
    while checked_string == "y":
        #Fragt immer wieder nach Dateinamen, bis korrektes Muster eingegeben wurde
        while True:
            try:
                filename = input("Geben Sie einen Dateinamen ein: ")
                arbeitsliste = fileimport(filename)
```

```
            break
        except FileNotFoundError:
            print ("Diese Datei existiert nicht. Versuchen Sie es erneut")

    word = input("Geben Sie ein Wort ein nach dessem Kontext gesucht werden soll: ")
    lgrenze = int(input("Geben Sie die linke Grenze der Konkordanz an: "))
    rgrenze = int(input("Geben Sie die rechte Grenze der Konkordanz an: "))

    konkordanzen= do_work(word, arbeitsliste, lgrenze, rgrenze)
    #Printed jede Konkordanz aus Mutterliste einzeln
    for sub_list in konkordanzen:
        print (sub_list)

    #Fragt User nach erneutem Programmaufruf
    checked_string = input("MÃ¶chten Sie nach weiteren Konkordanzen suchen? (Y/N) ").lower()
    #Warnmeldung falls User so bloed ist und weder y noch n eingibt
    while checked_string !="y" and checked_string !="n":
        checked_string = input("Falsche Eingabe. Bitte geben Sie ein:(Y/N) ").lower()
```

```
import re


def konkordanz(dateiname_einlesen, suchwort, slicing_indexintervall):
    """(word, word, number)   ->  string, string, int
        returns the concordance and all found search term indexes of a within a textfile
        existing word by a user given name of textfile, as well as a search term and an
        intervall that will control for how many indexes the concordance shall be identified
    >>> Gib den Dateinamen der einzulesenden Datei ein: minitext_token
    >>> Gib ein Suchwort ein, um seine Konkrodanz zu erzeugen: Kaninchenbau
    >>>Gib eine Zahl ein, um den Indexintervall zu bestimmen, in dem links und rechts um das Suchwort
die Konkordanz erzeugt werden soll: 3
    Hinunter in den          Kaninchenbau         .  Alice                    Suchwortindex: 6
    Der Eingang zum          Kaninchenbau         lief erst geradeaus         Suchwortindex: 305
    """
    wortmuster = re.compile("[A-Za-zäöü]")
    zahlenmuster = re.compile("[0-9]")
    inhalte_datei = open(dateiname_einlesen + ".txt", "r", encoding="utf8")
    rohzeichenketten_liste = inhalte_datei.readlines()
    rohwerte_index = 0
    zeichenketten_liste = []
    konkordanz_datei = open("konkordanz.txt", "a", encoding = "utf8")
    print("Gefundene Konkordanzen zum Suchwort '{}':".format(suchwort), file = konkordanz_datei)
    konkordanz_datei.close()
    if not re.search(wortmuster, suchwort):
        print("Bei Deiner Eingabe für das Suchwort handelt es sich nicht um ein Wort! Gib erneut die
angefragten Werte ein!")
        return konkordanz(input("Gib den Dateinamen der einzulesenden Datei ein: "), input("Gib ein
```

```
Suchwort ein, um seine Konkordanz zu erzeugen: "), input("Gib eine Zahl ein, um den Indexintervall zu bestimmen, in dem links und rechts um das Suchwort die Konkordanz erzeugt werden soll: "))
    elif not re.search(zahlenmuster, slicing_indexintervall):
        print("Bei Deiner Eingabe für das Konkordanzintervall handelt es sich nicht um eine Zahl! Gib erneut die angefragten Werte ein!")
        return konkordanz(input("Gib den Dateinamen der einzulesenden Datei ein: "), input("Gib ein Suchwort ein, um seine Konkordanz zu erzeugen: "), input("Gib eine Zahl ein, um den Indexintervall zu bestimmen, in dem links und rechts um das Suchwort die Konkordanz erzeugt werden soll: "))
    else:
        while rohwerte_index < len(rohzeichenketten_liste):    #while-Schleife, um durch rohzeichenketten_liste zu laufen und unerwünschte Zeichen zu entfernen
            zeichenketten_liste.append(rohzeichenketten_liste[rohwerte_index].strip("\n"))  #an betreffender Indexposition wird \n von der Zeichenkette entfernt und bereinigte Indexposition wird an Liste "zeichenketten_liste" angehangen
            rohwerte_index = rohwerte_index + 1          #nach jedem Schleifendurchlauf wird rohwerte_index um 1 erweitert
        #print(zeichenketten_liste)
        if suchwort not in zeichenketten_liste:     #Abfangen von falschen Suchwort-Eingaben durch den Nutzer
            print("Dein Suchwort '{}' konnte nicht in der Datei gefunden werden. Gib erneut die angefragten Werte ein!".format(suchwort))
            return konkordanz(input("Gib den Dateinamen der einzulesenden Datei ein: "), input("Gib ein Suchwort ein, um seine Konkordanz zu erzeugen: "), input("Gib eine Zahl ein, um den Indexintervall zu bestimmen, in dem links und rechts um das Suchwort die Konkordanz erzeugt werden soll: "))
        suchwort_index = 0
        while suchwort_index < len(zeichenketten_liste):
            if re.match(suchwort, zeichenketten_liste[suchwort_index]):
                x = suchwort_index - int(slicing_indexintervall)   #Definition der x-Indexkoordinate: wie hoch der Index für das Slicing nach vorne (weg vom Suchwortindex) sein soll
                y = suchwort_index + int(slicing_indexintervall) + 1    #Definition der y-Indexkoordinate: wie hoch der Index für das Slicing nach hinten (weg vom Suchwortindex) sein soll
                vorslicing_suchwortindex = zeichenketten_liste[x:suchwort_index]     #Definition des Slicingintervalls vom Suchwort weg nach vorne
                vorslicing_suchwortindex_string = " ".join(vorslicing_suchwortindex)   #Umwandlung in String
                vorslicing_suchwortindex_stringlaenge = len(vorslicing_suchwortindex_string)
                #Formatierung Textdatei "konkordanz.txt" - wie viele Leerzeichen sollen zwischen angezeigtem Suchwort und nach Vorne hin gefundener Konkordanz liegen:
                vorslicing_suchwortindex_string += (40 - vorslicing_suchwortindex_stringlaenge)*" "
                nachslicing_suchwortindex = zeichenketten_liste[suchwort_index+1:y]   #Definition des Slicingintervalls vom Suchwort weg nach hinten
                nachslicing_suchwortindex_string = " ".join(nachslicing_suchwortindex)  #Umwandlung in String
                nachslicing_suchwortindex_stringlaenge = len(nachslicing_suchwortindex_string)
                #Formatierung Textdatei "konkordanz.txt" - wie viele Leerzeichen sollen zwischen der vom Suchwort ausgehend nach Hinten hin gefundener Konkordanz und Suchwort-Index-Anzeige liegen:
                nachslicing_suchwortindex_string += (45 - nachslicing_suchwortindex_stringlaenge)*" "
                #print("Das Suchwort '{}' befinet sich in der Liste.".format(suchwort))
                string = " "
                string_laenge = len(suchwort)
                #Formatierung Textdatei "konkordanz.txt" - wie viele Leerzeichen sollen zwischen Suchwort und nach Hinten hin gefundener Konkordanz liegen:
                string += (20 - string_laenge) * " "
```

```
                    print("{} {} {} Suchwortindex: {}".format(vorslicing_suchwortindex, suchwort, nachslicing_suchwortindex, suchwort_index))
                    suchwort_index = suchwort_index + 1       #nach jedem Schleifendurchlauf wird suchwort_index um 1 erweitert
                    #Inhalte in Datei "konkordanzen.txt" schreiben:
                    konkordanz_datei = open("konkordanz.txt", "a", encoding = "utf8")
                    #print("{} {} {} Suchwortindex: {}".format(vorslicing_suchwortindex, suchwort, nachslicing_suchwortindex, suchwort_index), file = konkordanz_datei)
                    print("{} {} {} Suchwortindex: {}".format(vorslicing_suchwortindex_string, suchwort + string, nachslicing_suchwortindex_string, suchwort_index), file = konkordanz_datei)
                    konkordanz_datei.close()
                else:
                    suchwort_index = suchwort_index + 1   #ist Suchwort unter suchwort_index nicht in Liste, suche in der Liste eine Position weiter


    #Abfrage, ob Programm erneut durchgeführt werden soll:
    choice = ("y" or "Y" or "n" or "N")
    while choice == ("y" or "Y" or "n" or "N"):
        while True:
            choice = input("Möchtest Du die Konkordanzen für ein weiteres Suchwort ermitteln? [y/n] - ")
            if choice == "y" or choice == "Y":
                konkordanz(input("Gib den Dateinamen der einzulesenden Datei ein: "), input("Gib ein Suchwort ein, um seine Konkordanz zu erzeugen: "), input("Gib eine Zahl ein, um den Indexintervall zu bestimmen, in dem links und rechts um das Suchwort die Konkordanz erzeugt werden soll: "))
            elif choice == "n" or choice == "N":
                print("Gut! Dann bis zum nächsten Mal.")
                break
            elif (choice != "y" or choice != "Y") or (choice != "n" or choice != "N"):
                choice = input("Möchtest Du die Konkordanzen für ein weiteres Suchwort ermitteln? [y/n] - ")



konkordanz(input("Gib den Dateinamen der einzulesenden Datei ein: "), input("Gib ein Suchwort ein, um seine Konkordanz zu erzeugen: "), input("Gib eine Zahl ein, um den Indexintervall zu bestimmen, in dem links und rechts um das Suchwort die Konkordanz erzeugt werden soll: "))

import re
import random

def read_cmu_file(cmu_path):
    """
    string -> dictionary
    Erzeuge aus einer cmudict-Datei ein Python-Dictionary mit orthographischen
    WÃ¶rtern als Keys und ihren phonetischen Transkriptionen als Values.
    """
    data = open(cmu_path, "r", encoding="utf8")
    my_cmu_dict = {}
    for line in data:
        # Nur Zeilen behandeln, die nicht leer sind
        if line.strip() != "":
            # Nur Zeilen behandeln, die keine Kommentarzeilen sind
```

```
            if not line.startswith(";;;"):
                # Zeilen in orthographische Form und phonetische Form aufteilen
                line = line.split("  ", 1)
                orth = line[0].strip()
                phon = line[1].strip()
                # Setze den Wert fÃ¼r den Key "orth" im Dictionary auf "phon"
                my_cmu_dict[orth] = phon
    return my_cmu_dict


def get_any_transcription(input_word, my_cmu_dict):
    """
    string, dictionary -> string
    Ermittle aus dem Ã¼bergebenen Dictionary den Transkriptionswert des
    Ã¼bergebenen Wortes. Wenn das Wort nicht gefunden wird, ist der Wert "None".
    >>> get_any_transcription("apple", my_cmu_dict)
    AE1 P AH0 L
    >>> get_any_transcription("zqx", my_cmu_dict)
    Dieses Wort kann nicht nachgeschlagen werden.
    None
    """
    # Nur groÃŸgeschriebene WÃ¶rter sind im Dictionary enthalten
    input_word = input_word.upper()
    if input_word in my_cmu_dict.keys():
        return my_cmu_dict[input_word]
    else:
        print("Dieses Wort kann nicht nachgeschlagen werden.")
        return None


def get_rhime(phon):
    match = re.search("([A-Z]+[1-2][^1-2]*$)",phon)
    if match:
        rhime= match.group(1)
    else:
        rhime = ""
    return rhime

def same_rhime(rhime, my_cmu_dict):
    """ dict nehmen und den gleichen reim finden . LIste von wörten als ausgabe """
    reim = rhime
    reimliste = []
    for key in my_cmu_dict:
        if get_rhime(my_cmu_dict[key]) == reim:
            neuerReim = key
            reimliste.append(neuerReim)
    return reimliste

def get_woerter(my_cmu_dict, reimliste):
    wortliste = []
    while len(wortliste) < 12:
        neuesWort = my_cmu_dict.popitem()
        wortliste.append(neuesWort)
```

```
    return wortliste


# Rufe die Lesefunktion mit dem Pfad zur Originaldatei auf (ggf. anpassen)
my_cmu_dict = read_cmu_file("C:/Users/std_043/Desktop/cmudict.txt")

# Nutzerinput: Solange ein Wort eingegeben wird, wird die Nachschlagefunktion
# aufgerufen
input_word = input("Wort nachschlagen (Enter zum Beenden): ")
while input_word != "":
    phon = get_any_transcription(input_word, my_cmu_dict)
    print("Transkription: ", phon)
    print("Reim: ", get_rhime(phon))
    print("Reimliste", same_rhime(get_rhime(phon), my_cmu_dict))
    print("Wortliste", get_woerter(my_cmu_dict, same_rhime(get_rhime(phon), my_cmu_dict)))
    input_word = input_word = input("Wort nachschlagen (Enter zum Beenden): ")
```