WORD & SENTENCES GENERATOR

Project Definition: This project, aims to teach student how to use Pythons common libs, Zemberek Turkish NLP Framework, Java Virtual Machine etc. Project has 4 related **different module.** Modules can has different needs so their **approach** and their **scope** can be **fairly different**.

Used Technologies: JVM, PYTHON3

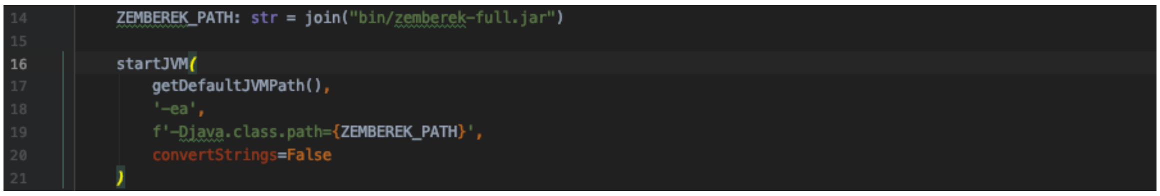
Project Team:

Zeynep Pehlivan:	**150116504**
Ceyda Polat:	**150117065**
Taha Yusuf Kömür:	**150114064**

COMMON CODE SEGMENT FOR ALL MODULES

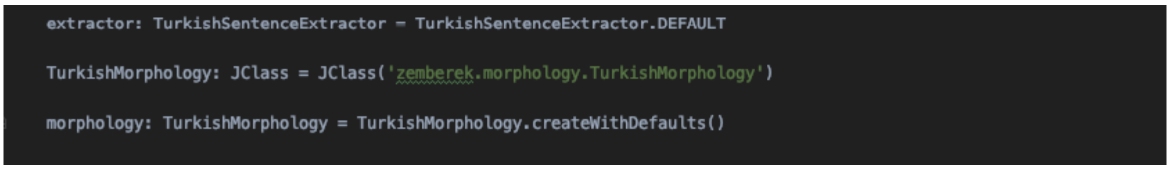
Code:

1- Starting JVM to install Zemberek



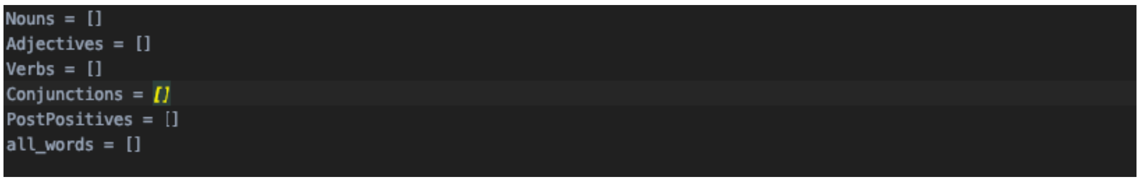
```python
    ZEMBEREK_PATH: str = join("bin/zemberek-full.jar")

    startJVM(
        getDefaultJVMPath(),
        '-ea',
        f'-Djava.class.path={ZEMBEREK_PATH}',
        convertStrings=False
    )
```

2- Creating Jclass Object



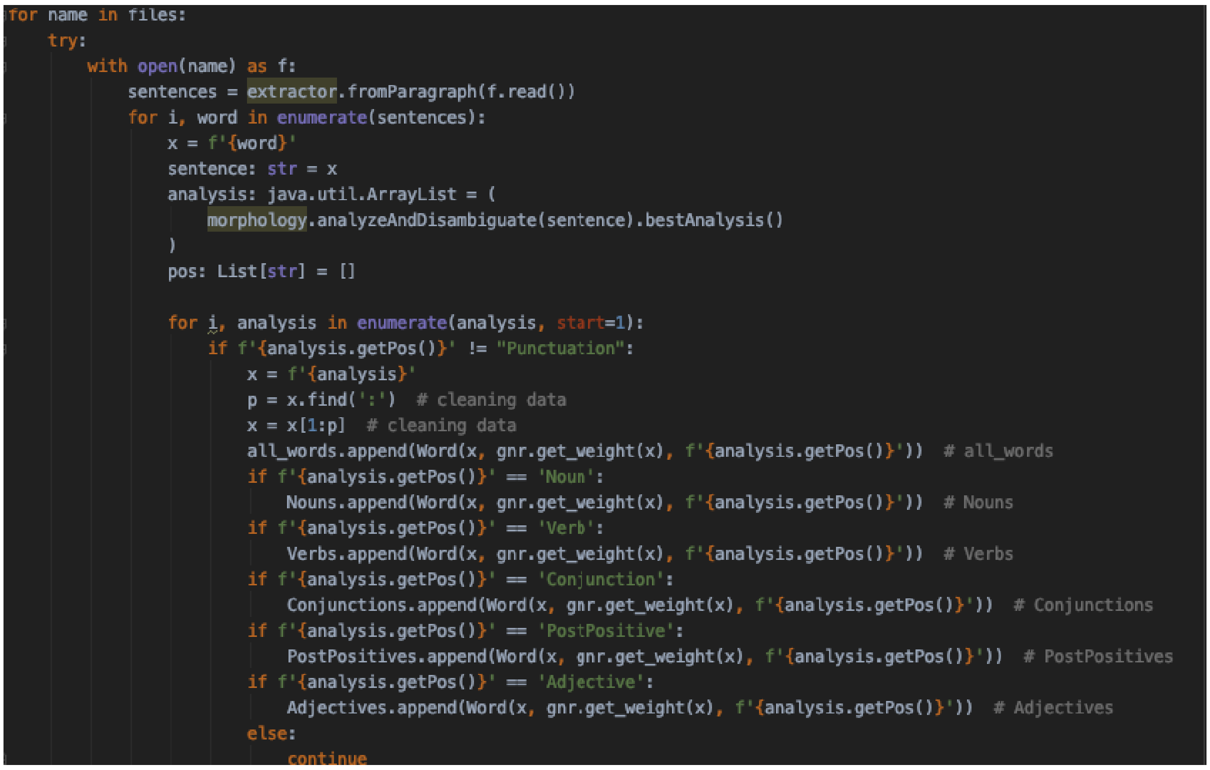
```python
    extractor: TurkishSentenceExtractor = TurkishSentenceExtractor.DEFAULT

    TurkishMorphology: JClass = JClass('zemberek.morphology.TurkishMorphology')

    morphology: TurkishMorphology = TurkishMorphology.createWithDefaults()
```

3- List For Words



```python
Nouns = []
Adjectives = []
Verbs = []
Conjunctions = []
PostPositives = []
all_words = []
```

4- Filling lists



```python
for name in files:
    try:
        with open(name) as f:
            sentences = extractor.fromParagraph(f.read())
            for i, word in enumerate(sentences):
                x = f'{word}'
                sentence: str = x
                analysis: java.util.ArrayList = (
                    morphology.analyzeAndDisambiguate(sentence).bestAnalysis()
                )
                pos: List[str] = []

                for i, analysis in enumerate(analysis, start=1):
                    if f'{analysis.getPos()}' != "Punctuation":
                        x = f'{analysis}'
                        p = x.find(':')  # cleaning data
                        x = x[1:p]  # cleaning data
                        all_words.append(Word(x, gnr.get_weight(x), f'{analysis.getPos()}'))  # all_words
                        if f'{analysis.getPos()}' == 'Noun':
                            Nouns.append(Word(x, gnr.get_weight(x), f'{analysis.getPos()}'))  # Nouns
                        if f'{analysis.getPos()}' == 'Verb':
                            Verbs.append(Word(x, gnr.get_weight(x), f'{analysis.getPos()}'))  # Verbs
                        if f'{analysis.getPos()}' == 'Conjunction':
                            Conjunctions.append(Word(x, gnr.get_weight(x), f'{analysis.getPos()}'))  # Conjunctions
                        if f'{analysis.getPos()}' == 'PostPositive':
                            PostPositives.append(Word(x, gnr.get_weight(x), f'{analysis.getPos()}'))  # PostPositives
                        if f'{analysis.getPos()}' == 'Adjective':
                            Adjectives.append(Word(x, gnr.get_weight(x), f'{analysis.getPos()}'))  # Adjectives
                    else:
                        continue
```

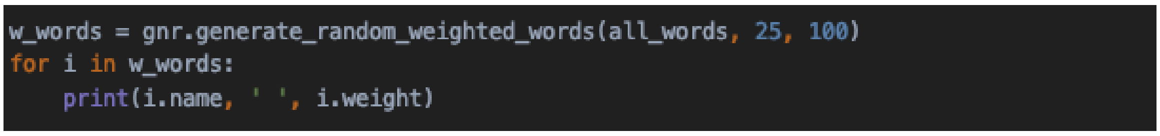
Module 1

Scope: With a given dataset, extracting words which's total alphabetic weight is equal to the given input.

Approach: Our approach is, using 4 different **Class** for this module which are:

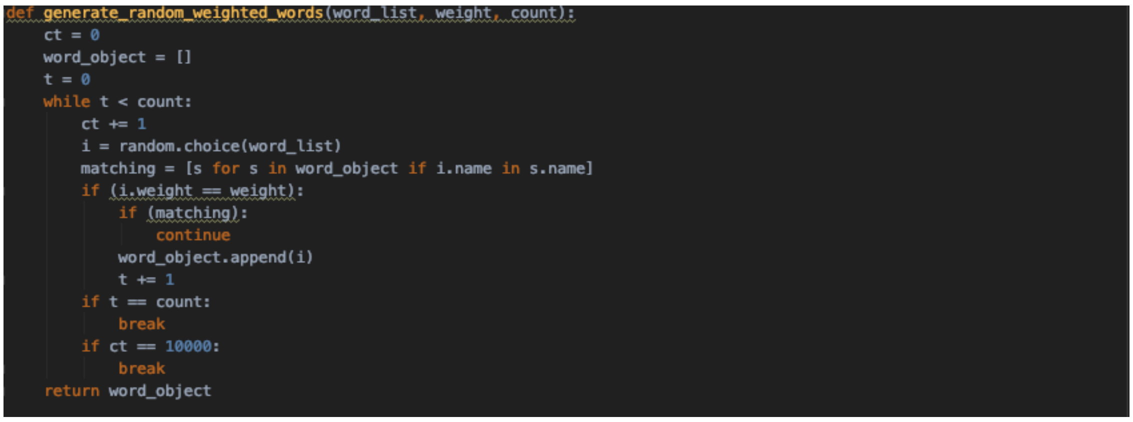
Generator: To keep necessary methods
Main: To drive the programme
Jclass: To use Zemberek Framework
Word: To store necessary information about word inside an object.

1- **Calling method inside main**



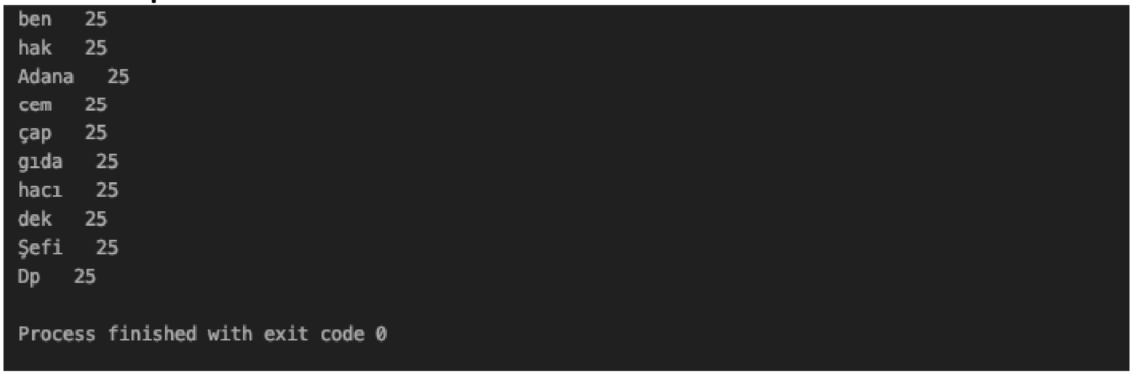
```
w_words = gnr.generate_random_weighted_words(all_words, 25, 100)
for i in w_words:
    print(i.name, ' ', i.weight)
```

2- **Generate_random_weighted_words()**



```
def generate_random_weighted_words(word_list, weight, count):
    ct = 0
    word_object = []
    t = 0
    while t < count:
        ct += 1
        i = random.choice(word_list)
        matching = [s for s in word_object if i.name in s.name]
        if (i.weight == weight):
            if (matching):
                continue
            word_object.append(i)
            t += 1
        if t == count:
            break
        if ct == 10000:
            break
    return word_object
```

3- **Output:**



```
ben   25
hak   25
Adana   25
cem   25
çap   25
gıda   25
hacı   25
dek   25
Şefi   25
Dp   25

Process finished with exit code 0
```

Module 2:

Scope: With a given dataset, extracting words and generate random sentences which's total alphabetic weight is equal to the given input.

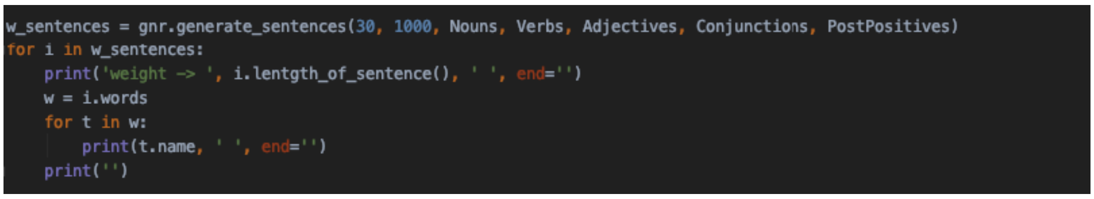
Approach: Our approach is, using 5 different **Class** for this module which are:

Generator: To keep necessary methods
Main: To drive the programme
Jclass: To use Zemberek Framework
Word: To store necessary information about word, inside an object.
Sentence: To store necessary information about sentence, inside an object

Average weight of:
Verbs = 83.57375024377559
Nouns = 74.10337317397078
Adjectives = 74.18319491939424
Conjunctions = 26.215071283095725
PostPositives = 54.84366373902133

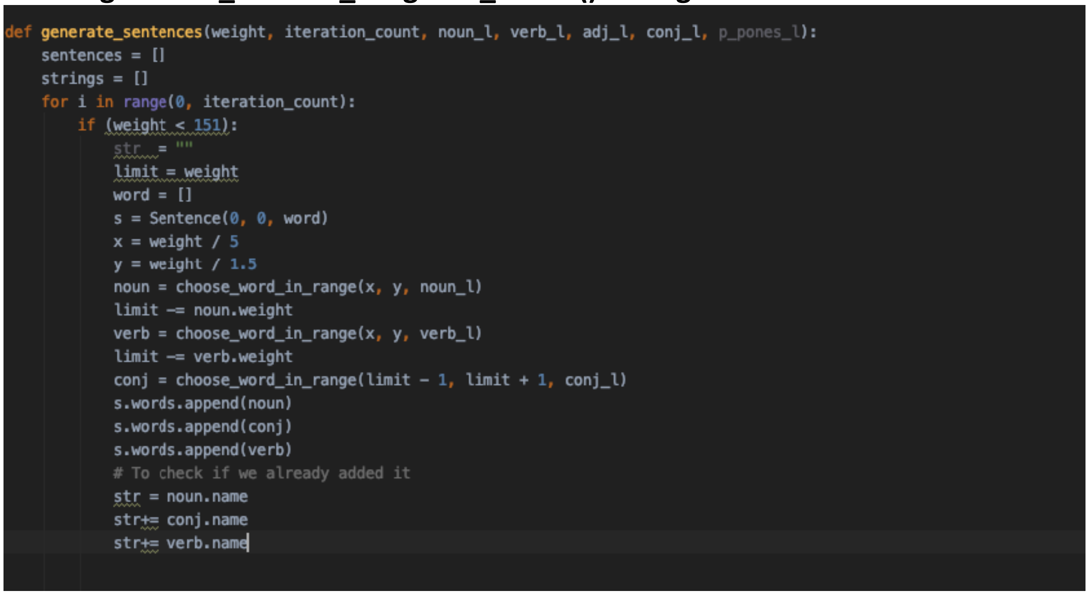
We decided to divide our generator function to 3, because depends to input, we might need to use smaller type of words, or vice versa.

1- **Calling the function:**



```
w_sentences = gnr.generate_sentences(30, 1000, Nouns, Verbs, Adjectives, Conjunctions, PostPositives)
for i in w_sentences:
    print('weight -> ', i.lentgth_of_sentence(), ' ', end='')
    w = i.words
    for t in w:
        print(t.name, ' ', end='')
    print('')
```

2- **generate_random_weighted_words() # weight<151**



```
def generate_sentences(weight, iteration_count, noun_l, verb_l, adj_l, conj_l, p_pones_l):
    sentences = []
    strings = []
    for i in range(0, iteration_count):
        if (weight < 151):
            str = ""
            limit = weight
            word = []
            s = Sentence(0, 0, word)
            x = weight / 5
            y = weight / 1.5
            noun = choose_word_in_range(x, y, noun_l)
            limit -= noun.weight
            verb = choose_word_in_range(x, y, verb_l)
            limit -= verb.weight
            conj = choose_word_in_range(limit - 1, limit + 1, conj_l)
            s.words.append(noun)
            s.words.append(conj)
            s.words.append(verb)
            # To check if we already added it
            str = noun.name
            str+= conj.name
            str+= verb.name
```

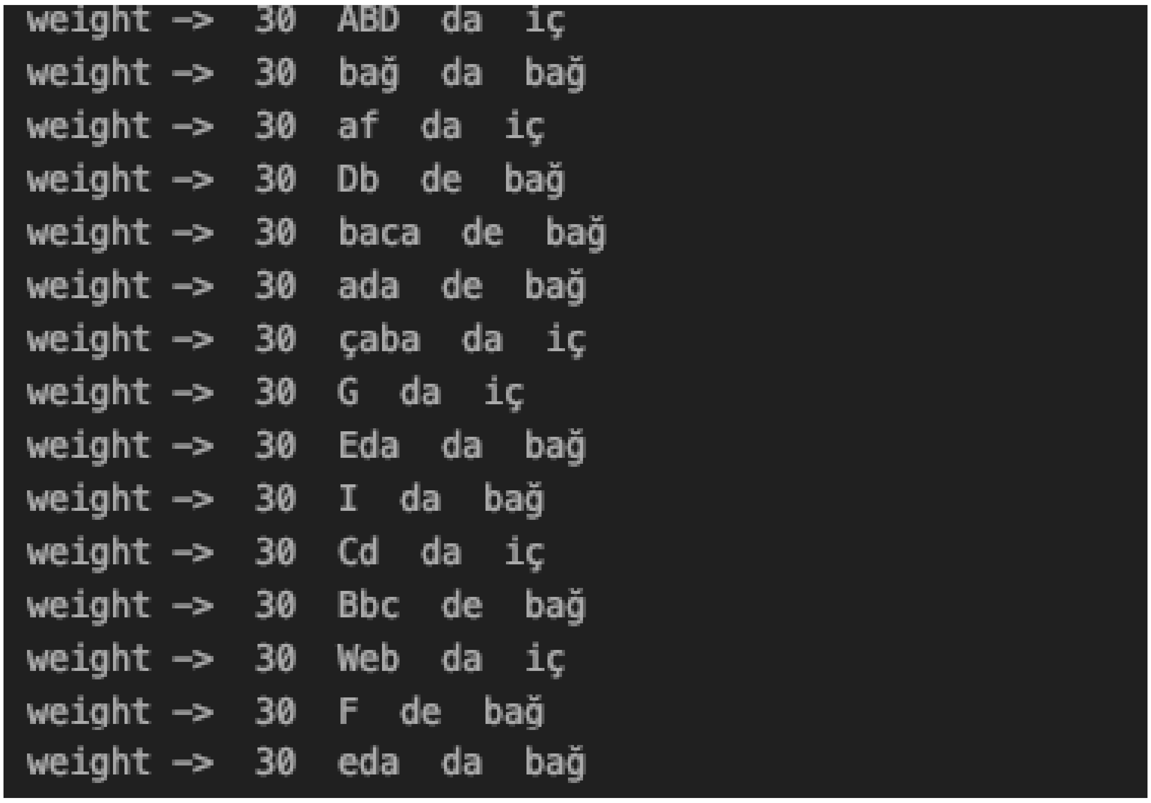
3- **generate_random_weighted_words() # 150 < weight < 251**

```
if (150 < weight < 251):
    limit = weight
    word = []
    s = Sentence(0, 0, word)
    x = weight / 5
    y = weight / 2
    noun = choose_word_in_range(x, y, noun_l)
    limit -= noun.weight
    verb = choose_word_in_range(x, y, verb_l)
    limit -= verb.weight

    s.words.append(noun)
    while True:
        if (limit >= 80):
            x = choose_word_in_range(30, 50, adj_l)
            limit -= x.weight
            s.words.append(x)
        else:
            break
    if (limit > 10):
        p_pone = choose_word_in_range(limit - 1, limit + 1, conj_l)
        s.words.append(p_pone)
    s.words.append(verb)

    if (s.lentgth_of_sentence() == weight):
        sentences.append(s)
```

4- **generate_random_weighted_words() # 250 < weight**

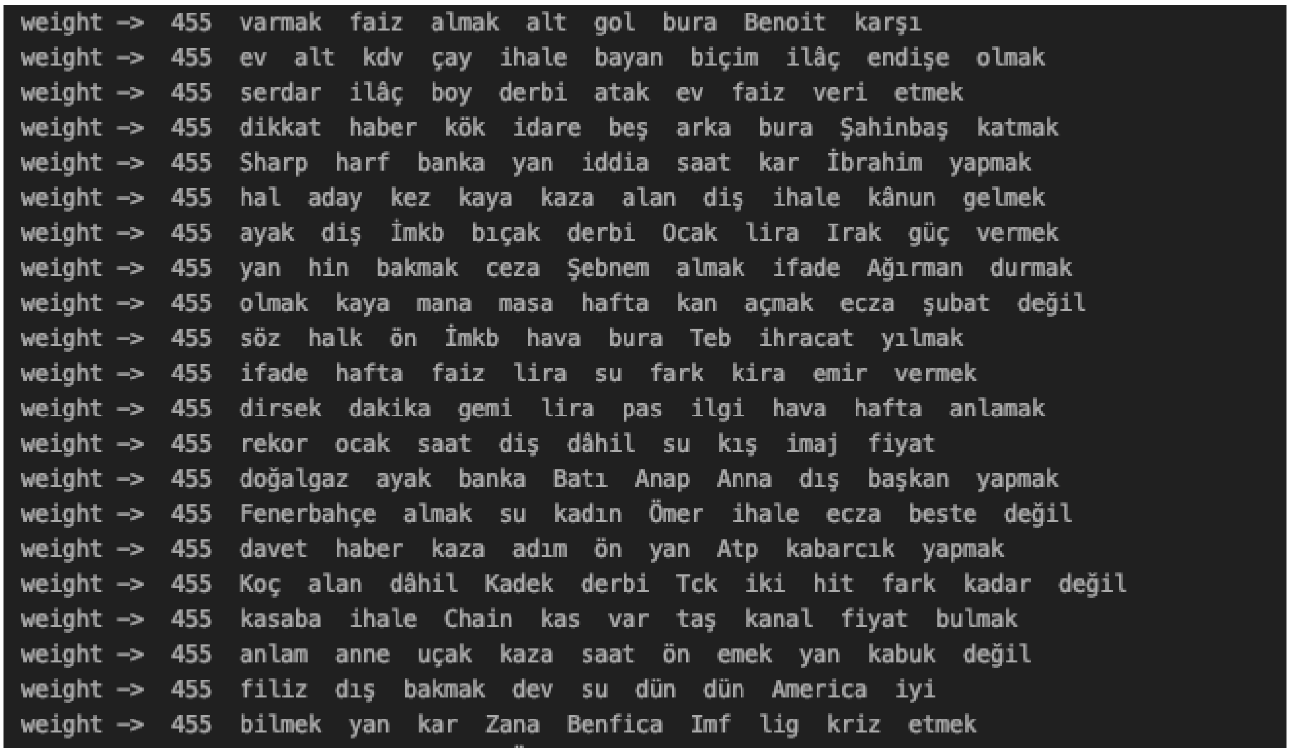
```
if (250 < weight):
        limit = weight
        word = []
        s = Sentence(0, 0, word)
        x = weight / 25
        y = weight / 5
        noun = choose_word_in_range(x, y, noun_l)
        limit -= noun.weight
        verb = choose_word_in_range(x, y, verb_l)
        limit -= verb.weight

        s.words.append(noun)
        while True:
            if (limit >= 80):
                x = choose_word_in_range(30, 50, noun_l)
                limit -= x.weight
                s.words.append(x)
            else:
                break
        if (limit > 10):
            conj = choose_word_in_range(limit - 1, limit + 1, noun_l)
            s.words.append(conj)
        s.words.append(verb)
        if (s.lentgth_of_sentence() == weight):
            sentences.append(s)

return sentences
```

5- **Output for 30:**



```
weight ->  30  ABD  da  iç
weight ->  30  bağ  da  bağ
weight ->  30  af  da  iç
weight ->  30  Db  de  bağ
weight ->  30  baca  de  bağ
weight ->  30  ada  de  bağ
weight ->  30  çaba  da  iç
weight ->  30  G  da  iç
weight ->  30  Eda  da  bağ
weight ->  30  I  da  bağ
weight ->  30  Cd  da  iç
weight ->  30  Bbc  de  bağ
weight ->  30  Web  da  iç
weight ->  30  F  de  bağ
weight ->  30  eda  da  bağ
```

6- Output for 455



```
weight ->  455  varmak  faiz  almak  alt  gol  bura  Benoit  karşı
weight ->  455  ev  alt  kdv  çay  ihale  bayan  biçim  ilâç  endişe  olmak
weight ->  455  serdar  ilâç  boy  derbi  atak  ev  faiz  veri  etmek
weight ->  455  dikkat  haber  kök  idare  beş  arka  bura  Şahinbaş  katmak
weight ->  455  Sharp  harf  banka  yan  iddia  saat  kar  İbrahim  yapmak
weight ->  455  hal  aday  kez  kaya  kaza  alan  diş  ihale  kânun  gelmek
weight ->  455  ayak  diş  İmkb  bıçak  derbi  Ocak  lira  Irak  güç  vermek
weight ->  455  yan  hin  bakmak  ceza  Şebnem  almak  ifade  Ağırman  durmak
weight ->  455  olmak  kaya  mana  masa  hafta  kan  açmak  ecza  şubat  değil
weight ->  455  söz  halk  ön  İmkb  hava  bura  Teb  ihracat  yılmak
weight ->  455  ifade  hafta  faiz  lira  su  fark  kira  emir  vermek
weight ->  455  dirsek  dakika  gemi  lira  pas  ilgi  hava  hafta  anlamak
weight ->  455  rekor  ocak  saat  diş  dâhil  su  kış  imaj  fiyat
weight ->  455  doğalgaz  ayak  banka  Batı  Anap  Anna  dış  başkan  yapmak
weight ->  455  Fenerbahçe  almak  su  kadın  Ömer  ihale  ecza  beste  değil
weight ->  455  davet  haber  kaza  adım  ön  yan  Atp  kabarcık  yapmak
weight ->  455  Koç  alan  dâhil  Kadek  derbi  Tck  iki  hit  fark  kadar  değil
weight ->  455  kasaba  ihale  Chain  kas  var  taş  kanal  fiyat  bulmak
weight ->  455  anlam  anne  uçak  kaza  saat  ön  emek  yan  kabuk  değil
weight ->  455  filiz  dış  bakmak  dev  su  dün  dün  America  iyi
weight ->  455  bilmek  yan  kar  Zana  Benfica  Imf  lig  kriz  etmek
```

Module 3:

Scope: With a given dataset, extracting words and generate random sentences which's total alphabetic weight is equal to the given input and semantically true!

Approach: Our approach is, using 2 different **Class** for this module which are:

Main: To drive the programme
Jclass: To use Zemberek Framework

Algorithm: Extracting meaninful sentences and changing its words with their synonyms by external source :

```
url = "http://www.es-anlam.com/kelime/" + kelime_new
```

We have 3 function to achieve this module

Functios:

- **generate_sentence()**
- **find_synonyms()**
- **Add_suffix()**

Algorithm: Firstly we generated 4 different lists which includes different types of sentence element. These are subject, adjective, noun and verb using Zemberek Library to determine word's types.

Then we sent request to the "http://www.es-anlam.com/" site and pull the synonyms of words that we have in lists.

We changed words which have synonym in the sentence. And added appropriate suffix to the word with Zemberek. Then we checked the sentence's value is whether equal to the given sentence value or not.

1- generate_sentence()

```
def generate_sentence():
    global verb_list
    global new_sentence
    sentence_sum = 600 #SENTENCE VALUE

    for i in wordList:
        sentence: str = i
        analysis: java.util.ArrayList = morphology.analyzeSentence(sentence)

        results: java.util.ArrayList = (
            morphology.disambiguate(sentence, analysis).bestAnalysis()
        )

        for j, result in enumerate(results, start=0):
            x = (str(result)).split(":")[1]
            y = x.split("]")[0]

            if y == "Verb": #Select Verb Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                verb_list.append(y)
                verb_list = list(dict.fromkeys(verb_list))

    gen_sentence_list = []
    sentence_value = 0
    cntrl = 0

    print("GENERATING SENTENCE, PLEASE WAIT...")
    for sentence in tokenized_sentence:
        if cntrl == 1:
            new_sentence = ""
        synonym_sentence = ""
        synonym_sentence = find_synonyms(sentence)
        synonym_sentence = synonym_sentence[1:]
        new_sentence = synonym_sentence.capitalize()
#          print("NEW SENTENCE: " + new_sentence)
```

```
        cntrl = 1

        sentence_value = valueList(synonym_sentence)

        total_value = 0
        for value in sentence_value:
            total_value += value

        if total_value <= sentence_sum+100 and total_value > sentence_sum:
            gen_sentence_list.append(new_sentence)

        if len(gen_sentence_list) == 1:
            break

    for i in gen_sentence_list:

        print(i)
```

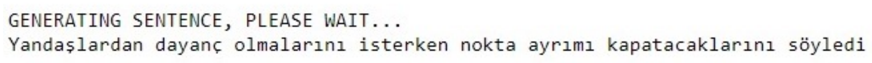
2- find_synonyms()

```
def find_synonyms(sentence):
    global new_sentence
    sentence = sentence.translate(str.maketrans('', '', string.punctuation))
#     print("ORIGINAL SENTENCE: " + sentence)
    return_sentence = ""
    sentence_words = []
    sentence_for_synonym_search = []

    if __name__ == '__main__':

        ZEMBEREK_PATH: str = join('..', '..', 'bin', 'zemberek-full.jar')

        TurkishMorphology: JClass = JClass('zemberek.morphology.TurkishMorphology')
        Paths: JClass = JClass('java.nio.file.Paths')

        morphology: TurkishMorphology = TurkishMorphology.createWithDefaults()

    for i in sentence.split():
        sentence: str = i
        analysis: java.util.ArrayList = morphology.analyzeSentence(sentence)

        results: java.util.ArrayList = (
            morphology.disambiguate(sentence, analysis).bestAnalysis()
        )

        #add sentence just subject, adjective, noun and verb from original sentence
        for j, result in enumerate(results, start=0):
            x = (str(result)).split(":")[1]
            y = x.split("]")[0]
            if y == "Pron,Pers": #Select Subject Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_for_synonym_search.append(y)
            elif y == "Adj": #Select Adj Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_words.append(sentence.split()[j])
                sentence_for_synonym_search.append(y)
            elif y == "Noun": #Select Noun Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_words.append(sentence.split()[j])
                sentence_for_synonym_search.append(y)
            elif y == "Verb": #Select Verb Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_words.append(sentence.split()[j])
                sentence_for_synonym_search.append(y)

    word_new = ""

    for kelime,original_word in zip(sentence_for_synonym_search,sentence_words):
        kelime_new = kelime.replace('ö','o')
        kelime_new = kelime_new.replace('ı','i')
        kelime_new = kelime_new.replace('ü','u')
        kelime_new = kelime_new.replace('ç','c')
        kelime_new = kelime_new.replace('ş','s')
        kelime_new = kelime_new.replace('ğ','g')
        kelime_new = kelime_new.strip()
#         print("KELIME_NEW: " + kelime_new)
        #find synonyms from site
#         try:

        if kelime not in verb_list:
            user_agent = 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.7) Gecko/2009021910 Firefox/3.0.7'

            url = "http://www.es-anlam.com/kelime/" + kelime_new
            headers={'User-Agent':user_agent,}

            request=urllib.request.Request(url,None,headers) #The assembled request
            response = urllib.request.urlopen(request)
            soup = BeautifulSoup(response, "html.parser")

            ana = soup.find('h2')
            alt=ana.find('strong')
            alt = str(alt)
            alt = (alt.split(">")[1]).split(",")[0]
            alt = alt.split("<")[0]
            alt = alt.strip()

            if alt == "BULUNAMADI !":
                new_sentence = new_sentence + " " + original_word
            else:
#                   new_sentence = new_sentence + " " + alt
                word_new = alt

                add_suffix(original_word, word_new)
#               print("NEW_SENTENCE IF: " + new_sentence)

        else:
            new_sentence += " " + original_word
#               print("NEW_SENTENCE ELSE: " + new_sentence)

    return new_sentence
```

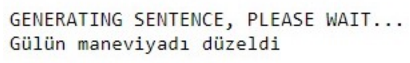
3- add_suffix()

```
def add_suffix(sentence_word, synonym_word):
    global new_sentence
    TurkishMorphology: JClass = JClass('zemberek.morphology.TurkishMorphology')
    WordAnalysis: JClass = JClass('zemberek.morphology.analysis.WordAnalysis')

    morphology: TurkishMorphology = TurkishMorphology.createWithDefaults()

    results: WordAnalysis = morphology.analyze(JString(sentence_word))

    number = ""
    possessive = ""
    case = ""
    word = ""
    for result in results:
        result = str(result)
        try:#set number
            if "A3sg" in result:
                number = "A3sg"
            elif "A3pl" in result:
                number = "A3pl"
        except:
            number = ""

        try: #set possessive
            if "P1sg" in result:
                possessive ="P1sg"
            elif "P2sg" in results:
                possessive = "P2sg"
            elif "P3sg" in result:
                possessive = "P3sg"
        except:
            possessive = ""

        try: #set case
            if "Dat" in result:
                case = "Dat"
            elif "Loc" in result:
                case = "Loc"
            elif "Abl" in result:
                case = "Abl"
        except:
            case = ""   #cases: List[JString] = [JString('Dat'), JString('Loc'), JString('Abl')]

        word = result.split(" ")[1]
        word = word.split(":")[0]

    morphology: TurkishMorphology = (
        TurkishMorphology.builder().setLexicon(synonym_word).disableCache().build()
    )

    try:
        item = morphology.getLexicon().getMatchingItems(synonym_word).get(0)

        if number != "" and possessive != "" and case != "":
            for result in morphology.getWordGenerator().generate(item, number, possessive, case):
                new_sentence += " " + str(result.surface)
        elif number != "" and possessive != "" and case == "":
             for result in morphology.getWordGenerator().generate(item, number, possessive):
                new_sentence += " " + str(result.surface)
        elif number != "" and possessive == "" and case != "":
             for result in morphology.getWordGenerator().generate(item, number, case):
                new_sentence += " " + str(result.surface)
        elif number == "" and possessive != "" and case != "":
             for result in morphology.getWordGenerator().generate(item, possessive, case):
                new_sentence += " " + str(result.surface)
        elif number == "" and possessive == "" and case != "":
             for result in morphology.getWordGenerator().generate(item, case):
                new_sentence += " " + str(result.surface)
        elif number == "" and possessive != "" and case == "":
             for result in morphology.getWordGenerator().generate(item, possessive):
                new_sentence += " " + str(result.surface)
        elif number != "" and possessive == "" and case == "":
             for result in morphology.getWordGenerator().generate(item, number):
                new_sentence += " " + str(result.surface)
        else:
            new_sentence += str(result.surface)   except:
      pass
```

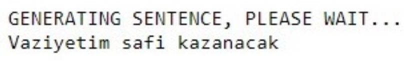
4- Output for module 3



```
GENERATING SENTENCE, PLEASE WAIT...
Yandaşlardan dayanç olmalarını isterken nokta ayrımı kapatacaklarını söyledi
```

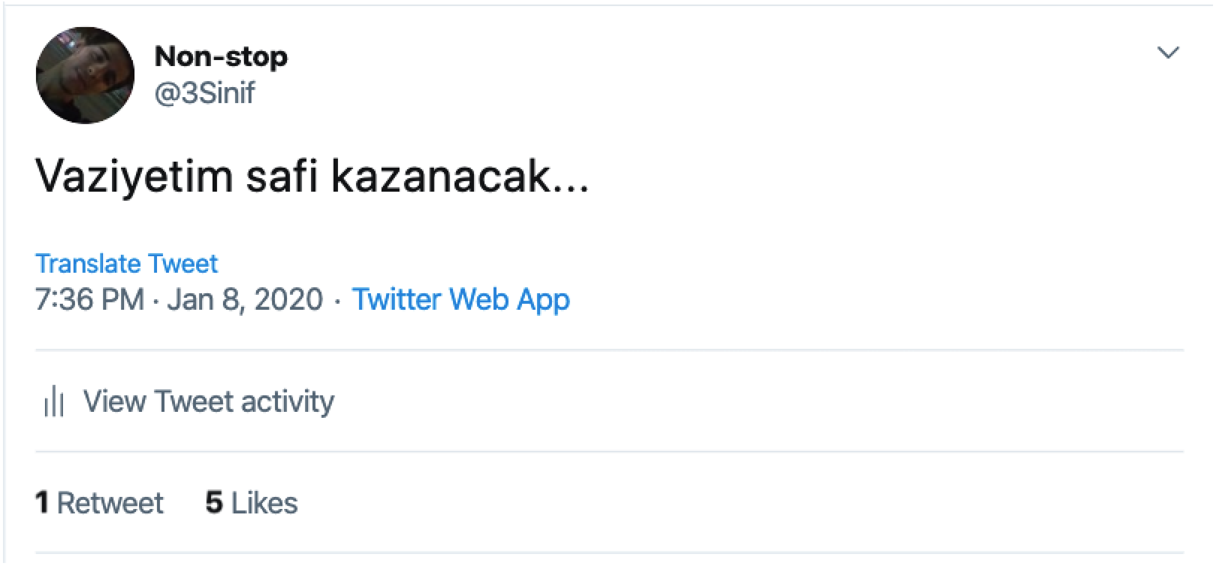


```
GENERATING SENTENCE, PLEASE WAIT...
Gülün maneviyadı düzeldi
```



```
GENERATING SENTENCE, PLEASE WAIT...
Vaziyetim safi kazanacak
```

We tweeted machine generated sentence and -> an interesting result.. People liked it.



Non-stop
@3Sinif

Vaziyetim safi kazanacak...

Translate Tweet
7:36 PM · Jan 8, 2020 · Twitter Web App

View Tweet activity

1 Retweet **5** Likes

Module 4:

Scope: With a given dataset, extracting words and generate random sentences which is positive or negative.

Approach: Our approach is, using 2 different **Class** for this module which are:

Main: To drive the programme
Jclass: To use Zemberek Framework

We have 3 function to achieve this module

- **generate_sentence()**
- **find_synonyms()**
- **Add_suffix()**

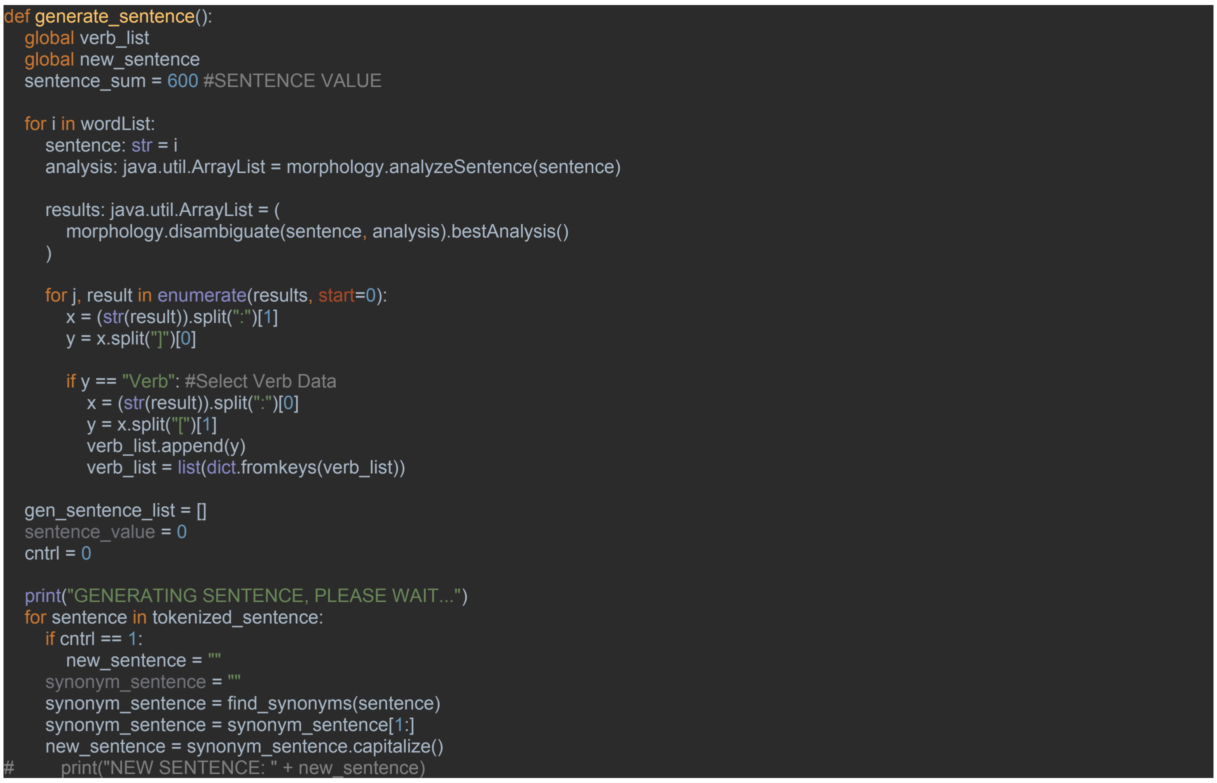
Algorithm definition: First we found sentences with labelled positive or negative and added it to the pandas dataframe.

Then we generated 4 different lists which includes different types of sentence element. These are subject, adjective, noun and verb using Zemberek Library to determine word's types.

Then we sent request to the "http://www.es-anlam.com/" site and pull the synonyms of words that we have in lists.

We changed words which have synonym in the sentence. And added appropriate suffix to the word with Zemberek. Then we checked the sentence's value is whether equal to the given sentence value or not.

1- generate_sentence()



```
def generate_sentence():
    global verb_list
    global new_sentence
    sentence_sum = 600 #SENTENCE VALUE

    for i in wordList:
        sentence: str = i
        analysis: java.util.ArrayList = morphology.analyzeSentence(sentence)

        results: java.util.ArrayList = (
            morphology.disambiguate(sentence, analysis).bestAnalysis()
        )

        for j, result in enumerate(results, start=0):
            x = (str(result)).split(":")[1]
            y = x.split("]")[0]

            if y == "Verb": #Select Verb Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                verb_list.append(y)
                verb_list = list(dict.fromkeys(verb_list))

    gen_sentence_list = []
    sentence_value = 0
    cntrl = 0

    print("GENERATING SENTENCE, PLEASE WAIT...")
    for sentence in tokenized_sentence:
        if cntrl == 1:
            new_sentence = ""
        synonym_sentence = ""
        synonym_sentence = find_synonyms(sentence)
        synonym_sentence = synonym_sentence[1:]
        new_sentence = synonym_sentence.capitalize()
#       print("NEW SENTENCE: " + new_sentence)
```

```
        cntrl = 1

        sentence_value = valueList(synonym_sentence)

        total_value = 0
        for value in sentence_value:
            total_value += value

        if total_value <= sentence_sum+100 and total_value > sentence_sum:
            gen_sentence_list.append(new_sentence)

        if len(gen_sentence_list) == 1:
            break

    for i in gen_sentence_list:



        print(i)
```

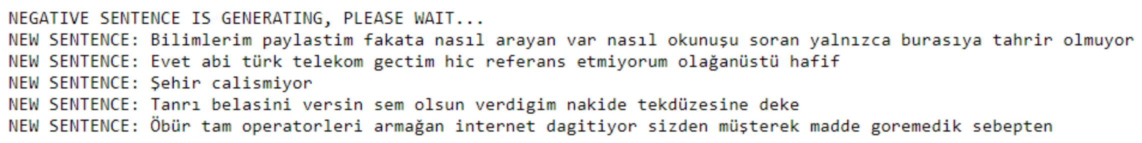
2- find_synonyms()

```
def find_synonyms(sentence):
    global new_sentence
    sentence = sentence.translate(str.maketrans('', '', string.punctuation))
#    print("ORIGINAL SENTENCE: " + sentence)
    return_sentence = ""
    sentence_words = []
    sentence_for_synonym_search = []

    if __name__ == '__main__':

        ZEMBEREK_PATH: str = join('..', '..', 'bin', 'zemberek-full.jar')

        TurkishMorphology: JClass = JClass('zemberek.morphology.TurkishMorphology')
        Paths: JClass = JClass('java.nio.file.Paths')

        morphology: TurkishMorphology = TurkishMorphology.createWithDefaults()

    for i in sentence.split():
        sentence: str = i
        analysis: java.util.ArrayList = morphology.analyzeSentence(sentence)

        results: java.util.ArrayList = (
            morphology.disambiguate(sentence, analysis).bestAnalysis()
        )

        #add sentence just subject, adjective, noun and verb from original sentence
        for j, result in enumerate(results, start=0):
            x = (str(result)).split(":")[1]
            y = x.split("]")[0]
            if y == "Pron,Pers": #Select Subject Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_for_synonym_search.append(y)
            elif y == "Adj": #Select Adj Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_words.append(sentence.split()[j])
                sentence_for_synonym_search.append(y)
            elif y == "Noun": #Select Noun Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_words.append(sentence.split()[j])
                sentence_for_synonym_search.append(y)
            elif y == "Verb": #Select Verb Data
                x = (str(result)).split(":")[0]
                y = x.split("[")[1]
                sentence_words.append(sentence.split()[j])
                sentence_for_synonym_search.append(y)

    word_new = ""

    for kelime,original_word in zip(sentence_for_synonym_search,sentence_words):
        kelime_new = kelime.replace('ö','o')
        kelime_new = kelime_new.replace('ı','i')
        kelime_new = kelime_new.replace('ü','u')
        kelime_new = kelime_new.replace('ç','c')
        kelime_new = kelime_new.replace('ş','s')
        kelime_new = kelime_new.replace('ğ','g')
        kelime_new = kelime_new.strip()
#        print("KELIME_NEW: " + kelime_new)
        #find synonyms from site
#        try:

        if kelime not in verb_list:
            user_agent = 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.7) Gecko/2009021910 Firefox/3.0.7'

            url = "http://www.es-anlam.com/kelime/" + kelime_new
            headers={'User-Agent':user_agent,}

            request=urllib.request.Request(url,None,headers) #The assembled request
            response = urllib.request.urlopen(request)
            soup = BeautifulSoup(response, "html.parser")
            ana = soup.find('h2')
            alt=ana.find('strong')
            alt = str(alt)
            alt = (alt.split(">")[1]).split(",")[0]
            alt = alt.split("<")[0]
            alt = alt.strip()

            if alt == "BULUNAMADI !":
                new_sentence = new_sentence + " " + original_word
            else:
#                   new_sentence = new_sentence + " " + alt
                word_new = alt

                add_suffix(original_word, word_new)
#               print("NEW_SENTENCE IF: " + new_sentence)

        else:
            new_sentence += " " + original_word
#               print("NEW_SENTENCE ELSE: " + new_sentence)

    return new_sentence
```

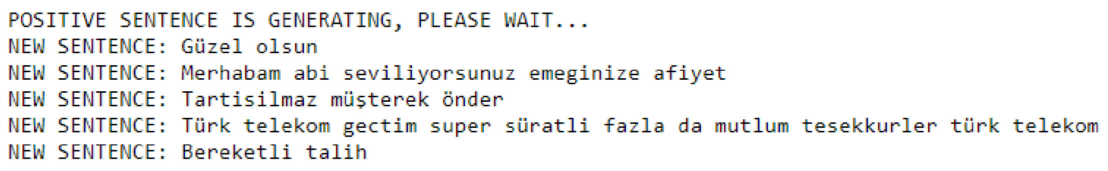
3- add_suffix()

```
def add_suffix(sentence_word, synonym_word):
    global new_sentence
    TurkishMorphology: JClass = JClass('zemberek.morphology.TurkishMorphology')
    WordAnalysis: JClass = JClass('zemberek.morphology.analysis.WordAnalysis')

    morphology: TurkishMorphology = TurkishMorphology.createWithDefaults()

    results: WordAnalysis = morphology.analyze(JString(sentence_word))

    number = ""
    possessive = ""
    case = ""
    word = ""

    for result in results:
        result = str(result)
        try:#set number
            if "A3sg" in result:
                number = "A3sg"
            elif "A3pl" in result:
                number = "A3pl"
        except:
            number = ""

        try: #set possessive
            if "P1sg" in result:
                possessive ="P1sg"
            elif "P2sg" in results:
                possessive = "P2sg"
            elif "P3sg" in result:
                possessive = "P3sg"
        except:
            possessive = ""

        try: #set case
            if "Dat" in result:
                case = "Dat"
            elif "Loc" in result:
                case = "Loc"
            elif "Abl" in result:
                case = "Abl"
        except:
            case = ""   #cases: List[JString] = [JString('Dat'), JString('Loc'), JString('Abl')]

        word = result.split(" ")[1]
        word = word.split(":")[0]

    morphology: TurkishMorphology = (
        TurkishMorphology.builder().setLexicon(synonym_word).disableCache().build()
    )

    try:
        item = morphology.getLexicon().getMatchingItems(synonym_word).get(0)

        if number != "" and possessive != "" and case != "":
            for result in morphology.getWordGenerator().generate(item, number, possessive, case):
                new_sentence += " " + str(result.surface)
        elif number != "" and possessive != "" and case == "":
            for result in morphology.getWordGenerator().generate(item, number, possessive):
                new_sentence += " " + str(result.surface)
        elif number != "" and possessive == "" and case != "":
            for result in morphology.getWordGenerator().generate(item, number, case):
                new_sentence += " " + str(result.surface)
        elif number == "" and possessive != "" and case != "":
            for result in morphology.getWordGenerator().generate(item, possessive, case):
                new_sentence += " " + str(result.surface)
        elif number == "" and possessive == "" and case != "":
            for result in morphology.getWordGenerator().generate(item, case):
                new_sentence += " " + str(result.surface)
        elif number == "" and possessive != "" and case == "":
            for result in morphology.getWordGenerator().generate(item, possessive):
                new_sentence += " " + str(result.surface)
        elif number != "" and possessive == "" and case == "":
            for result in morphology.getWordGenerator().generate(item, number):
                new_sentence += " " + str(result.surface)
        else:
            new_sentence += str(result.surface)
    except:
        pass
```

4- Outputs for module 4



```
NEGATIVE SENTENCE IS GENERATING, PLEASE WAIT...
NEW SENTENCE: Bilimlerim paylastim fakata nasıl arayan var nasıl okunuşu soran yalnızca burasıya tahrir olmuyor
NEW SENTENCE: Evet abi türk telekom gectim hic referans etmiyorum olağanüstü hafif
NEW SENTENCE: Şehir calismiyor
NEW SENTENCE: Tanrı belasini versin sem olsun verdigim nakide tekdüzesine deke
NEW SENTENCE: Öbür tam operatorleri armağan internet dagitiyor sizden müşterek madde goremedik sebepten
```



```
POSITIVE SENTENCE IS GENERATING, PLEASE WAIT...
NEW SENTENCE: Güzel olsun
NEW SENTENCE: Merhabam abi seviliyorsunuz emeginize afiyet
NEW SENTENCE: Tartisilmaz müşterek önder
NEW SENTENCE: Türk telekom gectim super süratli fazla da mutlum tesekkurler türk telekom
NEW SENTENCE: Bereketli talih
```