# N-Gram Analysis in Java

```java
public static String getFinalString(Path path) {

    Charset iso88599 = Charset.forName("ISO-8859-9");

    String corpus = "";

    try {
        corpus = Files.readString(path, iso88599);
        corpus = corpus.trim().replaceAll( regex: " +", replacement: " "); // remove extra spaces
        corpus = corpus.replaceAll( regex: "[\\r\\n]+", replacement: " "); // remove newline and cr
        corpus = corpus.replaceAll( regex: "([?!]?[.]+)|([«\u0093{(].{1}[»\u0094})])|([^a-zA-Z0-9ğüşäöıçİÂĞÜŞÖÇ])", replacement: " $0 ");

        corpus = corpus.trim().replaceAll( regex: " +", replacement: " ");// delete the first char in the string
    } catch (IOException ex) {
        System.out.println("File not found...");
    }

    return corpus.toLowerCase();
}
```

I used ISO-8859-9 charset for this work since Turkish characters is included in this charset. getFinalString function reads the whole txt file and separates punctuations and words. All extra spaces are trimmed and replaced with a single white space character.

After removing every unnecessary whitespace, all of the newline characters are removed thus a clean string is achieved. 3 regexes are used consecutively in order to separate punctuations. They are designed in a way that special cases such as !.. and …….. are handled, treated as one item. After that all of the extra spaces are again removed and clean corpus string is retrieved.

```java
public static Map<String, Integer> get_ngrams(String[] words, int stepSize) {
    Map<String, Integer> ngram_map = new HashMap<String, Integer>();

    for (int i = 0; i < words.length - stepSize - 1; i++) {
        String key = String.join( delimiter: " ", Arrays.copyOfRange(words, i, to: i + stepSize));
        if (ngram_map.containsKey(key)) {
            ngram_map.merge(key, value: 1, Integer::sum);
        } else {
            ngram_map.put(key, 1);
        }
    }
    return ngram_map;
}
```

Reference for n-gram: https://stackoverflow.com/questions/14617601/implementing-ngrams-in-python

I have altered the code so it works in Java and the items are held in maps.

According to the step sizes for loop shrinks and the tuple or 1 single item is retrieved from the words array. The words array is basically the items split by the “ “ delimiter of the corpus.

Since lists can’t be keys in maps as well I have cast them to strings. If the key exists as a key in the map, the value of the key is incremented. However if it does not exist in the map it is initial value is 1 and put into the map.

```
long startTime = System.currentTimeMillis();
// MAPS FOR TOP 100
Map<String, Integer> unimap = new HashMap<String, Integer>();
Map<String, Integer> bimap = new HashMap<String, Integer>();
Map<String, Integer> trimap = new HashMap<String, Integer>();

// PATHS FOR THE DOCUMENTS
Path path1 = Paths.get( first: "BİLİM İŞ BAŞINDA.txt");
Path path2 = Paths.get( first: "BOZKIRDA.txt");
Path path3 = Paths.get( first: "DEĞİŞİM.txt");
Path path4 = Paths.get( first: "DENEMELER.txt");
Path path5 = Paths.get( first: "UNUTULMUŞ DİYARLAR.txt");

// READ THE DOCUMENTS INTO STRINGS
String bib = getFinalString(path1);
String boz = getFinalString(path2);
String deg = getFinalString(path3);
String den = getFinalString(path4);
String unu = getFinalString(path5);

// FINAL STRINGS CONCATENATED
String ultimate_string = bib + " " + boz + " " + deg + " " + den + " " + unu;
String[] words = ultimate_string.split( regex: " ");
```

Main function holds the necessary variables such as maps and paths for the documents. The strings are retrieved after they are all cleaned and the ultimate string is created such as the corpus.

```java
// TOP 100's
// UNIMAP
List<Map.Entry<String, Integer>> top100_unigram = unimap.entrySet().stream()
        .sorted(comparing(Map.Entry::getValue, reverseOrder()))
        .limit(100)
        .collect(toList());
// BIMAP
List<Map.Entry<String, Integer>> top100_bigram = bimap.entrySet().stream()
        .sorted(comparing(Map.Entry::getValue, reverseOrder()))
        .limit(100)
        .collect(toList());
// TRIMAP
List<Map.Entry<String, Integer>> top100_trigram = trimap.entrySet().stream()
        .sorted(comparing(Map.Entry::getValue, reverseOrder()))
        .limit(100)
        .collect(toList());

System.out.println("TOP 100 FOR 1-GRAM: ");
System.out.println(top100_unigram + "\n");
System.out.println("TOP 100 FOR 2-GRAM: ");
System.out.println(top100_bigram + "\n");
System.out.println("TOP 100 FOR 3-GRAM: ");
System.out.println(top100_trigram);

long endTime = System.currentTimeMillis();

System.out.println("\nRuntime: " + (endTime - startTime) + " milliseconds");
```

As the last step the maps that are holding the ngrams are sorted and top 100 are gathered.

The average runtime over 20+ trials are 1.1 ms in Java.

The results are exactly the same as Python.

1-Gram Results:

```
TOP 100 FOR 1-GRAM:
[,=8073, .=6233, bir=3498, ve=1729, ;=1246, bu=1134, '=1124, :=938, de=920, da=842, -=824, ?=706, daha=699, için=677, ne=625, ...=620, !=614, (=524, )=523, ki=516, gibi
=516, kadar=496, o=463, her=461, çok=436, ama=436, diye=419, en=384, kendi=316, gregor=313, sonra=305, ?ey=284, zaman=264, bütün=258, hiç=246, büyük=235, ba?ka=234, ben
=224, onu=221, in=220, çünkü=218, de?il=216, un=212, öyle=212, ya=210, iyi=210, onun=206, olan=203, pek=200, malva=200, bile=196, kitap=194, yakov=191, do?ru=186, insan
=185, bölüm=177, böyle=171, hiçbir=168, mi=164, dedi=163, kendini=162, ona=161, «=158, vasili=156, »=154, olarak=152, içinde=152, ?=146, bana=146, az=145, var=145, ?=14
2, nas?l=137, iki=135, ancak=133, oldu?unu=129, montaigne=129, yaln?z=129, "=128, ?=127, bizim=124, m?=123, ayn?=122, yine=120, oldu?u=120, fazla=119, üzerinde=118, hem
=118, benim=117, göre=115, önce=115, bunu=114, fakat=110, hep=110, kar??=110, ile=109, yere=108, aras?nda=107, a=105, biri=104]
```

2-Gram Results:

```
TOP 100 FOR 2-GRAM:
[: -=371, . (=311, . bu=229, , bir=209, ' un=207, ( kitap=177, , bölüm=176, ki ,=174, gregor '=157, . bir=146, , bu=141, , ne=138, dedi .=133, ' in=128, bir ?ey=126, ,
ama=122, . ama=121, . -=116, ; ama=103, . ?=102, ; çünkü=96, o kadar=94, , daha=94, , en=88, , her=87, ' a=87, de?il ,=86, ya da=85, : «=81, malva '=80, ne kadar=77, ,
o=74, ! diye=72, ? diye=71, . malva=71, . yakov=70, ' n?n=70, gibi ,=66, . vasili=65, , gregor=65, 3 ,=65, kitap 3=65, , kendi=64, büyük bir=64, ' ya=61, . o=61, . ?=61
, ba?ka bir=60, da ,=58, hiç de=58, 2 ,=56, kitap 1=56, ? -=56, 1 ,=55, kitap 2=55, diye sordu=55, için ,=55, montaigne '=54, , sonra=54, ? in=53, bu kadar=53, ' da=52,
 . fakat=52, de?ildir .=52, hem de=52, ! dedi=52, ! »=51, ' e=51, mi ?=51, her ?eyi=50, , çünkü=50, . ne=50, böyle bir=49, . sonra=49, de ,=49, - ne=48, müdür bey=47, .
 «=47, daha çok=47, . ben=46, yakov '=46, sordu .=46, bir sesle=46, ' u=45, ( lucretius=45, lucretius )=45, . gregor=44, her ?ey=44, yine de=44, ' ?n=43, . her=43, , bü
tün=43, daha fazla=43, , ancak=43, , diye=42, ba?lad? .=42, , hem=41, olur .=41, » diye=41, gregor ,=41]
```

3-Gram Results:

```
TOP 100 FOR 3-GRAM:
[. ( kitap=154, gregor ' un=97, kitap 3 ,=65, ( kitap 3=65, 3 , bölüm=65, ( kitap 1=56, kitap 1 ,=55, ( kitap 2=55, 1 , bölüm=55, kitap 2 ,=54, 2 , bölüm=54, ! dedi .=4
7, ( lucretius )=45, diye sordu .=45, ? diye sordu=42, , gregor '=34, . ( lucretius=34, sesle : -=33, montaigne ' in=33, , dedi .=32, ( horatius )=31, bir sesle :=30, m
alva ' n?n=30, , hem de=29, ! » diye=29, " ö?renci "=28, diye ba??rd? .=28, : - ne=27, , bölüm 12=27, bölüm 12 )=27, , ne de=26, malva ' ya=25, gregor ' u=25, ... dedi
.=24, . ? ?=23, müdür bey '=23, gregor ' a=22, yakov ' un=22, , ? diye=21, ! diye ba??rd?=21, , ne kadar=20, ? dedi .=20, kar??l?k verdi .=20, malva : -=19, . ( horatiu
s=19, arilyn ? in=19, bey ' in=18, ba?ka bir ?ey=18, , malva '=18, diye kar??l?k verdi=18, sordu . -=18, vasili : -=17, yakov : -=16, dedi . -=16, malva ' y?=15, . greg
or ,=15, yakov ' a=15, vasili ' nin=15, z ? beryl=14, var ki ,=14, ? ? diye=14, . " ö?renci=14, . montaigne '=14, ( seneka )=14, ' un ,=14, , ya da=14, . vasili ,=14, (
 cicero )=14, ? ? ?=13, , sonra da=13, der ki ,=13, . gregor '=13, için de?il ,=13, bölüm 9 )=13, , bölüm 9=13, : - ben=13, , bölüm 13=13, , o da=13, bölüm 13 )=13, bir
 tav?rla :=12, kymil ? in=12, i?skender ' in=12, tav?rla : -=12, bir ?eydir .=12, . onun için=12, . yine de=11, ne var ki=11, ne yaz?k ki=11, ? » diye=11, de?il mi ?=11
, . yakov ,=11, ( ovidius )=11, dedi . yakov=11, . ( seneka=11, , demi? .=11, . malva '=11, . ancak ,=11, , yine de=10, seryojka : -=10, ? beryl ?=10]
```

# N-Gram Analysis in Python

In [3]:
```
import re
import time

start = time.time()
# dictionaries for n-grams
unidict = {}
bidict = {}
tridict = {}

# regexes to match ?.. !.. and other repeating chars such as ...
regexes = [
    r'[?!]?[.]+',
    r'[«•{(].{1}[»•})]',
    r'[^a-zA-Z0-9ğüşâöıçİÂĞÜŞÖÇ]'
]

# combine regexes into one ultimate regex
combined = "(" + ")|(".join(regexes) + ")"

# function to get the final version of the given string
def getFinalString (path):
    # read the file and join the paragraphs into one string

    corpus = open(path, encoding='ISO-8859-9').read()
    corpus = corpus.replace('\r', ' ')
    corpus = corpus.replace('\n', ' ').strip()


    # seperate punctuations and special characters (!.. ?.. (?) .....)

    corpus = re.sub(combined,' \g<0> ', corpus) #this will match the first regex it matches on the string
    corpus = re.sub(' +', ' ', corpus)
    return corpus

# paths for the documents and a list to keep them all
paths = []
path1 = "BİLİM İŞ BAŞINDA.txt"
path2 = "BOZKIRDA.txt"
path3 = "DEĞİŞİM.txt"
path4 = "DENEMELER.txt"
path5 = "UNUTULMUŞ DİYARLAR.txt"

paths.append(path1)
paths.append(path2)
paths.append(path3)
paths.append(path4)
paths.append(path5)
```

I used Jupyter Notebook for Python analysis. The operations are exactly the same as it was in the Java. Regexes are the same and they are combined into one big regex which is very useful since if the first one is matched we can obtain !.. as one item which means that all of the special cases are handled. getFinalString function works as the same in Java. It removes extra spaces, clears newlines and separates punctuations from words. Every document is read into the strings.

```python
# corpus to generate n-grams from
corpus = ''

# create ultimate final corpus (append each document to the corpus)
for path in paths:
    corpus = corpus + ' ' + getFinalString(path)
    corpus = corpus.strip()

# function for ngrams
def get_ngrams(wordlist,n):
    wordlist = [x.lower() for x in wordlist]
    wordlist = [x.replace(' +', ' ') for x in wordlist]
    ngrams = {}
    for i in range(len(wordlist)-(n-1)):
        if ' '.join(wordlist[i:i+n]) in ngrams:
            ngrams[' '.join(wordlist[i:i+n])] += 1
        else:
            ngrams[' '.join(wordlist[i:i+n])] = 1

    return sorted(ngrams.items(), key=lambda x: x[1], reverse=True)[:100] # sort them descending and return top 100

unidict = get_ngrams(corpus.split(' '), 1)
bidict = get_ngrams(corpus.split(' '), 2)
tridict = get_ngrams(corpus.split(' '), 3)

print('Top 100 1-grams for the corpus: \n')
print(unidict)
print('\n')
print('Top 100 2-grams for the corpus: \n')
print(bidict)
print('\n')
print('Top 100 3-grams for the corpus: \n')
print(tridict)
print('\n')
end = time.time()
print('Runtime in milliseconds: {0}'.format(end - start))
```

The corpus consists of the aggregation of all of the documents. Get_ngrams function works the same as it was in Java. The corpus is split by the delimiter “ ” and words are held in a list. All of the words are lowercase’d as well. Depending on the N number the tuple is gathered such as i:i+n meaning that if n is 1 the word itself will be an item, if n is 2 then the tuple will be added to the dictionary. The runtime is calculated at the end.

The average runtime over 20+ trials are 0.89 ms in Java.

Overall Python is faster than Java in this N-gram analysis.

1-Gram Results:

```
Top 100 1-grams for the corpus:

[(',', 8073), ('.', 6236), ('bir', 3502), ('ve', 1731), (';', 1246), ('bu', 1134), ("'", 1124), (':', 938), ('de', 922), ('da',
842), ('-', 824), ('?', 706), ('daha', 700), ('için', 677), ('...', 627), ('ne', 625), ('!', 614), ('(', 524), (')', 523), ('gi
bi', 516), ('ki', 516), ('kadar', 497), ('o', 463), ('her', 461), ('çok', 436), ('ama', 436), ('diye', 419), ('en', 384), ('ken
di', 316), ('gregor', 313), ('sonra', 305), ('şey', 285), ('zaman', 264), ('bütün', 258), ('hiç', 246), ('büyük', 235), ('başk
a', 234), ('ben', 224), ('onu', 221), ('in', 220), ('çünkü', 218), ('değil', 216), ('un', 212), ('öyle', 212), ('iyi', 210),
('ya', 210), ('onun', 206), ('olan', 204), ('malva', 201), ('pek', 200), ('bile', 197), ('kitap', 194), ('yakov', 191), ('doğr
u', 187), ('insan', 185), ('bölüm', 177), ('böyle', 171), ('hiçbir', 168), ('mi', 164), ('dedi', 163), ('kendini', 162), ('on
a', 161), ('«', 158), ('vasili', 156), ('»', 154), ('olarak', 152), ('içinde', 152), ('bana', 146), ('\x93', 146), ('var', 14
5), ('az', 145), ('\x94', 142), ('nasıl', 139), ('iki', 135), ('ancak', 134), ('olduğunu', 129), ('yalnız', 129), ('montaigne',
129), ('"', 128), ('\x92', 127), ('bizim', 124), ('aynı', 123), ('mı', 123), ('olduğu', 120), ('yine', 120), ('fazla', 119),
('hem', 118), ('üzerinde', 118), ('benim', 117), ('önce', 115), ('göre', 115), ('bunu', 114), ('ile', 110), ('karşı', 110), ('f
akat', 110), ('hep', 110), ('yere', 108), ('arasında', 107), ('a', 105), ('biraz', 104)]
```

## 2-Gram Results:

```
Top 100 2-grams for the corpus:

[(': -', 371), ('. (', 311), ('. bu', 229), (', bir', 210), ("' un", 207), ('( kitap', 177), (', bölüm', 176), ('ki ,', 174),
("gregor '", 157), ('. bir', 146), (', bu', 141), (', ne', 138), ('dedi .', 133), ("' in", 128), ('bir şey', 127), (', ama', 12
2), ('. ama', 121), ('. -', 116), ('; ama', 103), ('. \x93', 102), ('; çünkü', 96), (', daha', 94), ('o kadar', 94), (', en', 8
8), (', her', 87), ("' a", 87), ('değil ,', 86), ('ya da', 85), (': «', 81), ("malva '", 80), ('ne kadar', 77), (', o', 74),
('! diye', 72), ('? diye', 71), ('. malva', 71), ("' nın", 70), ('. yakov', 70), ('gibi ,', 66), ('. vasili', 65), (', gregor',
65), ('kitap 3', 65), ('3 ,', 65), ('büyük bir', 64), (', kendi', 64), ('. o', 61), ("' ya", 61), ('. \x94', 61), ('başka bir',
60), ('da ,', 58), ('hiç de', 58), ('2 ,', 56), ('? -', 56), ('kitap 1', 56), ('için ,', 55), ('diye sordu', 55), ('kitap 2', 5
5), ('1 ,', 55), (', sonra', 54), ("montaigne '", 54), ('bu kadar', 53), ('\x92 in', 53), ("' da", 52), ("' e", 52), ('hem de',
52), ('değildir .', 52), ('. fakat', 52), ('! dedi', 52), ('mi ?', 51), ('! »', 51), ('. ne', 50), (', çünkü', 50), ('her şey
i', 50), ('de ,', 49), ('böyle bir', 49), ('. sonra', 49), ('- ne', 48), ('daha çok', 47), ('. «', 47), ('müdür bey', 47), ('bi
r sesle', 46), ("yakov '", 46), ('. ben', 46), ('sordu .', 46), ("' u", 45), ('( lucretius', 45), ('lucretius )', 45), ('yine d
e', 44), ('her şey', 44), ('. gregor', 44), (', ancak', 43), ("' ın", 43), ('. her', 43), ('daha fazla', 43), (', bütün', 43),
('başladı .', 42), (', diye', 42), (', hem', 41), ('olur .', 41), ('» diye', 41), ('gregor ,', 41)]
```

## 3-Gram Results:

```
Top 100 3-grams for the corpus:

[('. ( kitap', 154), ("gregor ' un", 97), ('( kitap 3', 65), ('kitap 3 ,', 65), ('3 , bölüm', 65), ('( kitap 1', 56), ('( kitap
2', 55), ('kitap 1 ,', 55), ('1 , bölüm', 55), ('kitap 2 ,', 54), ('2 , bölüm', 54), ('! dedi .', 47), ('diye sordu .', 45),
('( lucretius )', 45), ('? diye sordu', 42), (", gregor '", 34), ('. ( lucretius', 34), ('sesle : -', 33), ("montaigne ' in", 3
3), (', dedi .', 32), ('( horatius )', 31), ('bir sesle :', 30), ("malva ' nın", 30), (', hem de', 29), ('! » diye', 29), ('diy
e bağırdı .', 28), ('" öğrenci "', 28), (': - ne', 27), (', bölüm 12', 27), ('bölüm 12 )', 27), (', ne de', 26), ("malva ' ya",
25), ("gregor ' u", 25), ('... dedi .', 24), ("müdür bey '", 23), ('. \x94 \x93', 23), ("yakov ' un", 22), ("gregor ' a", 22),
('! diye bağırdı', 21), (', \x94 diye', 21), (', ne kadar', 20), ('karşılık verdi .', 20), ('? dedi .', 20), ('malva : -', 19),
('. ( horatius', 19), ('arilyn \x92 in', 19), ('başka bir şey', 18), (", malva '", 18), ('diye karşılık verdi', 18), ('sordu .
-', 18), ("bey ' in", 18), ('vasili : -', 17), ('dedi . -', 16), ('yakov : -', 16), ("vasili ' nin", 15), ("yakov ' a", 15),
("malva ' yı", 15), ('. gregor ,', 15), ("' un ,", 14), ('. vasili ,', 14), ('var ki ,', 14), ('. " öğrenci', 14), (', ya da',
14), ('. montaigne '", 14), ('( cicero )', 14), ('( seneka )', 14), ('? \x94 diye', 14), ('z \x92 beryl', 14), ('için değil ,',
13), (', o da', 13), (', sonra da', 13), (': - ben', 13), ('. gregor '", 13), ('der ki ,', 13), (', bölüm 13', 13), ('bölüm 13
)', 13), (', bölüm 9', 13), ('bölüm 9 )', 13), ('? \x94 \x93', 13), ('bir şeydir .', 12), ('bir tavırla :', 12), ('tavırla : -
', 12), ("iskender ' in", 12), ('. onun için', 12), ('kymil \x92 in', 12), ('. ancak ,', 11), ('ne yazık ki', 11), ('. yine d
e', 11), ('. malva '", 11), ('dedi . yakov', 11), ('. yakov ,', 11), ('değil mi ?', 11), ('ne var ki', 11), ('? » diye', 11),
(', demiş .', 11), ('( ovidius )', 11), ('. ( seneka', 11), (', yine de', 10), ('. - ne', 10), ('ne var ?', 10)]
```