Tirana ésheté gírhmoné një hap përpata; po mendojmi për ditët e frohta dhe me shi, kur mund të kemi emergjencia civile. Këto janë të pashmangshme dhe pavarësisht histerisë, as kryetari i Bashkisë, as Kryeministri apo kushdo þetër nuk mundet ta ndalój shuan apo të rregullojë infrastrukturën e qëndërtuar ndèr vite, që janë krysisht ndërtime pa lëje büzë lumenjve apo në hapësira të tjera publike.

A világ legnagyobb fizetést hálózatának működési bevétele az amerikai gazdasági aktivitás két harmadát adó személyi fogyasztás folyamatos életének köszönhetően 14 százalékkal 4,86 milliárd dollárra emelkedett. Az eredményben az is szerepet játszott, hogy a Visa Inc. a múlt év közepén megpróbálta a Visa Europe Ltd. cégét. A működési költségek aleg változnak, 1,64 milliárd dollárt tettek këtu.

A Visa kártyákkal lebonyolított fizetések összege 9,8 százalékkal 1,93 ezer milliárd dollárra emelkedett, ennek 43 százaléka az Egyesült Államokra jutott.

「第8回日本ジオパーク全国大会 男鹿半島・大潟大会」が25日、3日間の日程で秋田県の男鹿市と大潟村で開催した。東北での全国大会開催は初めて。自治体関係者やガイド、研究者ら約千人が集まり、ハシド・カッシドーグなどの通じてジオパークの活用策を考えた。両市村と関連団体などでつくる実行委員会の主催。

Real-world problem: Language Detection

Creating features from text

1. Looking at the character sets: Latin, Cyrillic, Greek, CJK (Chinese, Japanese, Korean), etc. can provide a first categorization into language families.
2. Looking at one-letter, two-letter or three-letter words and their frequency in a text.
3. Character n-grams and their frequency.
4. Word n-grams and their frequency.

What are n-grams?


Word n-grams deals with sentences. “I like red cherries” will have as bigrams: “I like”, “like red”, “red cherries”.

N-grams are a common model for representing language in the field of Natural Language Processing (a subfield of Artificial Intelligence).
Comparing character bigrams in different languages

To notice:
• The top 3 bigrams for English cannot be found at all in the list of Spanish bigrams.
• The two lists have 14 bigrams in common out of 30 (less than half).
• The bigrams that are in common have different frequency. E.g., EN is 2.27 in Spanish and 1.13 in English.

Note: These bigrams were calculated from a large set of news stories. Because the word “the” is the most common word in English speech, that explains why the two bigrams “th” and “he” are at the top. If we use only the vocabulary of English words, the list will change. The most common bigram becomes “in”, because of the many words that start with “in” or that end in “ing”.

Back to CS 111

Question: How to build a program that identifies natural languages?
Answer: We create a “signature” for each known language by processing large amounts of text. This signature is composed of different features and their frequency distributions. Then, for new text, we compare its signature to that of known languages and pick the one that comes the closest.

Question: What does this problem have to do with CS 111?
Answer: While we cannot build the entire program, we can create many of the features that would be part of the signature of a language.

English word length distribution

Problem: Given the dictionary of all English words, what is the distribution of words by length?
Solution 1 (requires two separate loops)
1. Iterate over the list of words to find the length of each word and store it into a new list. [Accumulation in a list via mapping]
2. Iterate over the list of lengths and store it into a dictionary to keep track of the number of times we encounter each length. [Accumulation via a dictionary]

Solution 2 (requires one loop)
1. Iterate over the list of words to find the length of each word and immediately store it into a dictionary. [Accumulation via a dictionary]

Visualization of English word length distribution. It resembles a bell curve (normal distribution) that we found often in nature.
Fun with statistics

In Statistics, it is common to describe a dataset (e.g., the list of the length of all English words) in terms of descriptive statistics: the mean, the median, the mode (the value that occurs the most), the variance, the standard deviation, etc. All these statistics can be calculated with the operations we have been learning.

- The **mean** is the sum of all list elements divided by the length of the list. (sum = accumulation to a number)
- The **median** is the middle element of a sorted list.
- The **mode** is the most frequent element (i.e., the max value from the frequency dictionary.)
- The **variance** is the sum of the squares of the difference of each item to the mean.
- The **standard deviation** is the square of the variance.

### Try it out
Using the `lengthsList` and `lengthsDct` from the previous slide, you can practice calculating these statistics with Python code.

### Building character n-grams

Unigrams: 'b', 'o', 's', 't', 'o', 'n'

```python
word = 'boston'
list(word)
```

Bigrams: 'bo', 'os', 'st', 'to', 'on'

```python
""".join(pair) for pair in zip(word, word[1:])
```

Trigrams: 'bos', 'ost', 'sto', 'ton'

```python
""".join(trple) for trple in zip(word, word[1:], word[2:])
```

### Mutating Dictionaries via aliasing

A dictionary can be mutated via aliasing.

```python
def makeNgrams(ngramList, ngramDict):
    """Given a list of items and a dictionary, update the counts of the dictionary keys."
    for ngram in ngramList:
        ngramDict[ngram] = ngramDict.get(ngram, 0) + 1
```

Concepts in this slide:
- The function `zip` can be used with strings and returns a list of tuples.
- A dictionary can be mutated via aliasing.
Analyzing the Results

- Predict what will be the max lengths for the `unigramDct`, `bigramDct`, and `trigramDct`: ____, ____, ____
- Do you expect that all dictionaries will have that max length? Explain.
- Predict the top 3 unigrams, top 3 bigrams, and top 3 unigrams?
- How to write a function `sortByFreq` that given a frequency dictionary will return the sorted list (in descending order) of its items, based on the value of each (key, value) item? Hint: use `lambda`.
- Which will be more frequent (have the highest values): the top unigrams, the top bigrams, or the top trigrams?

Dictionary Comprehension

Very much like list comprehension: use `{}` instead of `[]` and create pairs with the colon syntax, e.g., `aKey: aValue`.

Syntax: `{ aKey: aValue for aKey in sequence}`

Example: Write a dictionary comprehension that pairs words with their lengths.

```
In [1]: wordsLst = 'the autumn is dragging its feet'.split()
In [2] {word: len(word) for word in wordsLst}
Out[2]: {'autumn': 6, 'dragging': 8, 'feet': 4, 'is': 2, 'its': 3, 'the': 3}
```

Important
We can use dictionary comprehension in situations when we want to start accumulation with a complex data structure (see next slide).

Accumulating in a dictionary of dictionaries

Problem: How can we create a dictionary that has two levels of keys? In the first level, each key is a unigram, in the second level the keys are bigrams that start with the unigram. [See example on the right.]

Solution 1: Assume we already have `bigramDct`:

```
from string import lowercase
# 'abcdefghijklmnopqrstuvwxyz'
# create the dict with unigrams as keys and empty dict as values
bigramsByFirstLetter = {char: {} for char in lowercase}
```

```
for bigram in bigramDct:
    unigram = bigram[0]
    # assign the second level of keys
    bigramsByFirstLetter[unigram][bigram] = bigramDct[bigram]
```

Accumulating in a dictionary of dictionaries [2]

Solution 2: We don’t have bigrams, we create them as we iterate over the list of words.

```
from string import lowercase  # 'abcdefghijklmnopqrstuvwxyz'
# create the dict with unigrams as keys and empty dict as values
bigramsByFirstLetter = {char: {} for char in lowercase}
```

```
for word in englishwords:
    # create list of bigrams from word
    bigramList = bigrams(word)
    # iterate over bigrams
    for bigram in bigramList:
        unigram = bigram[0]
        # access the bigram dict for easy reference
        bigramDict = bigramsByFirstLetter[unigram]
        # increase frequency counter
        bigramDict[bigram] = bigramDict.get(bigram, 0) + 1
```
Accumulating in a dictionary of lists.

**Problem:** Group words from `englishwords` based on their ending: words ending with ‘ed’, ‘ly’, ‘es’, etc.

**Solution Algorithm:**
1. Create an empty dictionary
2. Iterate over words and get the ending of each word.
3. Check to see if the ending is already in the dictionary using the method `get` with the default value an empty list.
4. Append the word to the list associated with its ending.

```python
wordsByEnding = {}
for word in englishwords:
    ending = word[-2:]
    wordsByEnding[ending] = wordsByEnding.get(ending, [])
    wordsByEnding[ending].append(word)
```

New collection type: `set`

Sets are collections that have a unique number of items. They are useful to find the number of unique values in lists.

Find unigrams in a phrase:

```python
set([char for char in list(phrase)])
```

In [3]: `phrase = 'the autumn is dragging its feet'`
In [4]: `set([char for char in list(phrase)])`
Out[4]: `{ ' ', 'a', 'd', 'e', 'f', 'g', 'h', 'i', 'm', 'n', 'r', 's', 't', 'u'}`

In [5]: `set([1, 2, 1, 2, 1, 2])`
Out[5]: `{1, 2}`

In [6]: `empty = set([])` # don’t use empty = {}
In [7]: `empty.add(3)` # add for sets is like append for lists
In [8]: `empty`
Out[8]: `{3}`