Sentence segmentation and tokenisation

Sentence segmentation

The Python shell

Try typing `python3` into your terminal and pressing return ⏎,

```
$ python3
Python 3.5.2+ (default, Aug  5 2016, 08:07:14)
[GCC 6.1.1 20160724] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

This is the *Python shell* (or *console*), any time you see `>>>` in the examples it means "run this code in the Python shell". You can use the shell to execute Python statements one by one. As before, when you see `$` it means run this command in your terminal on the command line.

To exit the shell you can type `CTRL + D`.

Basic structure of programs

Try typing (or copypasta) this program into a text file called `privet.py`:

```
import sys

print('Привет мир!')
```

Then run it using:

```
$ python3 privet.py
```

After that, try replacing the above with:

```
import sys

sys.stdout.write('Привет мир!')
```

What is the difference between using `print()` and `sys.stdout.write()` ?

Now try adding `\n` after the exclamation mark. What do you think `\n` might mean ?

Now try the following code:

```
import sys

text = sys.stdin.read()

sys.stdout.write(text)
```

Save it in a file called `cat.py` and try running it like this:

```
$ echo "Привет мир" | python3 cat.py
```

The `echo` command displays a line of text, and the `|` *pipe* symbol redirects the output of one command into the input of another.

Compare the output with the Unix command `cat`:

```
$ echo "Привет мир" | cat
```

Comments

In Python you can litter your code with comments. This is good, comments make it easier for other programmers to understand your code, and they remind you of what you were thinking when you wrote a particularly complicated sequence of statements. In Python comments are preceded by the *hash symbol* `#`.

```
import sys

# Read in everything from standard input
text = sys.stdin.read()

# Output everything to standard output
sys.stdout.write(text)
```

If you want your comments to span more than one line, you can either put a hash at the beginning of each line, or you can use `"""` as follows:

```
"""
This is my program that acts like
the Unix 'cat' command
"""
import sys

# Read in everything from standard input
text = sys.stdin.read()

# Output everything to standard output
sys.stdout.write(text)
```

Now you can run this command in the same way as the one above:

```
$ echo "Привет мир" | python3 cat.py
```

In your practical work, if your code doesn't work properly, but you have comments that say what it is supposed to do, you will get more marks than if your code doesn't work and you have no

comments.

Variables

When we did

```
text = sys.stdin.read()
```

above, we were assigning the result of a function (more on this later) to a variable called `text`. The outcome was that the program read all of the available input and stored it in the variable `text` before outputting it with the `write()` function.

We can store anything in a variable.

```
>>> a = 5
>>> a
5
```

This code stores the value `5` in a variable `a`. What happens if you assign something else to the same variable ?

Simple arithmetic

You can do simple arithmetic in the Python shell.

```
>>> a
5
>>> a/2
2.5
>>> b = a/2
>>> b
2.5
```

Try out the operators `+`, `/`, `*` and `-`, what do they do?

Loops

In computer programs we very often want to iterate over items (i.e. do the same thing to all or some of the items in a series), for example characters in a string. In Python we can use *loops* to accomplish this.

`for` loop

The `for` loop is probably the most basic way of looping in Python. Whenever you want to do the same thing for a number of items, you will probably use a `for` loop.

```
import sys

for c in sys.stdin.read():
    print(c)
```

What does this program do ?

```
import sys

text = sys.stdin.read()
for c in text:
    print(c)
```

How about this program ? How might you make the program output the text without a newline after each character ?

`while` loop

Your computer has a limited amount of memory, if we read in all the text in the input before processing it then we could easily run out of memory if we are processing large corpora. The `while` loop allows us to read in part of the data.

```
import sys

c = sys.stdin.read(1)
while c:
    print(c)
    c = sys.stdin.read(1)
```

We can read in one character at a time.

```
import sys

line = sys.stdin.readline()
while line:
    print(line)
    line = sys.stdin.readline()
```

Or we can read in one line at a time.

Flow control

`if` statement

The `if` statement allows you to put conditions in your program, for example, in the following program we read the input character by character and if the character is 'и', we increment the counter by one.

```
import sys

counter = 0

for c in sys.stdin.read():
    if c == 'и':
        counter = counter + 1
print(counter)
```

Save this program in a file called `counter.py` and run it with the input "Привет мир", what does it do ?

```
$ echo "Привет мир" | python3 counter.py
```

Now we could make the program count any vowel by giving a string containing the list of vowels to the `if` statement,

```
    if c in 'аэиоу':
        counter = counter + 1
```

Change your program, save it and run the previous command again.

In the first practical you found out that the Unix program `wc` can be used to count the number of lines, tokens and characters in the input. Now write a program called `wc.py` that counts the number of lines, tokens and characters in your corpus. (Hint: by counting the number of *space* characters you can count the number of tokens)

Compare the output of this program to the `wc` command. Then write a program to count the number of consonants and vowels in your corpus. Assuming that you can approximate the number of syllables by counting the number of orthographic vowels, extend the program to compute the average number of syllables per word in your corpus. What do you think are some potential drawbacks or inaccuracies of this method ?

String processing

A **string** is a sequence of characters (or more accurately bytes), you can find a good general reference to processing strings in Python here (https://docs.python.org/3/library/stdtypes.html#string-methods).

Assign a string to a variable, `s`

```
>>> s = 'Расположена на восточном побережье Балтийского моря.'
>>> s
'Расположена на восточном побережье Балтийского моря.'
```

Then write the following in the Python shell:

```
>>> s.find('о')
4
>>> s[4]
'о'
```

What does `find()` do ?

```
>>> len(s)
52
>>> s[52]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

What do you need to type to get the last character in a string ? What relation does this have with the length of the string ?

```
>>> s[51]
'.'
>>> s[-1]
'.'
>>> s[50]
'я'
>>> s[-2]
'я'
```

We can extract substrings from the string using the *slice* notation.

```
>>> s.find(' ')
11
>>> s[0:11]
'Расположена'
```

Here we get the first token by taking a slice from the beginning of the string to the first space.

```
>>> s.replace(' ', '$')
'Расположена$на$восточном$побережье$Балтийского$моря.'
>>> s.replace(' ', '$', 1)
'Расположена$на восточном побережье Балтийского моря.'
```

```
>>> s.split(' ')
['Расположена', 'на', 'восточном', 'побережье', 'Балтийского', 'моря.']
```

What would you need to type to get the output `'восточном'` ?

```
>>> ss = s.split(' ')
>>> ss
['Расположена', 'на', 'восточном', 'побережье', 'Балтийского', 'моря.']
>>> '|'.join(ss)
'Расположена|на|восточном|побережье|Балтийского|моря.'
```

In Python, what are two ways of getting from the string:

```
Расположена на восточном побережье Балтийского моря.
```

to the string:

```
Расположена|на|восточном|побережье|Балтийского|моря.
```

?

Sentence segmentation

Unless you're using a curated corpus that has been manually annotated, like the RNC, then most of the text you manage to pull off the web is going to be at best in lumps like this:

Общую схему течений Тихого океана определяют закономерности общей циркуляции атмосферы. Северо-восточный пассат северного полушария способствует возникновению Северо-Пассатного течения, пересекающего океан от Центрально-американского побережья до Филиппинских островов. Далее течение разделяется на два рукава: один отклоняется к югу и частью питает Экваториальное противотечение, а частью растекается по бассейнам индонезийских морей. Северный рукав следует в Восточно-Китайское море и, выходя из него южнее острова Кюсю, даёт начало мощному тёплому течению Куросио. Это течение следует на север до Японского побережья, оказывая заметное влияние на климат японского побережья. У 40° с. ш. Куросио переходит в Северное Тихоокеанское течение, следующее на восток к побережью Орегона. Сталкиваясь с Северной Америкой, оно разделяется на северную ветвь тёплого Аляскинского течения (проходящего вдоль материка до полуострова Аляска) и южную — холодного Калифорнийского течения (вдоль Калифорнийского полуострова, вливаясь в Северо-Пассатное течение, замыкая круг). В южном полушарии Юго-восточный пассат формирует Южное Пассатное течение, которое пересекает Тихий океан от берегов Колумбии до Молуккских островов. Между островами Лайн и Туамоту оно образует ответвление, следующее в Коралловое море и далее на юг вдоль берега Австралии, образуя Восточно-Австралийское течение. Основные массы Южного Пассатного течения восточнее Молуккских островов сливаются с южной ветвью Северо-Пассатного течения и совместно образуют Экваториальное противотечение. Восточно-австралийское течение южнее Новой Зеландии вливается в мощное Антарктическое циркумполярное течение, идущее из Индийского океана, пересекающее Тихий океан с запада на восток. У южного окончания Южной Америки это течение ответвляется на север в виде Перуанского течения, которое в тропиках вливается в Южное Пассатное течение, замыкая южный круг течений. Другая ветвь течения Западных ветров огибает Южную Америку под названием течения мыса Горн и уходит в Атлантический океан. Важная роль в циркуляции вод Тихого океана принадлежит холодному подповерхностному течению Кромвелла, протекающему под Южным Пассатным течением от 154° з. д. до района Галапагосских островов. Летом в восточной экваториальной части океана наблюдается Эль-Ниньо, когда тёплое слабосолёное течение оттесняет холодное Перуанское течение от берегов Южной Америки. При этом прекращается поступление

```
кислорода в подповерхностные слои, что приводит к гибели планктона, рыб
и питающихся ими птиц, а на обычно засушливое побережье обрушиваются
обильные дожди, вызывающие катастрофические наводнения.
```

For downstream NLP tasks, with a few exceptions, we are mostly going to want to segment this first into sentences — or sentence-like chunks — and then into tokens. The first task is called **sentence segmentation**.

The objective is to take input like a block of text like the one above and produce output like the following:

Общую схему течений Тихого океана определяют закономерности общей циркуляции атмосферы.
Северо-восточный пассат северного полушария способствует возникновению Северо-Пассатного течения, пересекающего океан от Центрально-американского побережья до Филиппинских островов.
Далее течение разделяется на два рукава: один отклоняется к югу и частью питает Экваториальное противотечение, а частью растекается по бассейнам индонезийских морей.
Северный рукав следует в Восточно-Китайское море и, выходя из него южнее острова Кюсю, даёт начало мощному тёплому течению Куросио.
Это течение следует на север до Японского побережья, оказывая заметное влияние на климат японского побережья.
У 40° с. ш. Куросио переходит в Северное Тихоокеанское течение, следующее на восток к побережью Орегона.
Сталкиваясь с Северной Америкой, оно разделяется на северную ветвь тёплого Аляскинского течения (проходящего вдоль материка до полуострова Аляска) и южную — холодного Калифорнийского течения (вдоль Калифорнийского полуострова, вливаясь в Северо-Пассатное течение, замыкая круг).
В южном полушарии Юго-восточный пассат формирует Южное Пассатное течение, которое пересекает Тихий океан от берегов Колумбии до Молуккских островов.
Между островами Лайн и Туамоту оно образует ответвление, следующее в Коралловое море и далее на юг вдоль берега Австралии, образуя Восточно-Австралийское течение.
Основные массы Южного Пассатного течения восточнее Молуккских островов сливаются с южной ветвью Северо-Пассатного течения и совместно образуют Экваториальное противотечение.
Восточно-австралийское течение южнее Новой Зеландии вливается в мощное Антарктическое циркумполярное течение, идущее из Индийского океана, пересекающее Тихий океан с запада на восток.
У южного окончания Южной Америки это течение ответвляется на север в виде Перуанского течения, которое в тропиках вливается в Южное Пассатное течение, замыкая южный круг течений.
Другая ветвь течения Западных ветров огибает Южную Америку под названием

```
течения мыса Горн и уходит в Атлантический океан.
Важная роль в циркуляции вод Тихого океана принадлежит холодному
подповерхностному течению Кромвелла, протекающему под Южным Пассатным
течением от 154° з. д. до района Галапагосских островов.
Летом в восточной экваториальной части океана наблюдается Эль-Ниньо,
когда тёплое слабосолёное течение оттесняет холодное Перуанское течение
от берегов Южной Америки.
При этом прекращается поступление кислорода в подповерхностные слои, что
приводит к гибели планктона, рыб и питающихся ими птиц, а на обычно
засушливое побережье обрушиваются обильные дожди, вызывающие
катастрофические наводнения.
```

Here each sentence has been separated onto its own line.

A simple approach, and the one we are going to try first, would simply be to replace every full stop '.' followed by a space ' ' with a full stop and a newline character `\n`. Can you write a program to do that? The program should be called **`segmenter.py`** and you should add it to your GitHub repository.

Questions

- How should you segment sentences with semi-colon? As a single sentence or as two sentences? Should it depend on context?
- Should sentences with ellipsis... be treated as a single sentence or as several sentences?
- If there is an exclamation after the first word in the sentence should it be a separate sentence? How about if there is a comma?
- Can you think of some hard tasks for the segmenter?

Further reading

- §4.2.4 "Sentences" in Manning and Schütze (2000) *Foundations of Statistical Natural Language Processing* (MIT Press)

Tokenisation

Tracing program flow

As your programs get more complicated, it will become increasingly more challenging to visualise what they are doing mentally. A simple way to see what your program is doing is to use `print` statements which print to standard error, or `stderr`.

Take the program from the last class and add print statements as follows:

```
import sys

a = 0 # This is a variable for counting the number of vowels

for c in sys.stdin.read():
    print('Main loop:', a, c, file=sys.stderr)
    if c in 'аэиоуяеыёю':
        print('Found a vowel!', c, file=sys.stderr)
        a = a + 1

print(a)
```

Now call the program (here called `prog.py`) from the command line like,

```
$ head wiki.txt | python3 prog.py > wiki.output
```

What do you see ?

Adding `print` statements like this for “debugging” is common and is also a good way to find out what someone elses code is doing.

Character encoding and special characters

On your computer, characters are stored as numbers, for example the word *cat* is stored as `0x63 0x61 0x74` and the word *мир* is stored as `0x43c 0x438 0x440`. Try the following:

```
>>> hex(ord(u'q'))
'0x71'
>>> hex(ord(u'я'))
'0x44f'
```

Now open up your character map (for example `gucharmap`) and search for `044f`, what do you find ? Now try:

```
>>> hex(ord(u'e'))
'0x435'
>>> hex(ord(u'e'))
'0x65'
```

What do you notice, what can you say about these two symbols ?

Why is all this important ? Well, there are a couple of instances when you need to know about this stuff. One is when you are processing text from the web and there is the wrong encoding for some symbol (e.g. the character looks ok, but has the wrong underlying numerical representation). This can have various reasons:

- Misconfigured *optical-character recognition* (OCR) software that detects Cyrillic characters as Latin characters
- Misconfigured keyboards that output the wrong characters

Sometimes when you get some text and your program is not behaving properly, for example the tokenisation messes up or your morphological analyser doesn't find the word but it is in the dictionary, it could be because of this.

It is also worth noting that this applies to spacing symbols in addition to characters. For example, there is a character *non-breaking space*. The normal spacing character is `0x20`, but non-breaking space is `0xa0`. Depending on the program or library you use these may be treated as different characters or not.

If that wasn't enough, you may also come across the issue of pre-composed versus composed characters. For example if you see the character *ë* it could be encoded in two ways, either with the single character `0x451` or with two characters `0x435 0x308`. The character `0x308` is a *combining character* which means that it combines with the previous character to make a new

character.

```
>>> hex(ord(u'ë'))
'0x451'
>>> hex(ord(u'ë'))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ord() expected a character, but string of length 2 found
```

You will see this when you try and look up the numerical value of the character, it throws and error because the function expects a single character string, but you give it two characters (the letter *e* and the combining diaeresis).

Now try:

```
>>> [hex(ord(c)) for c in u'ë']
['0x435', '0x308']
```

When you process each character in the string separately, it works. The most important thing to remember about character encoding when corpus processing is that what you see might not have the same underlying representation to that which you expect.

Lists and tuples

You have already come across something that looks like a list in the second class. Like lists, a string is a kind of sequence, that you can iterate over and address by index. Furthermore, when we split the string using `split()` the function returned a list of strings.

However, lists can include more than just strings. For example, lists can store fixed-length lists of values that we call *tuples*.

```
>>> spisok = []
>>> spisok.append((1, 'а'))
>>> spisok.append((2, 'б'))
>>> spisok.append((3, 'в'))
>>> spisok
[(1, 'а'), (2, 'б'), (3, 'в')]
>>> spisok[1]
(2, 'б')
>>> spisok[1][0]
2
>>> spisok[1][1]
'б'
```

Lists can also contain other lists, which are addressed in the same way as tuples:

```
>>> spisok = []
>>> spisok.append(['a', 'б', 'в'])
>>> spisok.append(['a', 'b', 'v'])
>>> spisok
[['a', 'б', 'в'], ['a', 'b', 'v']]
```

When you are embedding lists within other lists it can be easy to get confused about the indexing, so note how `append()` works differently when applied to the outer list `spisok.append('г')` and the inner list `spisok[0].append('г')`.

```
>>> spisok[0].append('г')
>>> spisok
[['a', 'б', 'в', 'г'], ['a', 'b', 'v']]
>>> spisok.append('г')
>>> spisok
[['a', 'б', 'в', 'г'], ['a', 'b', 'v'], 'г']
```

List slicing works the same way as string slicing. So you can also refer to the end of the list with `-1`

```
>>> spisok
[['а', 'б', 'в'], ['a', 'b', 'v']]
>>> spisok[-1]
['a', 'b', 'v']
>>> spisok[-1][-1]
'v'
```

Formatting output with `print()`

```
>>> index = 1
>>> print('# sent_id = %d' % (index))
# sent_id = 1
```

```
>>> tekst = 'Привет мир!'
>>> print('# text = %s' % (tekst))
# text = Привет мир!
```

If you try and print a string as a numeral you will get an error. But note that you won't get the same error if you try and print a numeral as a string.

```
>>> print('# text = %s' % (index))
# text = 1
>>> print('# text = %d' % (tekst))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: %d format: a number is required, not str
```

Numeral values are stored to a certain amount of precision in Python, but usually you don't want to print them out to such a high precision. A rule of thumb is to think very hard before giving more than four decimals. A better rule is to think about what you are measuring and try and report as few decimal points as is informative.

```
>>> p = 1/3
>>> p
0.3333333333333333
>>> print('p = %.2f' % (p))
p = 0.33
>>> print('p = %.4f' % (p))
p = 0.3333
```

In Python, to report decimal points use the above syntax. The dot stands for the decimal point then the number of decimals and finally the `f` stands for "*float*".

Tokenisation

So, let's say our sentence segmenter is outputting our sentences on one per line, what is the next step we need to do before we start some serious processing ?

```
Это течение следует на север до Японского побережья, оказывая заметное
влияние на климат японского побережья.
Далее течение разделяется на два рукава: один отклоняется к югу и частью
питает Экваториальное противотечение, а частью растекается по бассейнам
индонезийских морей.
У 40° с. ш. Куросио переходит в Северное Тихоокеанское течение,
следующее на восток к побережью Орегона.
```

We need to split the text into tokens, that means treat each word as a word in its own right. This is more difficult than it may appear.

The objective is to take output like a block of text like the one above and produce output like the following:

```
# sent_id = 1
# text = Это течение следует на север до Японского побережья, оказывая
заметное влияние на климат японского побережья.
1   Это _   _   _   _   _   _   _
2   течение _   _   _   _   _   _   _   _
3   следует _   _   _   _   _   _   _   _
4   на  _   _   _   _   _   _   _   _
5   север   _   _   _   _   _   _   _   _
6   до  _   _   _   _   _   _   _   _
7   Японского   _   _   _   _   _   _   _   _
8   побережья   _   _   _   _   _   _   _   _
9   ,   _   _   _   _   _   _   _   _
10  оказывая    _   _   _   _   _   _   _   _
11  заметное    _   _   _   _   _   _   _   _
12  влияние _   _   _   _   _   _   _   _
13  на  _   _   _   _   _   _   _   _
14  климат  _   _   _   _   _   _   _   _
15  японского   _   _   _   _   _   _   _   _
16  побережья   _   _   _   _   _   _   _   _
17  .   _   _   _   _   _   _   _   _

# sent_id = 2
# text = Далее течение разделяется на два рукава: один отклоняется к югу
и частью питает Экваториальное противотечение, а частью растекается по
бассейнам индонезийских морей.
1   Далее   _   _   _   _   _   _   _   _
2   течение _   _   _   _   _   _   _   _
3   разделяется _   _   _   _   _   _   _   _
4   на  _   _   _   _   _   _   _   _
5   два _   _   _   _   _   _   _   _
6   рукава  _   _   _   _   _   _   _   _
7   :   _   _   _   _   _   _   _   _
8   один    _   _   _   _   _   _   _   _
9   отклоняется _   _   _   _   _   _   _   _
10  к   _   _   _   _   _   _   _   _
```

```
11  югу _   _   _   _   _   _   _   _
12  и   _   _   _   _   _   _   _   _
13  частью  _   _   _   _   _   _   _   _
14  питает  _   _   _   _   _   _   _   _
15  Экваториальное  _   _   _   _   _   _   _   _
16  противотечение  _   _   _   _   _   _   _   _
17  ,   _   _   _   _   _   _   _   _
18  а   _   _   _   _   _   _   _   _
19  частью  _   _   _   _   _   _   _   _
20  растекается _   _   _   _   _   _   _   _
21  по  _   _   _   _   _   _   _   _
22  бассейнам   _   _   _   _   _   _   _   _
23  индонезийских   _   _   _   _   _   _   _   _
24  морей   _   _   _   _   _   _   _   _
25  .   _   _   _   _   _   _   _   _

# sent_id = 3
# text = У 40° с. ш. Куросио переходит в Северное Тихоокеанское течение,
следующее на восток к побережью Орегона.
1   У   _   _   _   _   _   _   _   _
2   40° _   _   _   _   _   _   _   _
3   с. ш.   _   _   _   _   _   _   _   _
4   Куросио _   _   _   _   _   _   _   _
5   переходит   _   _   _   _   _   _   _   _
6   в   _   _   _   _   _   _   _   _
7   Северное    _   _   _   _   _   _   _   _
8   Тихоокеанское   _   _   _   _   _   _   _   _
9   течение _   _   _   _   _   _   _   _
10  ,   _   _   _   _   _   _   _   _
11  следующее   _   _   _   _   _   _   _   _
12  на  _   _   _   _   _   _   _   _
13  восток  _   _   _   _   _   _   _   _
14  к   _   _   _   _   _   _   _   _
15  побережью   _   _   _   _   _   _   _   _
16  Орегона _   _   _   _   _   _   _   _
```

```
17  .   _   _   _   _   _   _   _   _
```

Here every token has been separated onto its own line and numbered. This kind of formatting for text is a subset of the popular CoNLL-U (http://universaldependencies.org/format.html) format used by the Universal Dependencies project. There are ten *columns* in total.

A simple approach, and the one we are going to first, would simply be to replace every space ' ' with a newline character `\n`. Following that you can use the `replace()` function to preprocess the text to add a space before or after certain punctuation characters like ',', ':', '(' and ')'. Can you write a program to do that? The program should be called **`tokeniser.py`** and you should upload it to your GitHub.

It should be possible to run your segmenter and tokeniser in a pipeline as follows:

```
$ cat corpus/wiki.txt | python3 segmenter.py | python3 tokeniser.py
```

Questions

- Why should we split punctuation from the token it goes with ?
- Should abbreviations with space in them be written as a single token or two tokens ?
    - How about numerals like 134 000 ?
- If you have a case suffix following punctuation, how should it be tokenised ?
- Should contractions and clitics be a single token or two (or more) tokens ?