О. В. Бісікало, В. М. Севастьянов, І. В. Богач

ЛАБОРАТОРНИЙ ПРАКТИКУМ

з дисципліни «КОМП'ЮТЕРНА ЛІНГВІСТИКА»
для студентів спеціальності
126 – «Інформаційні системи та технології»

Міністерство освіти і науки України
Вінницький національний технічний університет

О. В. Бісікало, В. М. Севастьянов, І. В. Богач

ЛАБОРАТОРНИЙ ПРАКТИКУМ

**з дисципліни «Комп'ютерна лінгвістика»
для студентів спеціальності
126 – «Інформаційні системи та технології»**

Електронний лабораторний практикум
комбінованого (локального та мережного) використання

Вінниця
ВНТУ
2022

УДК 004.912(076)
Б65

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 3 від 27.10.2022 р.)

Рецензенти:
В. Б. Мокін, доктор технічних наук, професор
В. В. Ковтун, доктор технічних наук, професор
А. Я. Кулик, доктор технічних наук, професор

Бісікало, О. В.

Б65 Лабораторний практикум з дисципліни «Комп'ютерна лінгвістика» для студентів спеціальності 126 – «Інформаційні системи та технології» : електронний лабораторний практикум комбінованого (локального та мережного) використання [Електронний ресурс] / Бісікало О. В. , Севастьянов В. М., Богач І. В. – Вінниця : ВНТУ, 2022. – 102 с.

Лабораторний практикум містить докладні пояснення щодо змісту, завдань та загальних вимог до виконання лабораторних робіт з нормативної дисципліни «Комп'ютерна лінгвістика» для освітньої програми «Інтелектуальні інформаційні системи» за спеціальністю 126 – «Інформаційні системи та технології». Наводяться приклади оформлення та розв'язання індивідуальних завдань, змісту звітів, що враховують специфіку дисципліни. Лабораторний практикум буде корисними викладачам, аспірантам, магістрам, студентам, а також всім бажаючим вивчити технологічні основи програмного розв'язання задач комп'ютерної лінгвістики.

УДК 004.912(076)

ЗМІСТ

ВСТУП..... 4
Лабораторна робота № 1. Основи роботи у середовищі Python та налаштування лінгвістичного пакета NLTK..... 5
Лабораторна робота № 2. Символьні типи даних у мові Python..... 12
Лабораторна робота № 3. Вивчення основ роботи з пакетом NLTK..... 27
Лабораторна робота № 4. Програмний доступ до корпусів текстів 41
Лабораторна робота № 5. Комплексне оброблення лексичних ресурсів .. 59
Лабораторна робота № 6. Оброблення довільної текстової інформації.... 76
ЛІТЕРАТУРА..... 101

ВСТУП

Виконання лабораторних робіт з обов'язкової професійної навчальної дисципліни «Комп'ютерна лінгвістика» передбачено навчальним планом освітньої програми «Інтелектуальні інформаційні системи» за спеціальностю 126 – «Інформаційні системи та технології».

Дисципліна «Комп'ютерна лінгвістика» базується на базових поняттях мовознавства, основах програмування, математичній статистиці та окремих розділах дискретної математики, зокрема на формальних граматиках, теорії графів, математичних методах штучного інтелекту. Ця дисципліна безпосередньо пов'язана і доповнює такі базові дисципліни, як «Спецрозділи вищої математики», «Програмування», «Математична лінгвістика».

У процесі виконання лабораторних робіт студенти вивчають мову програмування Python та мають проявити здатність розв'язувати типові задачі в предметній області комп'ютерної лінгвістики на основі використання лінгвістичного пакета NLTK. Це передбачає визначення чисельних показників якості результатів розв'язання та, за необхідності, покращення їх до потрібного рівня, а також застосування отриманих результатів розв'язку для реалізації інтерактивної функції у певній інформаційній системі та/або технології.

Поточний та підсумковий контроль знань студентів проводиться шляхом фронтального, індивідуального чи комбінованого опитування студентів під час лабораторних занять, тестування або консультацій.

Python – об'єктно-орієнтована мова програмування високого рівня з потужними можливостями для обробки символьних типів даних. Це інтерпретаційна мова, яка дозволяє зекономити час, що витрачається на компіляцію. Інтерпретатор можна використовувати інтерактивно, що дозволяє експериментувати з можливостями мови і створювати фрагменти програм або тестувати окремі функції. Інтерпретатор – це програма, яка виконує Python програми.

Natural Language Toolk (NLTK) – набір Python бібліотек, які призначені для аналізу текстів, написаних природною мовою. NLTK дозволяє здійснювати символьний та статистичний аналіз текстів, створювати графічні звіти та містить детальну документацію і використовується в проектах з лінгвістики, штучного інтелекту, машинного навчання, автоматизації документообігу. Його можна застосовувати як навчальний комплекс, готовий аналітичний інструмент або платформу для створення прикладних систем обробки текстової інформації. NLTK вільно розповсюджується (http://www.nltk.org) і всі бажаючі можуть його встановити відповідно до інструкції розробників.

Виконані та захищені лабораторні роботи в паперовому вигляді зберігаються на кафедрі АІІТ згідно з номенклатурою справ протягом одного року, потім знищуються у встановленому порядку.

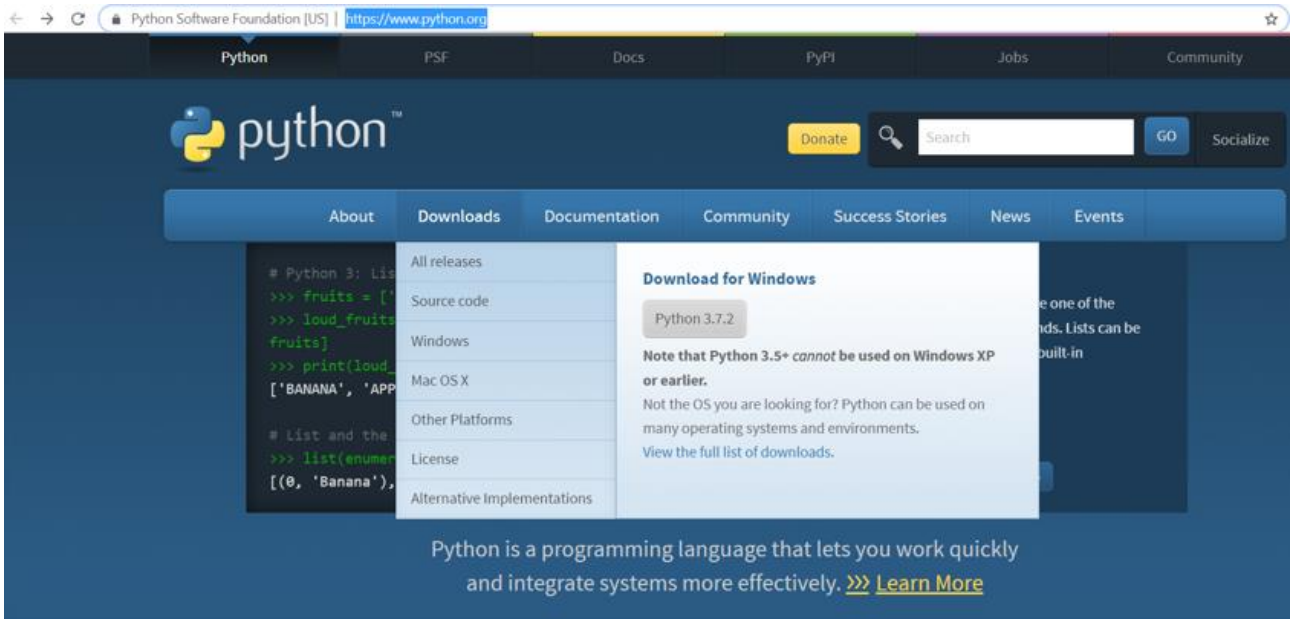
Лабораторна робота № 1. Основи роботи у середовищі Python та налаштування лінгвістичного пакетау NLTK

Мета роботи: ознайомитися з особливостями інтерпретатора мови програмування Python та методикою встановлення лінгвістичного пакету NLTK.

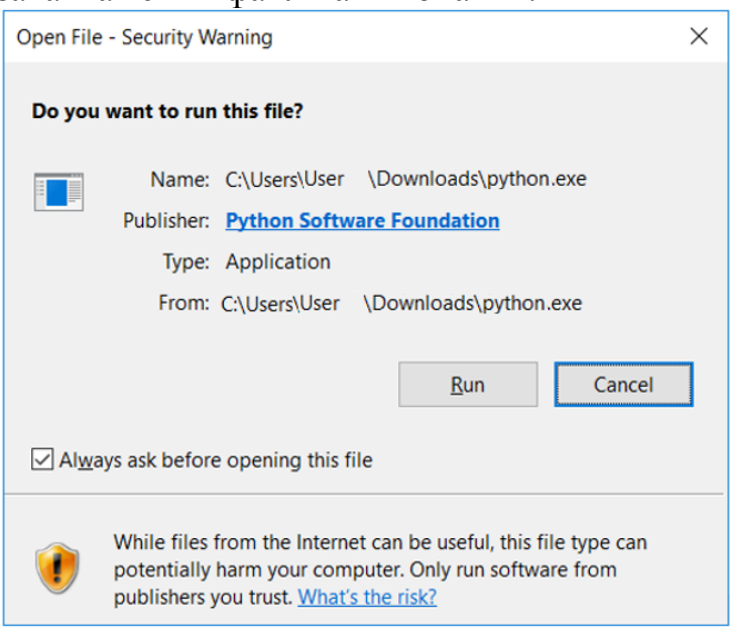
Порядок виконання роботи

1.1 Встановлення Python

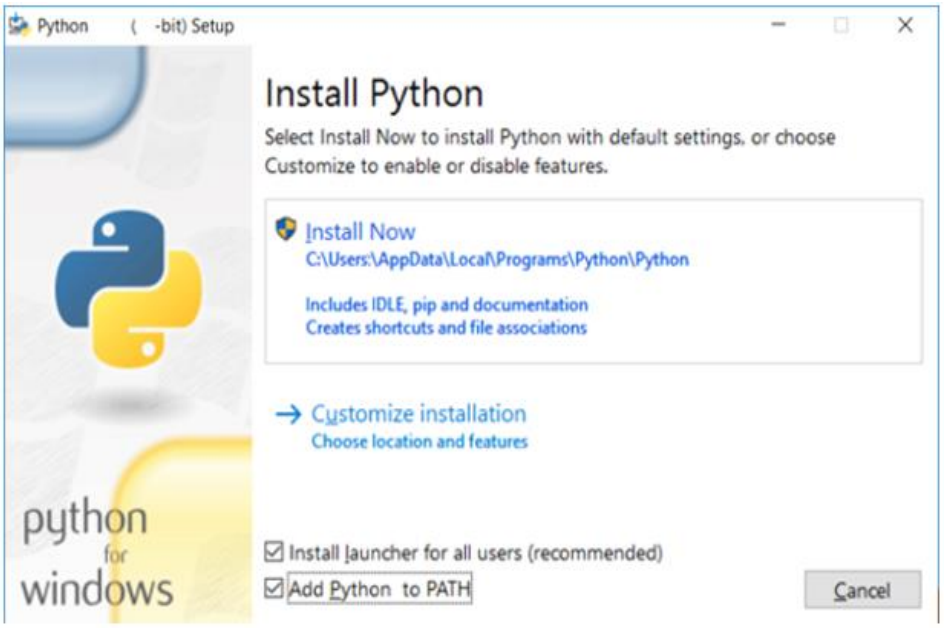
На сайті https://www.python.org/ вибрати та завантажити на свій комп'ютер потрібний файл для інсталяції Python.



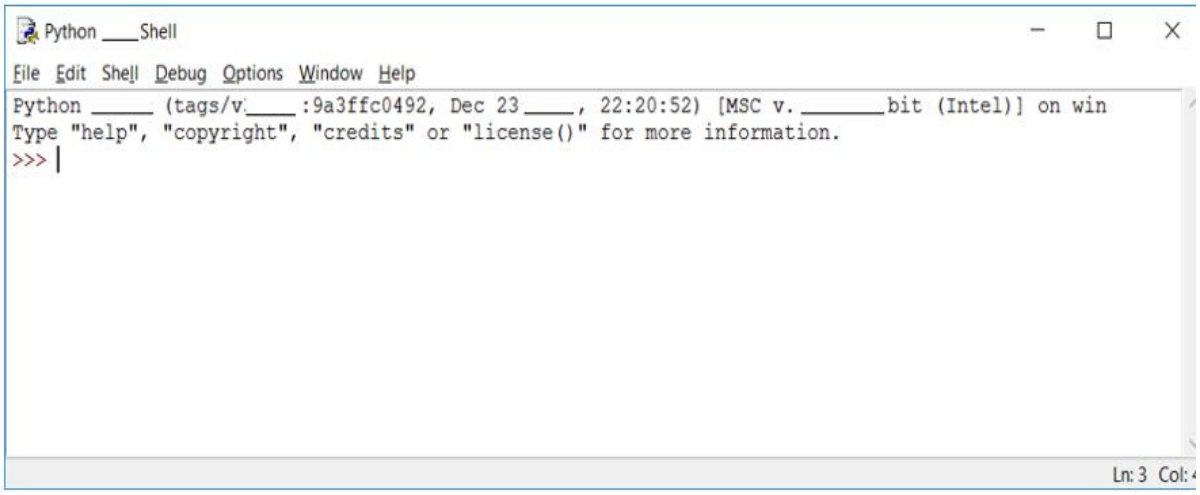
Запустити завантажений файл на виконання.



Не забути позначити опцію «Add Python v to PATH»

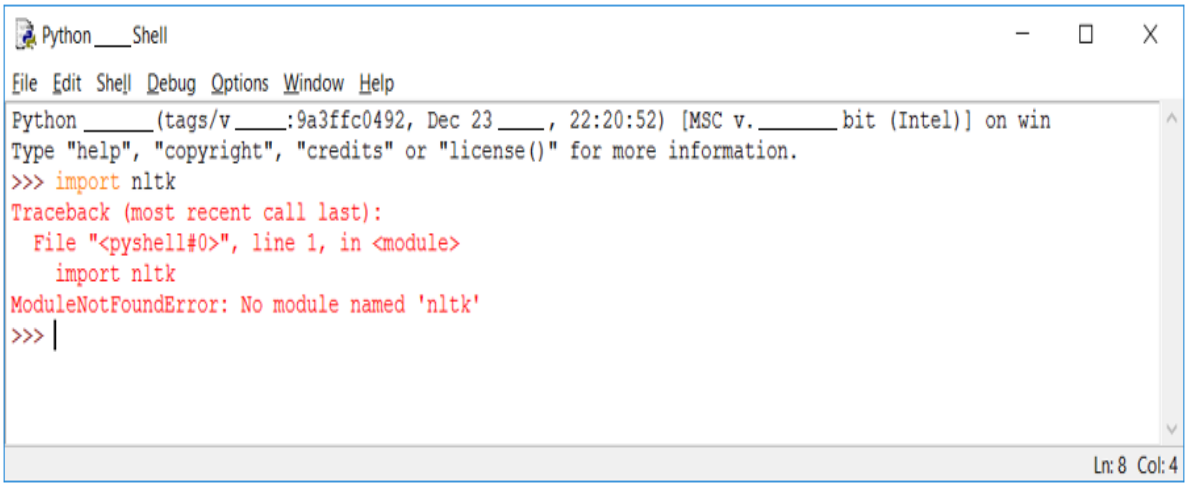


Запустити IDLE Python і пересвідчитись, що система працює.

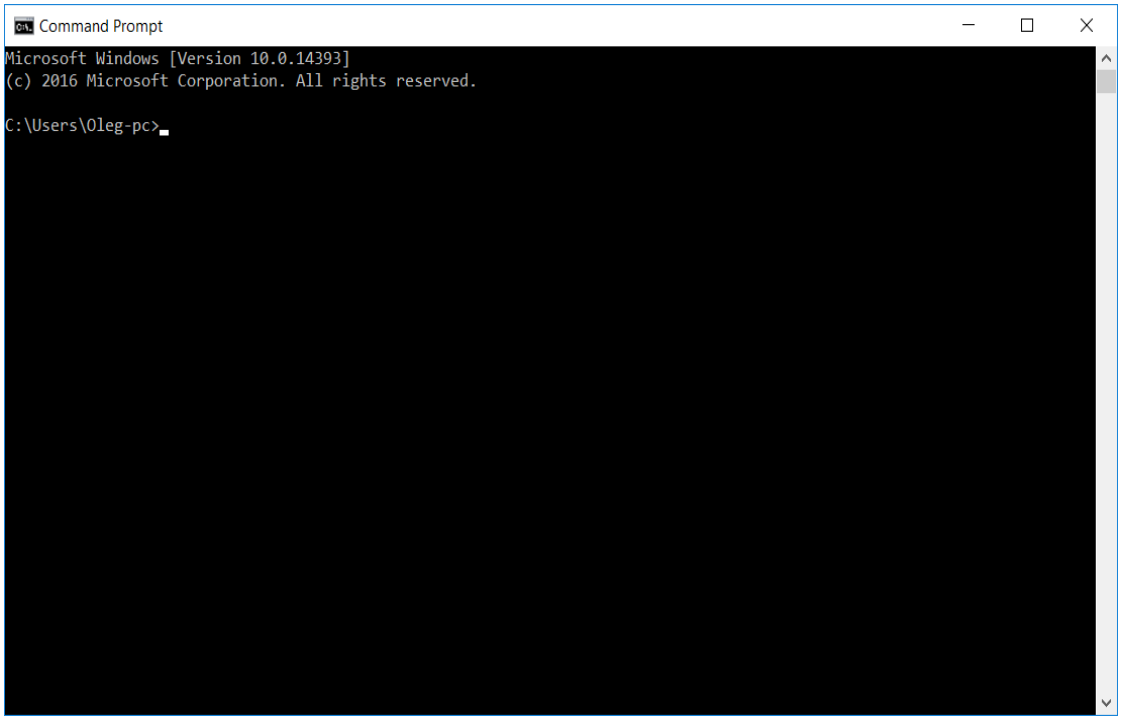


1.2 Встановлення лінгвістичного пакета NLTK

Якщо після команди
>>> import nltk
З’явиться повідомлення про помилку:
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import nltk
ModuleNotFoundError: No module named 'nltk'



Потрібно перевірити правильність встановлення шляхів path та завершити інсталяцію пакета NLTK – для цього перейдіть у в командний рядок Windows



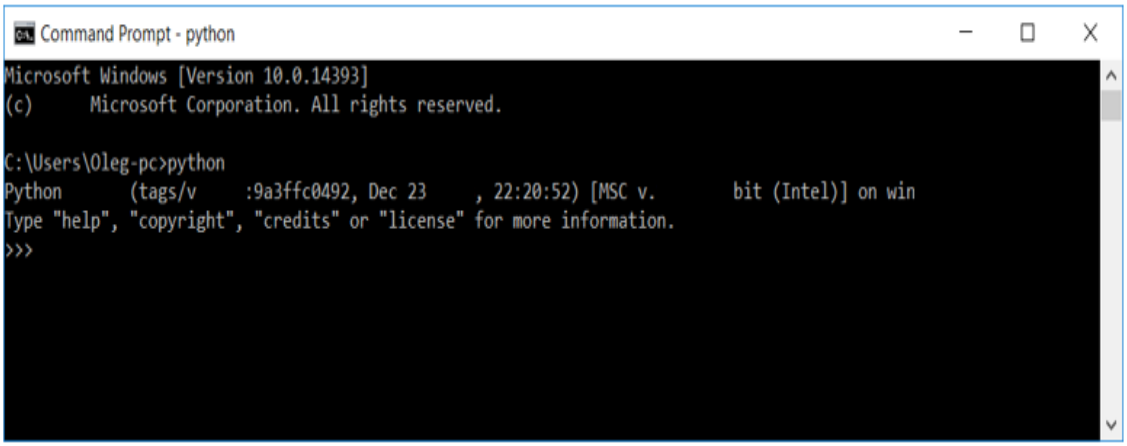
У командному рядку для перевірки потрібно ввести слово python, якщо покаже встановлену версію, то все нормально установлено – тоді для виходу з Python'а натискаємо ctrl+z та Enter і повертаємося у попередній каталог.

C:\Users\My-pc>python

Python _____ (tags/v_____:9a3ffc0492, Dec 23 _____, 22:20:52) [MSC v.1916 ___ bit (Intel)] on win___

Type "help", "copyright", "credits" or "license" for more information.

>>> ^Z



Далі у тому ж командному рядку вводимо команду **pip install numpy**

C:\Users\My-pc>pip install numpy
Collecting numpy
Downloading
https://files.pythonhosted.org/packages/d9/91/6829d324a2966b0f2b7da55b88d7492610e5c22c74a99f6da55df2f7b2d0/numpy-1.16.1-cp37-cp37m-win__.whl (10.0MB)
100% |████████████████████████████████| 10.0MB 90kB/s
Installing collected packages: numpy
Successfully installed numpy-1.16.1
You are using pip version 18.1, however version 19.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

Потім у рядку вводимо команду **python -m pip install --upgrade pip**

C:\Users\My-pc>python -m pip install --upgrade pip
Collecting pip
Downloading
https://files.pythonhosted.org/packages/d7/41/34dd96bd33958e52cb4da2f1bf0818e396514fd4f4725a79199564cd0c20/pip-19.0.2-py2.py3-none-any.whl (1.4MB)
100% collected packages
|████████████████████████████████| 1.4MB 514kB/s
Installing : pip
Found existing installation: pip 18.1
Uninstalling pip-18.1:
Successfully uninstalled pip-18.1
Successfully installed pip-19.0.2

На останок у командному рядку вводимо команду **pip install nltk**

C:\Users\My-pc>pip install nltk
Collecting nltk
Downloading https://files.pythonhosted.org/packages/6f/ed/9c755d357d33bc1931e157f537721efb5b88d2c583fe593cc09603076cc3/nltk-3.4.zip (1.4MB)
100% |████████████████████████████████| 1.4MB 514kB/s
Collecting six (from nltk)
Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl
Collecting singledispatch (from nltk)
Downloading https://files.pythonhosted.org/packages/c5/10/369f50bcd4621b263927b0a1519987a04383d4a98fb10438042ad410cf88/singledispatch-3.4.0.3-py2.py3-none-any.whl
Installing collected packages: six, singledispatch, nltk
Running setup.py install for nltk ... done
Successfully installed nltk-3.4 singledispatch-3.4.0.3 six-1.12.0
C:\Users\My-pc>

Тепер, після отримання таких повідомлень, інсталяція пакета NLTK завершена, отже можна виходити з командного рядку.

Робота з базовим англомовним підручником https://docs.python.org/3/

Заходимо в IDLE Python і вводимо команду **import nltk**

Python _____ (tags/v_____:9a3ffc0492, Dec 23 _____, 22:20:52) [MSC v._____ __ bit (Intel)] on win__

Type "help", "copyright", "credits" or "license()" for more information.

>>> **import nltk**

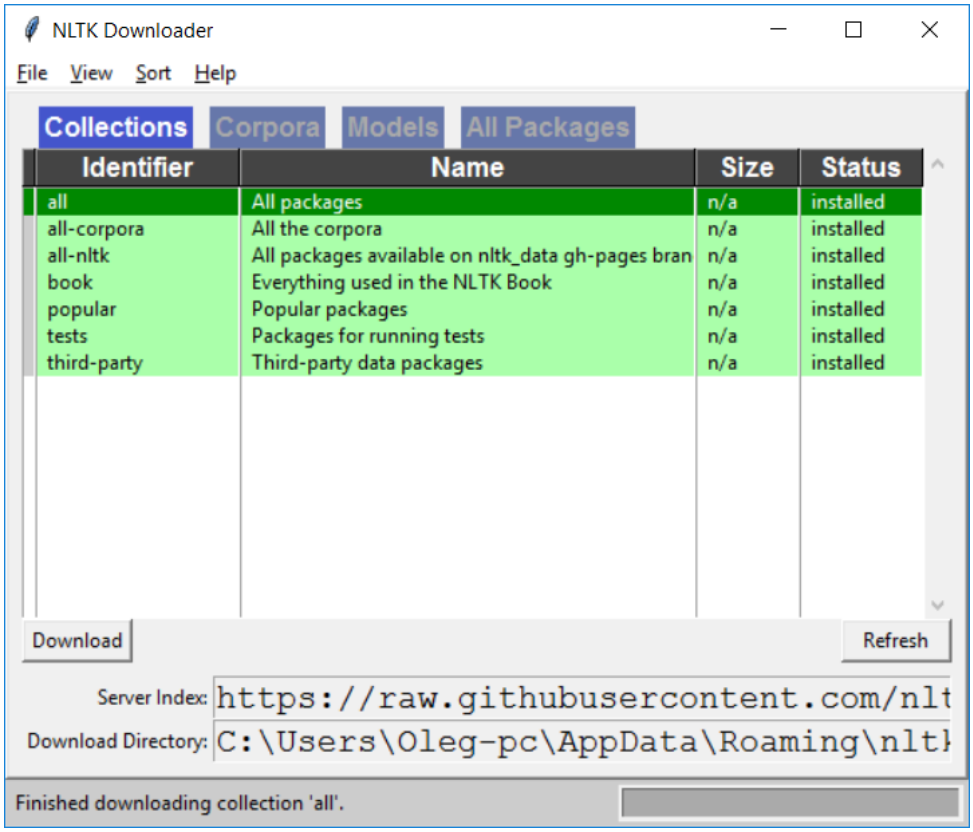
Після цього згідно з інструкцією *Chapter 1* підручника https://docs.python.org/3/ послідовно запускаємо команди:

>>> **nltk.download()**

showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

True

З'являється вікно, в якому потрібно вибрати компоненти для завантаження (обов'язково *book*, а всі пакети займають 100 Mb на диску):



Якщо все нормально завантажилося, то йти за інструкцією далі:

>>> **from nltk.book import ***
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908

Для перевірки правильності встановлення базового підручника пакета nltk.book застосуємо дві команди:

>>> **text1**
<Text: Moby Dick by Herman Melville 1851>
>>> **text2**
<Text: Sense and Sensibility by Jane Austen 1811>

Примітка: для самостійного вивчення мови програмування Python у мережі Інтернет існує значна кількість ресурсів вільного доступу; як приклади можна навести збірку підручників https://docs.python.org/3/, https://wiki.python.org/moin/PythonBooks, зокрема україномовні https://wiki.python.org/moin/UkrainianPythonBooks тощо.

Контрольні запитання

1. Що таке Python? У чому полягають особливості цієї мови програмування?
2. Основні характеристики IDLE Python?
3. Як встановити Python на комп'ютер?
4. Що таке NLTK? Дайте загальну характеристику цього лінгвістичного пакета.
5. Яка процедура інсталяції лінгвістичного пакету NLTK?
6. Як отримати доступ до ресурсів базового підручника пакета **nltk.book**?

Лабораторна робота № 2. Символьні типи даних у мові Python

Мета роботи: отримати навички побудови елементарних програм обробки текстової інформації на мові *Python* з використанням символьних типів даних.

Теоретичні відомості

Під час запуску будь-якого інтерпретатора Python ми бачимо інформацію про його версію, додаткову інформацію і запрошення >>> вводити оператори Python. У випадку використання базового інтерпретатора IDLE (Python's Integrated Development and Learning Environment) користувачам доступні додаткові зручності, зокрема у відображенні тексту програми на екрані.

2.1. Використання інтерпретатора Python як калькулятора

Спробуємо використати Python, як калькулятор.

```
>>> 3 + 2 * 5 - 1
12
>>>
```

Після того як ми натискаємо Enter, виконуються дії, інтерпретатор видає результат і чекає введення наступного оператора. Операція множення виконана правильно з дотриманням пріоритету виконання арифметичних дій. Можемо спробувати виконати інші операції множення і ділення:

```
>>> 3/3
1
>>> 1/3
0
>>>
```

У другому випадку отримано нуль, оскільке ділення в цьому випадку цілочисленне. Математичні операції будуть використовуватись в процесі роботи з лінгвістичними даними в Python. Якщо ввести вираз, у якому немає арифметичного змісту, то інтерпретатор видає повідомлення про помилку з вказанням місця помилки і її типу.

```
>>> 1 +
Traceback (most recent call last):
  File "<stdin>", line 1
    1 +
      ^
SyntaxError: invalid syntax
>>>
```

2.2. Рядки та змінні

2.2.1. Подання текстової інформації

Спробуємо працювати з текстом, безпосередньо ввівши його в командний рядок інтерпретатора.

```
>>> Hello World
Traceback (most recent call last):
  File "<stdin>", line 1
    Hello World
              ^
SyntaxError: invalid syntax
>>>
```

Отримали помилку. Текст або його фрагменти в програмах на Python подаються за допомогою *рядкового* (*string*) типу даних. Синтаксично текст як рядок має відділятися від решти програми лапками – одинарними (1), подвійними (2) або потрійними. Наприклад:

```
>>> 'Monty Python'                          (1)
'Monty Python'

>>> "Monty Python's Flying Circus"          (2)
"Monty Python's Flying Circus"

>>> 'Monty Python\'s Flying Circus'         (3)
"Monty Python's Flying Circus"

>>> 'Monty Python's Flying Circus'          (4)
>>> 'Monty Python's Flying Circus'
SyntaxError: invalid syntax
>>>
```

Якщо рядок містить одинарні лапки, необхідно використовувати лівий слеш перед апострофом (3) для того, щоб символ апострофа не розглядався як символ завершення рядка або застосовувати подвійні лапки (2). Якщо цього не зробити, то отримаємо помилку (4).

У більш простих випадках одинарні та подвійні лапки працюють однаково:

```
>>> 'Hello World'
'Hello World'
>>> "Hello World"
'Hello World'
>>>
```

У певних випадках рядок (з фрагментом тексту) може складатися з кількох рядків-компонентів. Мова Python забезпечує декілька способів роботи з ними. В наступному прикладі послідовність з двох підрядків об’єднується в один. Тут потрібно використовувати лівий слеш (3) або круглі дужки для того, щоб інтерпретатор знав – процес введення всього рядка ще не завершився після введення першого рядка.

```
>>> "Shall I compare you to a Summer's day? "\

    "You are more lovely and more temperate)."
```

Shall I compare thee to a Summer's day? Thou are more lovely and more temperate).

```
>>>
>>> ("Rough winds do shake the darling buds of May,"

        " And Summer's lease hath all too short a date:")
```

Rough winds do shake the darling buds of May, And Summer's lease hath all too short a date:

```
>>>
```

І, нарешті, застосування потрійних лапок дозволяє подавати рядок так, як він уведений з візуальним розділенням окремих підрядків:

```
>>> """Shall I compare thee to a Summer's day?
Thou are more lovely and more temperate:"""
Shall I compare thee to a Summer's day?
Thou are more lovely and more temperate:
>>> '''Rough winds do shake the darling buds of May,
And Summer's lease hath all too short a date:'''
Rough winds do shake the darling buds of May,
And Summer's lease hath all too short a date:
>>>
```

Спробуємо використати оператори додавання і множення для роботи з рядками.

```
>>> 'Hello' + 'World'
'HelloWorld'
>>>
```

Оператор додавання виконує забезпечує поєднання (конкатенацію) рядків. Додавання дозволяє створити новий рядок на основі двох існуючих, але цей оператор не додає пропуск між словами. Спробуємо поєднати три однакових рядки за допомогою операторів множення та додавання.

```
>>> 'Hi' + 'Hi' + 'Hi'
'HiHiHi'
>>> 'Hi' * 3
'HiHiHi'
>>>
```

Виконати самостійно! Виконайте таку послідовність дій і спробуйте пояснити отримані результати.

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1]
>>> b = [' ' * 2 * (7 - i) + 'very' * i for i in a]
>>> for line in b:
        print (line)
```

2.2.2. Змінні. Операції присвоювання

Для того, щоб програмно використовувати в подальшому значення даних рядкового типу, нам потрібно їх зберегти. Це досягається, якщо

розмістити дані в певному місці в пам'яті комп'ютера. Місце, де зберігаються певні значення даних, має назву змінної, якій обов'язково присвоюється унікальне (у межах окремої програми) ім'я. Значення кожній змінній задається операцією присвоювання, яку у мові Python позначає символ дорівнює ('='). Виконаємо цю операцію і переглянемо результат на екрані.

```
>>> msg = 'Hello World'          ①
>>> msg                          ②
'Hello World'                    ③
>>>
```

Також можемо використати оператори множення під час роботи зі змінними.

```
>>> msg = 'Hi'
>>> num = 3
>>> msg * num
'HiHiHi'
>>>
```

Можемо використовувати і операцію переприсвоювання.

```
>>> msg = msg * num
>>> msg
'HiHiHi'
>>>
```

В цьому випадку ми беремо значення змінної *msg*, виконуємо з ним певні дії, а результат знову присвоюємо змінній *msg*.

2.2.3. Друк і перегляд рядків

Для того, щоб вивести вміст змінної на екран, ми набираємо ім'я цієї змінної і натискаємо Enter.

```
>>> msg
'HiHiHi'
>>>
```

Тепер створимо файл *test.py*, який буде містити один рядок

```
msg = 'Hello World'
```

Відкриємо цей файл в IDLE і запустимо його на виконання командою *Run Module* з пункту меню *Run*. Результат буде такий:

```
>>> ================================ RESTART ================================
>>>
>>>
```

Для відображення значення змінної на екрані наша програма має містити команду *print*, її текст буде такий:

```
msg = 'Hello World'
print msg
```

У цьому випадку результат виконання програми буде видимий.

```
>>> ================================ RESTART ============================
>>>
Hello World
>>>
```

Аналогічно можна використовувати команду *print* в інтерпретаторі. Аргументами команди можуть бути як окремі змінні, так і їх послідовності, розділені комою.

```
>>> print msg
Hello World
>>>
>>> msg2 = 'Goodbye'
>>> print msg, msg2
Hello World Goodbye
>>>
```

2.3. Доступ до окремих символів рядка та виділення підрядків

2.3.1. Доступ до окремих символів рядка.

Змінна, яка містить рядок (тобто тип змінної – рядковий) є дуже важливою під час опрацювання текстів. Всі позиції в рядку є пронумерованими, починаючи з нуля. Для доступу до окремої позиції в рядку ми записуємо цю позицію в квадратних дужках. Наприклад:

```
>>> msg = 'Hello World'
>>> msg[0]
'H'
>>> msg[3]
'l'
>>> msg[5]
' '
>>>
```

Така операція має назву індексування, а номер позиції рядка, вказаний у квадратних дужках, називається індексом. Цей номер додається до фізичної адреси змінної, яка знаходиться в пам’яті (рис. 2.1).

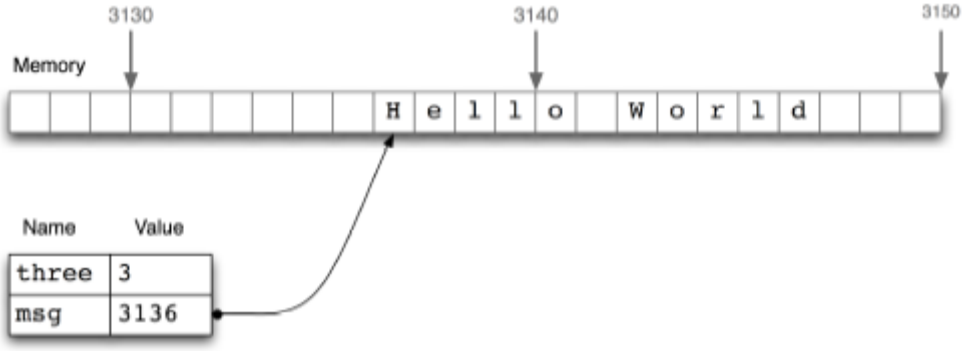


Рисунок 2.1 – Подання змінної у пам’яті комп’ютера

Тоді індекси рядка *msg* будуть мати такий вигляд (рис. 2.2.).

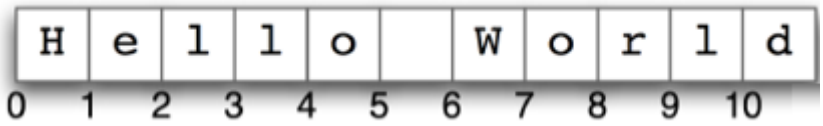


Рисунок 2.2 – Індексування рядка

У випадку, якщо ми хочемо отримати доступ до символа, індекс якого більший, аніж кількість символів у рядку, то отримуємо помилку.

```
>>> msg[11]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
>>>
```

Для визначення довжини рядка і, відповідно, максимального індекса, можна порахувати символи і відняти 1 або використати функцію *len()*.

```
>>> len(msg)
11
>>>
```

Загалом, функція – це іменований фрагмент програми, який виконує певну послідовність дій під час виклику і виконанняі функції. Функція *len()* є вбудованою функцією Python, виклик якої здійснюється за єдиним параметром – назвою рядка.

Можна використовувати індекси і з від'ємними значеннями.

```
>>> msg[-1]
'd'
>>>
```

Символ, який буде виведено, визначається як довжина рядка мінус індекс (рис. 2.3).

```
>>> msg[-3]
'r'
>>> msg[-6]
' '
>>>
```

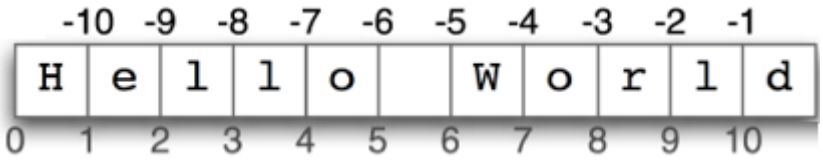


Рисунок 2.3 – Від'ємне індексування рядка

2.3.2. Доступ до підрядків

У випадку, коли ми хочемо отримати доступ до більш ніж одного символа, ми маємо в квадратних дужках вказати індекси першого і останнього символів, розділивши їх двокрапкою (рис. 2.4). Цей процес має назву *slicing (зріз)*.

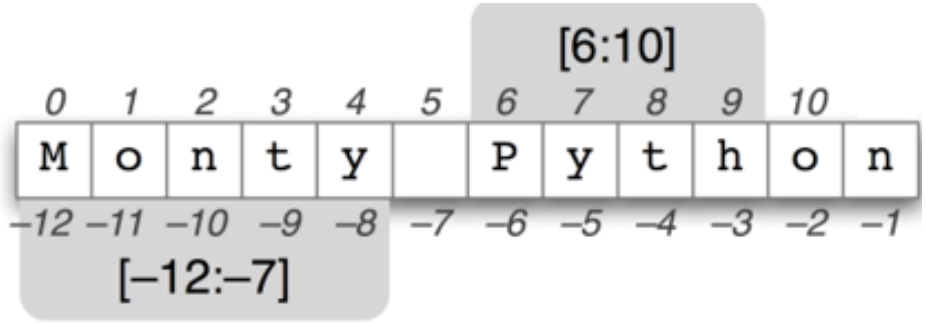


Рисунок 2.4 – Формування зрізів для доступу до підрядків

```
>>> msg[1:4]
'ell'
>>>
```

Звертаємо увагу, що ми отримали три символи, без четвертого, оскільки між індексами 1 та 4 знаходяться три позиції. Якщо ввести діапазон [0:11], то отримаємо весь рядок.

```
>>> len(msg)
11
>>> msg[0:11]
'Hello World'
>>>
```

Також можна використовувати від’ємні індекси та опускати перший чи останні індекс.

```
>>> msg[0:-6]
'Hello'
>>>
>>> msg[:3]
'Hel'
>>> msg[6:]
'World'
>>>
>>> msg[:-1]
'Hello Worl'
>>> msg[:]
'Hello World'
>>>
```

Задля доступу до підрядків можна задавати крок в зрізі. У наступному прикладі ми доступаємось до кожного другого символа з вказаного діпазона та отримуємо обернену послідовність.

```
>>> msg[6:11:2]
'Wrd'
>>> msg[10:5:-2]
'drW'
>>>
```

2.4. Рядки, послідовності, речення

Речення також можна подати як рядок, а тоді працювати з ним аналогічно тому, як було показано вище.

```
>>> sent = 'colorless green ideas sleep furiously'
>>> sent[16:21]
'ideas'
>>> len(sent)
37
>>>
```

Але важливо розуміти, що з реченням зручно і потрібно працювати як з послідовністю слів, а не окремих символів.

2.4.1. Списки

Lists (списки) – тип даних для опису послідовності символьних значень. В *Python* списки подаються як послідовність, записана через кому і у квадратних дужках.

```
>>> phrase1 = ['colorless', 'green', 'ideas']
>>> phrase1
['colorless', 'green', 'ideas']
>>>
```

Списки багато у чому подібні до рядків, але елементами списків, на відміну від рядків, можуть бути не тільки окремі символи, але й рядки та, навіть, інші списки.

```
>>> bat = ['bat', [[1, 'n', 'flying mammal'], [2, 'n', 'striking instrument']]]
>>>
```

Доступ до значень списку отримуємо аналогічно до рядків:

```
>>> len(phrase1)
3
>>> phrase1[0]
'colorless'
>>> phrase1[-1]
'ideas'
>>> phrase1[-5]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
>>>
```

Аналогічним чином значення списку індексуються і утворюються їх зрізи.

```
>>> phrase1[1:3]
['green', 'ideas']
>>> phrase1[-2:]
['green', 'ideas']
>>>
```

Списки, аналогічно до рядків, можуть поєднуватися.

```
>>> phrase2 = phrase1 + ['sleep', 'furiously']
>>> phrase2
['colorless', 'green', 'ideas', 'sleep', 'furiously']
>>> phrase1
['colorless', 'green', 'ideas']
>>>
```

Значення у списках можна змінювати на відміну від рядків, де значення змінити неможливо. Вміст списків можна змінювати в будь-який момент часу, а тому списки підтримують набір операцій або методів.

```
>>> phrase1[0] = 'colorful'
>>> phrase1
['colorful', 'green', 'ideas']
>>>
>>> msg[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
>>>
```

Використаємо щодо списку два методи – *sorting* (сортування) і *reversing* (обернення). Ці методи не створюють нові списки, а модифікують початкові.

```
>>> phrase2.sort()
>>> phrase2
['colorless', 'furiously', 'green', 'ideas', 'sleep']
>>> phrase2.reverse()
>>> phrase2
['sleep', 'ideas', 'green', 'furiously', 'colorless']
>>>
```

Тепер спробуємо використати ще два методи – *append* (додати) та *index* (індексувати).

```
>>> phrase2.append('said')
>>> phrase2.append('Chomsky')
>>> phrase2
['sleep', 'ideas', 'green', 'furiously', 'colorless', 'said', 'Chomsky']
>>> phrase2.index('green')
2
>>>
```

2.4.2. Обробка послідовностей

Часто під час роботи з лінгвістичними послідовностями різного типу виникає необхідність послідовно обробляти їх елементи. У *Python* це зручно здійснити, використовуючи цикл за параметром *for* (одна з структур управління мови програмування). Наприклад:

```
>>> for word in phrase2:
...     print len(word), word
5 sleep
5 ideas
5 green
9 furiously
9 colorless
4 said
7 Chomsky
```

Ця програма виконує оператор print len(word), word для кожного елемента зі списку. Такий процес називається ітераціями. Наведемо інший приклад *for* циклу.

```
>>> total = 0
>>> for word in phrase2:
...     total += len(word)
...
>>> total / len(phrase2)
6
>>>
```

У цій програмі визначається відношення між загальною кількість символів в елементах списку та кількістю елементів списку.

Можна написати *for* цикл і для роботи з символами (*char*) рядка:

```
>>> sent = 'colorless green ideas sleep furiously'
>>> for char in sent:
...     print char,
c o l o r l e s s  g r e e n  i d e a s  s l e e p  f u r i o u s l y
```

2.4.3. Кортежі

Tuples (кортежі) – подібний до списків тип даних, але, подібно до рядків, кортежі також не можемо змінювати.

```
>>> t = ('walk', 'fem', 3)
>>> t[0]
'walk'
>>> t[1:]
('fem', 3)
>>> t[0] = 'run'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
>>>
```

2.4.4. Форматування рядків

Під час використання оператора *print* можна передбачити певне форматування. Без форматування одержуємо таке:

```
>>> for word in phrase2:
...     print word, '(', len(word), '),',
sleep ( 5 ), ideas ( 5 ), green ( 5 ), furiously ( 9 ), colorless ( 9 ),
said ( 4 ), Chomsky ( 7 ),
```

У випадку виведення з форматуванням побачимо на екрані дещо інше:

```
>>> for word in phrase2:
...     print "%s (%d)," % (word, len(word)),
sleep (5), ideas (5), green (5), furiously (9), colorless (9),
said (4), Chomsky (7),
```

У цьому прикладі команда *print* має такий синтаксис: *format % values*. *Format* – це така частина команди, яка містить специфікатори формату, такі як *%s* і *%d*. Специфікатор *%s* вказує Python на те, що відповідна змінна є рядком або може бути конвертована в рядок, а специфікатор *%d* вказує, що відповідна змінна може бути конвертована в десяткове число. У *Values* записано кортеж значень, для якого визначено специфікатори формату.

Наступний приклад демонструє складніший випадок форматування:

```
>>> for word in phrase2:
...     print "Word = %s\nIndex = %s\n*****" % (word, phrase2.index(word))
...
Word = sleep
Index = 0
*****
Word = ideas
Index = 1
*****
Word = green
Index = 2
*****
Word = furiously
Index = 3
*****
Word = colorless
Index = 4
*****
Word = said
Index = 5
*****
Word = Chomsky
Index = 6
     *****
     >>>
```

2.4.5. Конвертування між рядками і списками

На практиці часто виникає потреба конвертації рядку, який містить послідовність записаних через пропуск слів у список, або, навпаки – список у рядок. Поданий нижче цикл перетворить список у рядок, а пропуск додаємо в циклі перед кожним зі слів.

```
>>> str = ''
>>> for word in phrase2:
...     str += ' ' + word
...
>>> str
' sleep ideas green furiously colorless said Chomsky'
>>>
```

Цю саму операцію ми можемо здійснити з використанням методу *string.join()*.

```
>>> import string
>>> phrase3 = string.join(phrase2)
>>> phrase3
'sleep ideas green furiously colorless said Chomsky'
>>>
```

Цю ж операцію можна здійснити, викликавши відповідний модуль:

```
>>> import string
>>> string.join(chomsky)
'colorless green ideas sleep furiously'
>>>
```

Еквівалентним є також і такий запис:

```
>>> from string import join
>>> join(chomsky)
'colorless green ideas sleep furiously'
>>>
```

Обернену операцію для перетворення рядка у список можна нескладно здійснити, побудувавши відповідний цикл. Проте в цьому випадку доцільніше використати метод *split()*, який розділить рядок за символами пробілу, або можна задати інший символ, за яким буде здійснюватися розділення.

```
>>> phrase3.split(' ')
['sleep', 'ideas', 'green', 'furiously', 'colorless', 'said', 'Chomsky']
>>> phrase3.split('s')
['', 'leep idea', ' green furiou', 'ly colorle', '', ' ', 'aid Chom', 'ky']
>>>
```

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями.
2. Виконати приклади, які використовуються в теоретичних відомостях.

3. Виконати вправи згідно з варіантом індивідуального завдання за своїм порядковим номером у списку групи (таблиця 2.1).

Таблиця 2.1

Варіант	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***
Номери завдань	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3
	5	6	7	8	9	10	5	6	7	8	9	10	5	6	7
	8	9	10	5	6	5	6	7	8	9	10	7	8	9	10
	11	12	13	14	15	16	17	18	11	12	13	14	15	16	17
	14	15	16	17	18	11	12	13	15	16	17	18	11	12	13
	19	20	21	22	23	24	25	26	21	22	23	24	25	26	22
	22	23	24	25	26	19	20	21	22	23	24	25	26	22	23

Варіант	***16***	***17***	***18***	***19***	***20***	***21***	***22***	***23***	***24***	***25***	***26***	***27***	***28***	***29***	***30***
Номери завдань	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
	8	9	10	5	6	7	8	9	10	5	6	7	8	9	10
	9	10	5	6	7	5	6	7	8	9	10	5	6	7	8
	18	11	12	13	14	15	16	17	18	11	12	13	14	15	16
	14	15	16	17	18	16	17	18	11	12	13	14	15	16	17
	23	24	25	26	19	20	21	22	23	24	25	26	23	24	25
	24	25	26	19	20	21	25	26	19	20	21	22	23	24	19

3.1. Створити змінну *msg*, присвоїти їй значення рядка, який відповідає імені та прізвищу студента.

3.2. Роздрукувати вміст змінної *msg* двома шляхами, перший – набравши назву змінної в інтерпретаторі, другий – використавши команду *print.*

3.3. Здійснити арифметичні операції з рядком *msg.*

3.4. Визначити новий рядок *hello*. Здійснити операцію *hello + msg.* Змінити рядок *hello,* додавши в його кінці символ пропуску і знову виконати операцію *hello+ msg.*

3.5. Використовуючи зрізи та операцію поєднання, змінити рядок *msg* до вигляду – ім’я, по батькові, прізвище студента.

3.6. Визначити рядок s=’colorless’. Використовуючи зрізи та операцію поєднання, змінити цей рядок до вигляду ‘colourless’.

3.7. Використовуючи зрізи, видаліть афікси у таких словоформах: *dish-es, run-ning, nation-ality, un-do, pre-heat*.

3.8. Спробуйте згенерувати IndexError, доступаючись до символів рядка з індексами менше 0.

3.9. Організуйте доступ до елементів рядка з певним кроком. Результати поясніть.

3.10. Поясніть результат виконання *msg[::-1]*.

3.11. Подайте прізвище, ім’я та по батькові як список рядків. Здійсніть різноманітні операції індексування, сортування та зрізів. Реалізуйте операцію доступу до окремих елементів списку та операцій з ними.

3.12. Подайте прізвище, ім’я та по батькові як список рядків. Розділіть речення на окремі елементи, межа розділу – голосна літера.

3.13. Подайте прізвище, ім’я та по батькові як список рядків. Використовуючи метод *.reverse()* та зріз *[::-1]*, змініть рядок. Результати поясніть.

3.14. Напишіть *for* цикл, який виведе на екран символи рядка *msg* по одному на рядок екрана.

3.15. Створіть список *phrase1*, який складається із значень ім’я, по батькові, прізвище студента. Що відбудеться у разі спроби ввести в інтерпретатор такий оператор *phrase1[2][2]*? Поясніть результат.

3.16. Створіть змінну *words*, яка містить список слів. Дослідіть операції *words.sort()* і sorted(words).

3.17. Створіть файл *test.py*, який містить рядок *msg*. Використайте такі оператори і поясніть отримані результати.

```
>>> from test import msg
>>> msg
```

3.18. Напишіть *for* цикл, який обробить *phrase1*, визначивши довжину кожного елемента, а результати збереже у новому списку *lengths*. (Створіть пустий список *lengths = []*. Далі використовуйте метод *append()* в тілі циклу для додавання довжин до списку).

3.19. Перетворіть рядок *msg* на список рядків, кожний з яких відповідає одному слову, використовуючи *split()* оператор без символа розділення та з такими символами розділення: подвійні лапки, табуляція, послідовність пропусків, послідовність табуляцій та пропусків.

3.20. Визначіть змінну *silly*, яка буде містити рядок ’newly formed bland ideas are inexpressible in an infuriating way’ і напишіть програму перетворення рядка на список рядків (кожне слово – це окремий рядок). Результати збережіть в змінній *bland.*

3.21. Напишіть програму, що створить рядок, в якому будуть записані інші символи всіх слів з рядка *silly*.

3.22. Напишіть програму перетворення списку рядків в один рядок.

3.23. Напишіть програму, яка надрукує слова із рядка *silly* за абеткою.

3.24. Використайте функцію index() таким чином: *’inexpressible’.index(’e’)*. Що станеться якщо виконати *’inexpressible’.index(’re’)*?

3.25. Визначіть позиції усіх слів у списку *phrase1*, використовуючи метод *index()*.

3.26. Визначіть змінну *silly*, яка буде містити рядок *’newly formed bland ideas are inexpressible in an infuriating way’*. Напишіть програму її перетворення в список *phrase*, який буде містити всі слова *silly* крім *‘in’*.

4. Підготувати і оформити звіт з виконання лабораторної роботи №2.

Контрольні запитання

1. Що таке *string*?
2. Навіщо використовувати потрійні лапки для позначення рядків?
3. Яку дію зі рядками виконують оператори додавання та множення?
4. Як ви розумієте термін індекс?
5. Що являє собою операція індексування?
6. Що забезпечує функція *len()*?
7. Як отримати доступ до першого та останнього елементів рядка?
8. Що таке *slicing*?
9. Що таке зріз?
10. Як задати крок у зрізі?
11. Що таке список?
12. Що можна вважати елементами списку?
13. Що забезпечують методи *sorting* та *reversing*. Як їх використати?
14. Що забезпечують методи *append* та *index*. Як їх використати?
15. Що потрібно для обробки лінгвістичних послідовностей?
16. Поняття циклу за параметром *for*, ітерації.
17. Що таке кортежі?
18. На що вказують специфікатори формату *(%s), (%d)*?
19. Що виконує операція *string.join()*?
20. Як конвертувати список у рядок?
21. Як конвертувати рядок в список?

Лабораторна робота № 3. Вивчення основ роботи з пакетом NLTK

Мета роботи: ознайомлення зі структурами керування програмами та класом FreqDist лінгвістичного пакета NLTK.

Теоретичні відомості

Виконання цієї та наступних лабораторних робіт починаємо з:
import nltk
*from nltk.book import **

3.1. Засоби керування логікою виконання програм

Програми, які згадувалися в лабораторній роботі № 2, доволі прості, але вже вони дозволяють опрацьовувати текстову інформацією та полегшити роботу людини шляхом автоматизації послідовностей дій, що повторюються. Але основна перевага програмування – це здатність програми приймати формальні рішення замість людини, зокрема, виконуючи інструкції, коли виконуються певні умови або послідовно обробляти дані до тих пір, доки не виконано задану умову. До таких засобів управління поведінкою програми (інша назва – структури керування) будемо відносити умовні вирази, засоби здійснення простого керування, застосування циклів з умовними виразами, ітерації та *if*–твердження.

3.1.1. Умовні вирази

Мова програмування Python підтримує широкий набір операторів для встановлення взаємозв'язків між значеннями певних змінних. Повний набір цих операторів наведено у таблиці 3.1.

Таблиця 3.1

Operator	Relationship
<	less than
<=	less than or equal to
==	equal to (note this is two not one = sign)
!=	not equal to
>	greater than
>=	greater than or equal to

Можна використовувати ці оператори безпосередньо:

```
>>> 3 < 5
True
>>> 5 < 3
False
>>> not 5 < 3
True
>>>
```

Як результат цих дій ми отримали вирази із булевими значеннями *True, False*. Водночас *in, not* – це булеві оператори. Рядки і списки також підтримують умовні вирази.

```
>>> word = 'sovereignty'
>>> 'sovereign' in word
True
>>> 'gnt' in word
True
>>> 'pre' not in word
True
>>> 'Hello' in ['Hello', 'World']
True
>>> 'Hell' in ['Hello', 'World']
False
>>>
```

Можна перевірити наявність певних слів як в окремих реченнях, так і в усьому тексті:

```
>>> 'hello' in sent7
False
>>> 'hello' in text1
False
>>> 'he' in text1
True
>>>
```

Загальна схема роботи цих прикладів (*[w for w in text if condition]*), де умова *condition* або справджується, або ні (набуває значення *True* або *False*). Зазвичай ми використовуємо умовні оператори, як частину *if* операторів. Для перевірки властивостей окремих слів існує ряд відповідних функцій, поданих у таблиці 3.2.

Таблиця 3.2

Функція	Пояснення
s.startswith(t)	Чи починається *s* з *t*
s.endswith(t)	Чи закінчується *s* на *t*
t in s	Чи t міститься в *s*
s.islower()	Чи всі символи в *s* є малі
s.isupper()	Чи всі символи в *s* є великі
s.isalpha()	Чи всі символи в *s* є букви
s.isalnum()	Чи всі символи в *s* є букви і цифри
s.isdigit()	Чи всі символи в *s* є цифри
s.istitle()	Чи всі слова в *s* є з великої літери

Розглянемо приклади використання цих операторів для вибору окремих слів з тексту: слова з закінченням *–ableness*; слова, які містять *gnt*; слова, які починаються з великої літери; слова, які повність складаються з цифр.

```
>>> sorted([w for w in set(text1) if w.endswith('ableness')])
['comfortableness', 'honourableness', 'immutableness', 'indispensableness', ...]
>>> sorted([term for term in set(text4) if 'gnt' in term])
['Sovereignty', 'sovereignties', 'sovereignty']
>>> sorted([item for item in set(text6) if item.istitle()])
['A', 'Aaaaaaaaah', 'Aaaaaaaah', 'Aaaaaah', 'Aaaah', 'Aaaaugh', 'Aaagh', ...]
>>> sorted([item for item in set(sent7) if item.isdigit()])
['29', '61']
>>>
```

Можна ставити і складніші умови. Якщо *C* – це умова, то *not C* – це також умова. Якщо є дві умови *C1* та *C2*, то побудувати нову умову можна, використовуючи оператори диз'юнкції та кон'юнкції: c_1 *and* c_2, c_1 *or* c_2.

Виконати самостійно. Виконайте такі приклади і спробуйте пояснити, що відбувається в кожному з них:

```
>>> sorted([w for w in set(text7) if '-' in w and 'index' in w])
>>> sorted([wd for wd in set(text3) if wd.istitle() and len(wd) > 10])
>>> sorted([w for w in set(sent7) if not w.islower()])
>>> sorted([t for t in set(text2) if 'cie' in t or 'cei' in t])
```

Обробка кожного елемента. Розглянемо такі приклади:

```
>>> [len(w) for w in text1]
[1, 4, 4, 2, 6, 8, 4, 1, 9, 1, 1, 8, 2, 1, 4, 11, 5, 2, 1, 7, 6, 1, 3, 4, 5, 2, ...]
>>> [w.upper() for w in text1]
['[', 'MOBY', 'DICK', 'BY', 'HERMAN', 'MELVILLE', '1851', ']',
'ETYMOLOGY', '.', ...]
>>>
```

В цих прикладах бачимо такі вирази: [*f(w) for* ...] або [*w.f() for* ...], де *f* – це функція, яка або визначає довжину слова, або перетворює малі літери на великі. У кожному з цих прикладів здійснюється обробка кожного елемента списку. Змінній *w* послідовно присвоюються значення слів з текста і над цією змінною виконуються передбачені програмою дії. Такий запис [*f(w) for* ...] називається "list comprehension" (включення списків або спискові висловлювання) і є важливим для написання та розуміння програм на Python.

Застосуємо включення списків для підрахунку слів в тексті:

```
>>> len(text1)
260819
>>> len(set(text1))
19317
>>> len(set([word.lower() for word in text1]))
17231
>>>
```

В цьому прикладі ми уникнули подвійного підрахунку слів з великої літери (This and this), що зменшило загальну кількість приблизно на 2000 слів. Подібним способом можна уникнути підрахунку розділових знаків та чисел.

```
>>> len(set([word.lower() for word in text1 if word.isalpha()]))
16948
>>>
```

Цілі, рядки і списки є типами даних в Python. Кожне значення (змінна) має свій тип. Цей тип визначає, які операції ви можете виконувати зі змінною. Наприклад, ми можемо індексувати рядки і списки, але не можемо індексувати цілі.

```
>>> one = 'cat'
>>> one[0]
'c'
>>> two = [1, 2, 3]
>>> two[1]
2
>>> three = 3
>>> three[2]
Traceback (most recent call last):
  File "<pyshell#95>", line 1, in -toplevel-
    three[2]
TypeError: 'int' object is unsubscriptable
>>>
```

Можна використовувати функцію *type()* мови Python для перевірки типу об'єкта, з яким ми працюємо.

```
>>> data = [one, two, three]
>>> for item in data:
...     print "item '%s' belongs to %s" % (item, type(item))
...
item 'cat' belongs to <type 'str'>
item '[1, 2, 3]' belongs to <type 'list'>
item '3' belongs to <type 'int'>
>>>
```

3.1.2. Засоби здійснення простого керування

Більшість мов програмування дозволяють виконання окремих блоків програми, коли використовуються умовні вирази або *if* оператори. В наступній програмі ми створили змінну *word*, яка містить значення 'cat' типу рядок. *if*- оператор перевіряє умову – чи довжина слова <5, чи ні. Якщо умова виразу справджується, то виконується тіло *if* оператора і виконується оператор *print*.

```
>>> word = "cat"
>>> if len(word) < 5:
...     print 'word length is less than 5'
...
word length is less than 5
>>>
```

Якщо змінити умови виразу (довжина слова більша або дорівнює 5), то вираз не справджується і оператор *print* не виконується.

```
>>> if len(word) >= 5:
...     print 'word length is greater than or equal to 5'
...
>>>
```

if оператор – це структура керування, оскільки вона безпосередньо керує виконанням програми, зокрема визначає, яка частина коду в програмі буде виконуватися. Іншою типовою структурою керування є оператор циклу *for*. У наступному прикладі ми бачимо його застосування:

```
>>> for word in ['Call', 'me', 'Ishmael', '.']:
...     print word
...
Call
me
Ishmael
.
>>>
```

Як видно з наведених прикладів, рядок з операторами *If* та *for* має завершуватися двокрапкою.

3.1.3. Організація циклів з умовними виразами

Під час написання програм часто виникає необхідність поєднувати оператори *if* та *for*. В наступному прикладі в циклі обробляються всі елементи списка і друкуються тільки ті, які мають останню літеру *l*:

```
>>> sent1 = ['Call', 'me', 'Ishmael', '.']
>>> for xyzzy in sent1:
...     if xyzzy.endswith('l'):
...         print xyzzy
...
Call
Ishmael
>>>
```

Якщо потрібно, щоб щось відбувалося, коли умовний вираз не справджується, використовується оператор *else* в *if*–твердженні.

```
>>> if len(word) >= 5:
...    print 'word length is greater than or equal to 5'
... else:
...    print 'word length is less than 5'
...
word length is less than 5
>>>
```

Наступний приклад містить ще складнішу комбінацію з операторів *if, else, elif*.

```
>>> for token in sent1:
...     if token.islower():
...         print token, 'is a lowercase word'
...     elif token.istitle():
...         print token, 'is a titlecase word'
...     else:
...         print token, 'is punctuation'
...
Call is a titlecase word
me is a lowercase word
Ishmael is a titlecase word
. is punctuation
>>>
```

Якщо залишити частину умов *if* твердження пустою, то непустий рядок або список буде оброблятися як true, а пустий рядок чи список – як false.

```
>>> mixed = ['cat', '', ['dog'], []]
>>> for element in mixed:
...     if element:
...         print element
...
cat
['dog']
```

В цьому прикладі *if* твердження тотожне до *if len(element) > 0*.

Для встановлення відмінності у використанні *if...elif* роглянемо такий приклад:

```
>>> animals = ['cat', 'dog']
>>> if 'cat' in animals:
...     print 1
... elif 'dog' in animals:
...     print 2
...
1
>>>
```

У випадку, коли *if* вираз твердження задовольняється, *elif* вираз не виконується і програма ніколи не виведе на екран 2. *elif* вираз надає більше інформації ніж *if* вираз. Якщо *elif* вираз справджується, то це означає, що не тільки одна умова справджується, але й, водночас означає, що умова *if* виразу не справдилася.

3.1.4. Ітерації та *if*–твердження.

Нехай маємо рядок і необхідно написати програму, яка виведе на екран всі слова, що мають закінчення «ow». Програма буде складатися з таких частин. Спочатку розділимо рядок на список слів.

```
>>> sentence = 'how now brown cow'
>>> words = sentence.split()
>>> words
['how', 'now', 'brown', 'cow']
>>>
```

Тепер потрібно послідовно перебрати всі слова у списку. Напишемо такий цикл, за допомогою якого виводимо на екран всі слова .

```
>>> for word in words:
...     print word
...
how
now
brown
cow
```

Далі потрібно вивести на екран тільки необхідні нам слова із закінченням «ow». Попередньо можна перевірити, які саме слова необхідно вивести на екран.

```
>>> 'how'.endswith('ow')
True
>>> 'brown'.endswith('ow')
False
>>>
```

Тепер все готове для написання *if* твердження у *for* циклі.

```
>>> sentence = 'how now brown cow'
>>> words = sentence.split()
>>> for word in words:
...     if word.endswith('ow'):
...         print word
...
how
now
cow
>>>
```

На завершення розглянемо такий приклад:

```
>>> tricky = sorted([w for w in set(text2) if 'cie' in w or 'cei' in w])
>>> for word in tricky:
...     print word,
ancient ceiling conceit conceited conceive conscience
conscientious conscientiously deceitful deceive ...
>>>
```

Отже, спочатку було створено список, куди увійшли слова з буквосполученнями «cie» та «cei». Елементи цього списку оброблялися в циклі і роздруковувалися. Зауважимо, що кома в кінці оператора *print* вказує на те, щоб результати виводилися у рядок.

3.2. Клас FreqDist для простих статистичних досліджень

Для розв’язання задачі автоматичного визначення слів, які є найбільш інформативними для текстів певного жанру або певної тематики, спочатку інтуїтивно виникає думка побудувати частотний список або частотний розподіл. Частотний розподіл вказує на частоту, з якою в тексті зустрічається кожне зі слів. Такий частотний список називають розподілом тому, що він вказує, яким чином загальна кількість слів розподіляється між словниковими статтями (оригінальні слова) у тексті. Враховуючи, що побудова частотних розподілів досить часто необхідна під час обробки природної мови, у NLTK реалізовано окремий клас *FreqDist* у модулі *nltk.probability*. Наразі застосуємо цей клас для знаходження 50 найчастотніших слів в тексті Moby Dick.

```
>>> fdist1 = FreqDist(text1)                    #1
>>> fdist1                                      #2
<FreqDist with 260819 outcomes>
>>> vocabulary1 = fdist1.keys()                 #3
>>> vocabulary1[:50]                            #4
[',', 'the', '.', 'of', 'and', 'a', 'to', ';', 'in', 'that', '"', '-',
'his', 'it', 'I', 's', 'is', 'he', 'with', 'was', 'as', '"', 'all', 'for',
'this', '!', 'at', 'by', 'but', 'not', '--', 'him', 'from', 'be', 'on',
'so', 'whale', 'one', 'you', 'had', 'have', 'there', 'But', 'or', 'were',
'now', 'which', '?', 'me', 'like']
>>> fdist1['whale']
906
>>>
```

Під час першого виклику *FreqDist* назва тексту вказується як аргумент класу #1. Можна дізнатися загальну кількість слів, які були підраховані #2. Вираз *keys()* дозволяє встановити список оригінальних слів тексту #3, також можна переглянути перші 50 з них #4. Проте серед цих 50 слів тільки одне дає певну інформацію про зміст тексту (whale), причому це слово зустрічається у тексті 906 раз. Всі інші слова не є інформативними щодо змісту, зазвичай їх називають службовими. З метою визначення, яку частину

тексту займають такі службові слова, знайдемо їх сумарну частоту. Для цього побудуємо графік (рис. 3.1) за допомогою виразу *fdist1.plot(50, cumulative=True).*

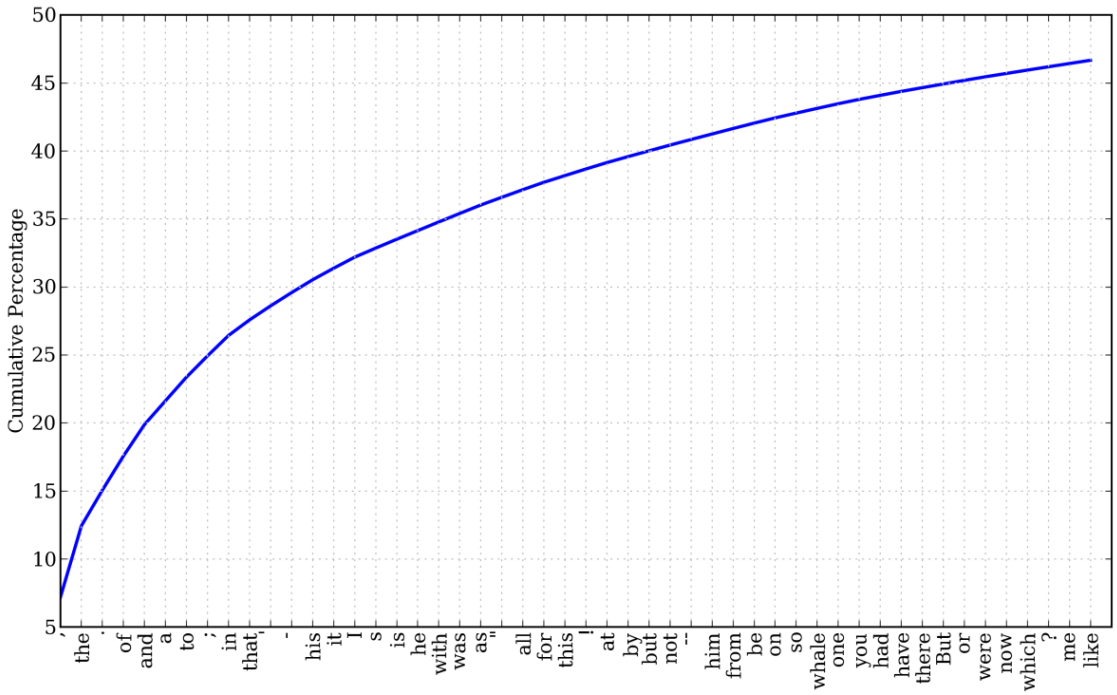


Рисунок 3.1 – Частотний розподіл (з накопиченням) 50 слів тексту Moby Dick

Якщо найбільш частотні слова не дають уяви про текст, то спробуємо за допомогою виразу *fdist1.hapaxes()* побудувати список слів, які зустрічаються тільки один раз. В цьому тексті маємо таких 9002 слова, зокрема серед них є такі слова як: exicographer, cetological, contraband, expostulation. Сенс деяких слів без контексту важко зрозуміти, отже вони також не дають уяву про текст.

3.2.1. Вибір слів з тексту

Наразі спробуємо знайти найдовші слова, сподіваючись що вони є більш інформативними:

```
>>> V = set(text1)
>>> long_words = [w for w in V if len(w) > 15]
>>> sorted(long_words)
['CIRCUMNAVIGATION', 'Physiognomically', 'apprehensiveness',
'cannibalistically', 'characteristically', 'circumnavigating', 'circumnavigation',
'circumnavigations', 'comprehensiveness', 'hermaphroditical', 'indiscriminately',
'indispensableness', 'irresistibleness', 'physiognomically', 'preternaturalness',
'responsibilities', 'simultaneousness', 'subterraneousness', 'supernaturalness',
'superstitiousness', 'uncomfortableness', 'uncompromisedness',
'undiscriminating', 'uninterpenetratingly']
>>>
```

Для кожного слова зі списку *V* здійснюється перевірка – чи його довжина є більша ніж 15 символів, всі інші слова ігноруються.

Виконати самостійно: виконати попередній приклад для інших текстів, змінюючи довжину слів. Результати порівняти і пояснити.

Повертаючись до задачі пошуку характерних (або ключових) для тексту слів зазначимо, що довгі слова в тексті № 4 constitutionally, transcontinental вказують на його офіційний характер, а довгі слова тексту №5, наприклад boooooooooooglyyyyyy and yuuuuuuuuuuuummmmmmmmmmm, вказують на його неформальний характер. Але довгі слова часто зустрічаються у тексті тільки один раз, а тому потрібно знайти ті довгі слова, які зустрічаються у тексті з певною частотою. Одночасно ми уникаємо врахування коротких неінформативних слів, а також уникаємо врахування довгих, але рідкісних слів.

В наступному прикладі показано, як із текста № 5 вибираються слова, які мають довжину більше ніж 7 символів і, водночас, зустрічаються у тесті більше, ніж 7 разів.

```
>>> fdist5 = FreqDist(text5)
>>> sorted([w for w in set(text5) if len(w) > 7 and fdist5[w] > 7])
['#14-19teens', '#talkcity_adults', '((((((((((', '.....', 'Question',
'actually', 'anything', 'computer', 'cute.-ass', 'everyone', 'football',
'innocent', 'listening', 'remember', 'seriously', 'something', 'together',
'tomorrow', 'watching']
>>>
```

3.2.2. Колокації та біграми

Колокація – це словосполучення, яке зустрічається дуже часто. Наприклад, red wine – це колокація, a the wine – ні. Характерною рисою колокацій є те, що вони дуже стійкі до заміни одного зі слів на інше, подібне за змістом (maroon wine). Для того, щоб побудувати колокації, спочатку потрібно побудувати на основі тексту пари слів, або біграми. Для цього можна використати функцію *bigrams()*:

```
>>> bigrams(['more', 'is', 'said', 'than', 'done'])
[('more', 'is'), ('is', 'said'), ('said', 'than'), ('than', 'done')]
>>>
```

Оскільки колокації являють собою частотні біграми з врахуванням випадків рідкісних слів, то потрібно знайти такі біграми, частота яких вища за частоту слів, з яких складається біграма. Функція *collocations()* реалізує такі дії і демонструє – колокації є характерними для текстів різних тематик і жанрів.

```
>>> text4.collocations()
Building collocations list
United States; fellow citizens; years ago; Federal Government; General
Government; American people; Vice President; Almighty God; Fellow
citizens; Chief Magistrate; Chief Justice; God bless; Indian tribes;
public debt; foreign nations; political parties; State governments;
```

```
National Government; United Nations; public money
>>> text8.collocations()
Building collocations list
medium build; social drinker; quiet nights; long term; age open;
financially secure; fun times; similar interests; Age open; poss
rship; single mum; permanent relationship; slim build; seeks lady;
Late 30s; Photo pls; Vibrant personality; European background; ASIAN
LADY; country drives
>>>
```

Крім підрахунку кількості окремих слів, цікаво також здійснити підрахунок довжин слів в тексті, використовуючи FreqDist.

```
>>> [len(w) for w in text1]                                    #1
[1, 4, 4, 2, 6, 8, 4, 1, 9, 1, 1, 8, 2, 1, 4, 11, 5, 2, 1, 7, 6, 1, 3, 4, 5, 2, ...]
>>> fdist = FreqDist([len(w) for w in text1])                  #2
>>> fdist                                                      #3
<FreqDist with 260819 outcomes>
>>> fdist.keys()
[3, 1, 4, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20]
>>>
```

Спочатку будуємо список довжин слів в тексті № 1 – #1. FreqDist здійснює підрахунок кількості вживань слів різної довжини – #2. Як результат в #3 отримуємо розподіл, в якому довжині слова відповідає кількість таких слів в тексті. Бачимо, що в тексті зустрічаються слова різної довжини від 1 до 20 символів. Частоту різних довжин слів можна переглянути:

```
>>> fdist.items()
[(3, 50223), (1, 47933), (4, 42345), (2, 38513), (5, 26597), (6, 17111), (7,
14399),
(8, 9966), (9, 6428), (10, 3528), (11, 1873), (12, 1053), (13, 567), (14, 177),
(15, 70), (16, 22), (17, 12), (18, 1), (20, 1)]
>>> fdist.max()
3
>>> fdist[3]
50223
>>> fdist.freq(3)
0.19255882431878046
>>>
```

Отже, у тексті № 1 найчастіше зустрічаються слова з довжиною 3 символи, причому такі слова становлять близько 20 % обсягу всього цього тексту. Можна здійснити аналіз довжин слів для текстів різних жанрів, авторів та мов.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями.
2. Виконати приклади, які використовуються в теоретичних відомостях.
3. Виконати такі вправи.

3.1. Створіть змінну sentence і присвойте їй значення ‘she sells sea shells by the sea shore’. Напишіть фрагмент програми для виведення на екран всіх слів, які починаються з ‘sh’.

3.2. Створіть змінну sentence і присвойте їй значення ‘she sells sea shells by the sea shore’. Напишіть фрагмент програми для виведення на екран всіх слів, довжина яких більша ніж 4 символи.

3.3. Створіть змінну sentence і присвойте їй значення ‘she sells sea shells by the sea shore’. Напишіть фрагмент програми, яка генерує новий рядок, додаючи ‘like’ перед кожним зі слів, яке починається з ‘se’.

3.4. Напишіть програму, яка видаляє всі голосні з рядка, що відповідає імені, по батькові та прізвищу студента. Програма має здійснювати таку послідовність дій: створення початкового рядка; створення рядка, у якому буде зберігатися результат; for цикл для обробки рядка за принципом «символ за символом» і запису неголосних символів в результуючий рядок.

3.5. Пустий рядок і пустий список в частині умов if виразу призводить до помилки. Напишіть програму для демонстрації таких випадків у випадку використання if–тверджень.

3.6. Перегляньте результати виконання умовних виразів: ’row’ in ’brown’ та ’row’ in [’ brown’, ’cow’]. Напишіть програму для перевірки наявності в рядку sent=’ ’colorless green ideas sleep furiously’ окремих слів та підрядків.

3.7. Виконайте ці приклади і поясніть, чому отримано різні результати (різні значення змінних):

```
sorted(set([w.lower() for w in text1]))
sorted([w.lower() for w in set(text1)])
```

3.8. Виконайте ці приклади і поясніть різницю між ними:

```
w.isupper()
not w.islower()
```

3.9. Знайдіть в тексті № 5 всі слова, довжина яких дорівнює 4 і побудуйте для них частотний розподіл.

3.10. Використовуючи оператори *if* та *for*, виведіть на екран всі слова з тексту № 6, які написані з великої літери.

3.11. Напишіть вираз для знаходження в тексті № 6 всіх слів, які відповідають таким вимогам: закінчуються на ize; містять літеру z; містять послідовність літер pt; написані з великої літери. Результат подайте як список слів.

3.12. Використайте вираз sum([len(w) for w in text1]) для знаходження середньої довжини слів в тексті.

3.13. Перевірте виконання виразу set(sent3) < set(text1). Змініть аргументи функції. Результати поясніть.

3.14. Побудуйте колокації для текстів № 1 та № 2. Результати порівняйте.

3.15. Побудуйте колокації для текстів № 1 та № 4. Результати порівняйте.

3.16. Побудуйте колокації для текстів № 1 та № 5. Результати порівняйте.

3.17. Побудуйте колокації для текстів № 1 та № 6. Результати порівняйте.

3.18. Побудуйте колокації для текстів № 1 та № 7. Результати порівняйте.

3.19. Побудуйте колокації для текстів № 1 та № 9. Результати порівняйте.

3.20. Побудуйте колокації для текстів № 2 та № 4. Результати порівняйте.

3.21. Побудуйте колокації для текстів № 2 та № 5. Результати порівняйте.

3.22. Побудуйте колокації для текстів № 2 та № 6. Результати порівняйте.

3.23. Побудуйте колокації для текстів № 2 та № 7. Результати порівняйте.

3.24. Побудуйте колокації для текстів № 2 та № 9. Результати порівняйте.

5. Підготувати і оформити звіт відповідно до варіанта індивідуального завдання за своїм порядковим номером у списку групи (таблиця 3.3).

Таблиця 3.3.

Варіант	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Номери завдань	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
	4	6	5	5	4	6	4	6	5	5	4	6	4	6	5
	8	7	8	8	7	7	8	8	7	7	8	8	7	7	8
	9	13	9	13	9	13	9	13	9	13	9	13	13	9	9
	11	12	11	12	11	12	11	12	11	12	11	12	11	12	12
	14	15	16	17	18	19	20	21	22	23	24	14	15	16	17
Варіант	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Номери завдань	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
	4	6	5	5	4	6	4	6	5	5	4	6	5	4	6
	7	8	8	7	7	8	8	7	7	8	8	7	7	8	7
	9	13	9	13	9	13	9	13	9	13	9	13	13	9	13
	11	12	11	12	11	12	11	12	11	12	11	12	11	12	11
	15	16	17	18	19	20	21	22	23	24	14	15	16	14	22

Контрольні запитання

1. Які ви знаєте оператори для встановлення взаємозв’язків між змінними у Python?
2. Як побудувати умову за допомогою операторів кон’юнкції та диз’юнкції?
3. Як називається запис [*f(w) for…*]?
4. Які типи даних підтримує мова Python?
5. Що являє собою *if* оператор?
6. Для чого використовується оператор *else* в *If*–твердженні?
7. Навіщо використовується *elif* вираз?
8. Що виконує клас *FreqDist* у модулі *nltk.probability*?
9. Як побудувати графік частот у класі *FreqDist*?
10. Що таке колокації?
11. Як побудувати колокації?
12. Що таке функція *bigrams()*?

Лабораторна робота № 4. Програмний доступ до корпусів текстів

Мета роботи: вивчення методів і засобів доступу до корпусів текстів та класу ConditionalFreqDist.

Теоретичні відомості

Вирішення задач обробки текстів природною мовою передбачає використання великих обсягів лінгвістичних даних, або, іншими словами, передбачає роботу з корпусами текстів. Виконання цієї лабораторної роботи допоможе знайти відповідь на такі запитання: які є відомі корпуси текстів та лексичні ресурси і як отримати до них доступ засобами Python; які корисні конструкції має Python для розв'язання таких задач.

4.1. Доступ до корпусів текстів

Корпус текстів – це складений за певними правилами досить великий набір текстів. Багато корпусів розроблено із збереженням балансу між текстами різних жанрів або авторів. Серед іншого – промови опрацьовувалися як один текст, не зважаючи на те, що кожна промова має окремого автора. Обробку було здійснено засобами мови Python. Під час роботи з копусами важливо мати засоби доступу як до всіх текстів, так і до окремих частин кожного з них, а також і до окремих слів.

4.1.1. Корпус Гутенберга

В NLTK входить невелика частина текстів з електронного архіву текстів Project Gutenberg, який містить 25000 безкоштовних електронних книжок різних авторів (http://www.gutenberg.org/). Тексти творів знаходяться в окремих файлах. Для одержання назв файлів (ідентифікаторів файлів), в яких зберігаються тексти корпусу, потрібно використати таку функцію:

```
>>> import nltk
>>> nltk.corpus.gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-
kjv.txt',
'blake-poems.txt', 'bryant-stories.txt', 'burgess-busterbrown.txt',
'carroll-alice.txt', 'chesterton-ball.txt', 'chesterton-brown.txt',
'chesterton-thursday.txt', 'edgeworth-parents.txt', 'melville-moby_dick.txt',
'milton-paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-hamlet.txt',
'shakespeare-macbeth.txt', 'whitman-leaves.txt']
```

Для роботи з першим текстом цього корпусу (роман Емма, автор Джейн Остін), створюємо змінну *emma*, через яку можемо, наприклад, знайти скільки слів має цей текст.

```
>>> emma = nltk.corpus.gutenberg.words('austen-emma.txt')
>>> len(emma)
```

192427

Під час створення змінної *emma* було використано функцію *words()* об'єкта *gutenberg* пакета *corpus* бібліотеки NLTK. Аналогічного результату можна досягнути, застосувавши більш компактний запис конструкцій Python.

```
>>> from nltk.corpus import gutenberg
>>> gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', ...]
>>> emma = gutenberg.words('austen-emma.txt')
```

Інший приклад ілюструє можливість програмної реалізації важливого у лінгвістиці поняття конкордансу – відображення всіх випадків вживання певного слова у тексті разом з відповідними контекстами. Для побудови конкордансу для слова *surprize* у файлі *austen-emma.txt* необхідно використати такі вирази:

```
>>> emma = nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt'))
>>> emma.concordance("surprize")
```

Спробуємо написати невелику програму для одержання інформації про кожен текст з корпусу. Побудуємо цикл за змінною *fileid*, яка відповідає ідентифікатору файла з текстом, і на кожному кроці будемо визначати деяку статистичну інформацію, яку для компактності запису будемо відображати цілими числами *int()*.

```
>>> for fileid in gutenberg.fileids():
  num_chars = len(gutenberg.raw(fileid))                    #1
  num_words = len(gutenberg.words(fileid))
  num_sents = len(gutenberg.sents(fileid))
  num_vocab = len(set([w.lower() for w in gutenberg.words(fileid)]))
  print int(num_chars/num_words), int(num_words/num_sents),
int(num_words/num_vocab), fileid
...
4 21 26 austen-emma.txt
4 23 16 austen-persuasion.txt
4 24 22 austen-sense.txt
4 33 79 bible-kjv.txt
4 18 5 blake-poems.txt
4 17 14 bryant-stories.txt
4 17 12 burgess-busterbrown.txt
4 16 12 carroll-alice.txt
4 17 11 chesterton-ball.txt
4 19 11 chesterton-brown.txt
4 16 10 chesterton-thursday.txt
4 18 24 edgeworth-parents.txt
4 24 15 melville-moby_dick.txt
4 52 10 milton-paradise.txt
4 12 8 shakespeare-caesar.txt
```

```
4 13 7 shakespeare-hamlet.txt
4 13 6 shakespeare-macbeth.txt
4 35 12 whitman-leaves.txt
```

Операція ділення в останньому операторі *print* видає цілочисленні результати (з округленням). Для одержання результатів без округлення потрібно використати конструкцію *from __future__ import division*:

```
>>> from __future__ import division
>>> for fileid in gutenberg.fileids():
  num_chars = len(gutenberg.raw(fileid))                    #1
  num_words = len(gutenberg.words(fileid))
  num_sents = len(gutenberg.sents(fileid))
  num_vocab = len(set([w.lower() for w in gutenberg.words(fileid)]))
  print int(num_chars/num_words), int(num_words/num_sents),
int(num_words/num_vocab), fileid
...
4.60990921232 21.6477669029 26.2019335512 austen-emma.txt
4.74979372727 23.7128019324 16.8245072836 austen-persuasion.txt
4.75378595242 24.1721017586 22.1108855224 austen-sense.txt
```

Ця програма відображає такі статистичні дані для кожного з текстів: середня довжина слова; середня довжина речення; значення лексичної різноманітності (відношення загальної кількості слів до кільності оригінальних слів). Чисельні значення (однакові для всіх текстів) вказують, що для англійської мови середнє значення довжини слова становить 4 символи (насправді 3, оскільки змінна *num_chars* містить і пропуски). На відміну від довжини слова, інші чисельні значення відрізняються і, певною мірою, є характерними для різних авторів.

У попередньому прикладі використовувалась функція *.raw()* для доступу до тексту книжки, без його поділу на окремі слова. Ця функція дозволяє доступитися до вмісту файла без будь-якої його попередньої лінгвістичної обробки. Тому використання *len(gutenberg.raw('blake-poems.txt')* дозволяє встановити скільки символів (разом з пропусками), що є в тексті. Функція *sents()* ділить текст на окремі речення, причому кожне речення подається як список рядків, де рядки – елементи списку містять окремі слова речення.

```
>>> macbeth_sentences = gutenberg.sents('shakespeare-macbeth.txt')
>>> macbeth_sentences
[['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare',
'1603', ']'], ['Actus', 'Primus', '.'], ...]
>>> macbeth_sentences[1037]
['Double', ',', 'double', ',', 'toile', 'and', 'trouble', ';',
'Fire', 'burne', ',', 'and', 'Cauldron', 'bubble']
>>> longest_len = max([len(s) for s in macbeth_sentences])
>>> [s for s in macbeth_sentences if len(s) == longest_len]
```

```
[['Doubtfull', 'it', 'stood', ',', 'As', 'two', 'spent', 'Swimmers', ',', 'that',
'doe', 'cling', 'together', ',', 'And', 'choake', 'their', 'Art', ':', 'The',
'mercilesse', 'Macdonwald', ...], ...]
```

4.1.2. Тексти з Інтернету

Project Gutenberg містить тисячі книжок, тому є представником літературної англійської мови. Для роботи з менш формальною мовою NLTK містить набір текстів з Інтернету: тексти з форумів, тексти з фільму «Пірати Карибського моря», тексти особистих оголошень, телефонні розмови, огляд вин тощо:

```
>>> from nltk.corpus import webtext
>>> for fileid in webtext.fileids():
...     print fileid, webtext.raw(fileid)[:65], '...'
...
firefox.txt Cookie Manager: "Don't allow sites that set removed cookies to
se...
grail.txt SCENE 1: [wind] [clop clop clop] KING ARTHUR: Whoa there!
[clop...
overheard.txt White guy: So, do you have any plans for this evening?
Asian girl...
pirates.txt PIRATES OF THE CARRIBEAN: DEAD MAN'S CHEST, by
Ted Elliott & Terr...
singles.txt 25 SEXY MALE, seeks attrac older single lady, for discreet
encoun...
wine.txt Lovely delicate, fragrant Rhone wine. Polished leather and
strawb...
```

Також в NLTK входить корпус повідомлень з чатів, створений в Naval Postgraduate School для досліджень з метою автоматичного виявлення Інтернет-злочинців. Цей корпус містить 10000 анонімних повідомлень в яких імена користувачів замінені за шаблоном "UserNNN", а також видалена інша персональна інформація, Корпус організовано як 15 окремих файлів, кожен з яких містить декілька сотень повідомлень з певною датою створення та вікових даних авторів (підлітки, 20-ти, 30-ти та 40-річні, дорослі). Назва файла містить інформацію про дату, вікову групу та кількість повідомлень, наприклад файл 10-19-20s_706posts.xml містить 706 повідомлень двадцятирічних дописувачів від 19 жовтня 2006 року.

```
>>> from nltk.corpus import nps_chat
>>> chatroom = nps_chat.posts('10-19-20s_706posts.xml')
>>> chatroom[123]
['i', 'do', "n't", 'want', 'hot', 'pics', 'of', 'a', 'female', ',', 'I', 'can', 'look', 'in', 'a', 'mirror', '.']
>>>
```

4.1.3. Корпус Brown

Корпус Brown – це перший корпус англійської мови обсягом один мільйон слів, який було створено в 1961–1964 роках в університеті Brown. Цей корпус містить тексти з 500 різних джерел, які відповідають різним жанрам. В табл. 4.1 наведено приклади для кожного з жанрів.

Використовуючи засоби NLTK можна отримати доступ до цього корпусу як до списку слів або списку речень (кожне речення, власне – список слів). Також доступна можливість вибору текстів з окремої категорії або з окремого файлу.

Таблиця 4.1 – Приклади текстів для кожного з жанрів корпуса Brown

ID	**Файл**	**Жанр**	**Опис тексту**
A16	ca16	news	Chicago Tribune: *Society Reportage*
B02	cb02	editorial	Christian Science Monitor: *Editorials*
C17	cc17	reviews	Time Magazine: *Reviews*
D12	cd12	religion	Underwood: *Probing the Ethics of Realtors*
E36	ce36	hobbies	Norling: *Renting a Car in Europe*
F25	cf25	lore	Boroff: *Jewish Teenage Culture*
G22	cg22	belles_lettres	Reiner: *Coping with Runaway Technology*
H15	ch15	government	US Office of Civil and Defence Mobilization: *The Family Fallout Shelter*
J17	cj19	learned	Mosteller: *Probability with Statistical Applications*
K04	ck04	fiction	W.E.B. Du Bois: *Worlds of Color*
L13	cl13	mystery	Hitchens: *Footsteps in the Night*
M01	cm01	science_fiction	Heinlein: *Stranger in a Strange Land*
N14	cn15	adventure	Field: *Rattlesnake Ridge*
P12	cp12	romance	Callaghan: *A Passion in Rome*
R06	cr06	humor	Thurber: *The Future, If Any, of Comedy*

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies',
'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'reviews', 'romance',
'science_fiction']
>>> brown.words(categories='news')
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
>>> brown.words(fileids=['cg22'])
['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]
>>> brown.sents(categories=['news', 'editorial', 'reviews'])
[['The', 'Fulton', 'County'...], ['The', 'jury', 'further'...], ...]
```

Корпус Brown – зручний та відомий ресурс для систематичного вивчення відмінностей між жанрами, або, іншими словами, для дослідження стилістики текстів. Спробуємо порівняти жанри і встановити, яким чином у текстах різних жанрів використовуються модальні дієслова. Для цього можна зробити підрахунки вживання різних модальних дієслів для різних жанрів.

```
>>> from nltk.corpus import brown
>>> news_text = brown.words(categories='news')
>>> fdist = nltk.FreqDist([w.lower() for w in news_text])
>>> modals = ['can', 'could', 'may', 'might', 'must', 'will']
>>> for m in modals:
...     print m + ':', fdist[m],
...
can: 94 could: 87 may: 93 might: 38 must: 53 will: 389
```

Виконати самостійно. Здійснити розрахунки для інших жанрів та для інших слів. Наприклад для *wh*-слів, тобто таких, як what, when, where, who, та why.

4.1.4. Корпус інформаційного агентства Рейтер

Корпус Reuters містить 10788 текстів новин загальним обсягом 1,3 мільйона слів. Всі тексти поділено на категорії за 90 темами і поділено на два набори (тренування та тестування). Такий поділ необхідний для тренування та тестування алгоритмів автоматичного визначення тематики тексту.

```
>>> from nltk.corpus import reuters
>>> reuters.fileids()
['test/14826', 'test/14828', 'test/14829', 'test/14832', ...]
>>> reuters.categories()
['acq', 'alum', 'barley', 'bop', 'carcass', 'castor-oil', 'cocoa',
'coconut', 'coconut-oil', 'coffee', 'copper', 'copra-cake', 'corn',
'cotton', 'cotton-oil', 'cpi', 'cpu', 'crude', 'dfl', 'dlr', ...]
```

На відміну від корпусу Brown, категорії текстів в цьому корпусі можуть накладатися одна на одну, оскільки тематика новин (газетних публікацій) переважно торкається багатьох тем. Засобами NLTK можна звернутися до тем, які відображаються в одному або декількох текстах або, навпаки, дізнатися весь перелік текстів, що належать до певної категорії.

```
>>> reuters.categories('training/9865')
['barley', 'corn', 'grain', 'wheat']
>>> reuters.categories(['training/9865', 'training/9880'])
['barley', 'corn', 'grain', 'money-fx', 'wheat']
>>> reuters.fileids('barley')
['test/15618', 'test/15649', 'test/15676', 'test/15728', 'test/15871', ...]
>>> reuters.fileids(['barley', 'corn'])
['test/14832', 'test/14858', 'test/15033', 'test/15043', 'test/15106',
```

'test/15287', 'test/15341', 'test/15618', 'test/15618', 'test/15648', ...]

4.1.5. Корпус інаугураційних промов президентів США

Знайомлячись з бібліотекою програм NLTK ми вже працювали з цим корпусом і розглядали весь корпус як один текст, що давало можливість знайти місце окремого слова в текстах промов, починаючи від першого слова першої промови. Насправді корпус – це набір 55 текстів, кожний з яких є промовою одного президента. Цікавою особливістю цього корпусу є можливість дослідити розподіл текстів за часовими проміжками. Назва кожного тексту містить рік проголошення промови і, відповідно, є можливість доступитися до цієї інформації, звернувшись до перших чотирьох символів назви файлу [fileid[:4].

```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()
['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', ...]
>>> [fileid[:4] for fileid in inaugural.fileids()]
['1789', '1793', '1797', '1801', '1805', '1809', '1813', '1817', '1821', ...]
```

4.1.6. Анотовані (розмічені) корпуси текстів

Більшість корпусів текстів є лінгвістично анотованими, тобто містять різного типу розмітку – морфологічну, синтаксичну, семантичну, в них можуть бути виділені власні назви, вказані семантичні ролі тощо. NLTK забезпечує способи доступу до багатьох корпусів текстів і розповсюджується з цими корпусами або їх фрагментами. В таблиці 4.2 наведено перелік доступних для NLTK корпусів текстів та їх короткий опис.

Таблиця 4.2 – Перелік корпусів текстів, які розповсюджуються разом з NLTK

Corpus	Compiler	Contents
Brown Corpus	Francis, Kucera	15 genres, 1.15M words, tagged, categorized
CESS Treebanks	CLiC-UB	1M words, tagged and parsed (Catalan, Spanish)
Chat-80 Data Files	Pereira & Warren	World Geographic Database
CMU Pronouncing Dictionary	CMU	127k entries
CoNLL 2000 Chunking Data	CoNLL	270k words, tagged and chunked
CoNLL 2002 Named Entity	CoNLL	700k words, pos- and named-entity-tagged (Dutch, Spanish)
CoNLL 2007 Dependency Treebanks (sel)	CoNLL	150k words, dependency parsed (Basque, Catalan)
Dependency Treebank	Narad	Dependency parsed version of Penn Treebank sample
Floresta Treebank	Diana Santos et al	9k sentences, tagged and parsed (Portuguese)
Gazetteer Lists	Various	Lists of cities and countries
Genesis Corpus	Misc web sources	6 texts, 200k words, 6 languages

Corpus	Compiler	Contents
Gutenberg (selections)	Hart, Newby, et al	18 texts, 2M words
Inaugural Address Corpus	CSpan	US Presidential Inaugural Addresses (1789-present)
Indian POS-Tagged Corpus	Kumaran et al	60k words, tagged (Bangla, Hindi, Marathi, Telugu)
MacMorpho Corpus	NILC, USP, Brazil	1M words, tagged (Brazilian Portuguese)
Movie Reviews	Pang, Lee	2k movie reviews with sentiment polarity classification
Names Corpus	Kantrowitz, Ross	8k male and female names
NIST 1999 Info Extr (selections)	Garofolo	63k words, newswire and named-entity SGML markup
NPS Chat Corpus	Forsyth, Martell	10k IM chat posts, POS-tagged and dialogue-act tagged
PP Attachment Corpus	Ratnaparkhi	28k prepositional phrases, tagged as noun or verb modifiers
Proposition Bank	Palmer	113k propositions, 3300 verb frames
Question Classification	Li, Roth	6k questions, categorized
Reuters Corpus	Reuters	1.3M words, 10k news documents, categorized
Roget's Thesaurus	Project Gutenberg	200k words, formatted text
RTE Textual Entailment	Dagan et al	8k sentence pairs, categorized
SEMCOR	Rus, Mihalcea	880k words, part-of-speech and sense tagged
Senseval 2 Corpus	Pedersen	600k words, part-of-speech and sense tagged
Shakespeare texts (selections)	Bosak	8 books in XML format
State of the Union Corpus	CSPAN	485k words, formatted text
Stopwords Corpus	Porter et al	2,400 stopwords for 11 languages
Swadesh Corpus	Wiktionary	comparative wordlists in 24 languages
Switchboard Corpus (selections)	LDC	36 phonecalls, transcribed, parsed
Univ Decl of Human Rights	United Nations	480k words, 300+ languages
Penn Treebank (selections)	LDC	40k words, tagged and parsed
TIMIT Corpus (selections)	NIST/LDC	audio files and transcripts for 16 speakers
VerbNet 2.1	Palmer et al	5k verbs, hierarchically organized, linked to WordNet
Wordlist Corpus	OpenOffice.org et al	960k words and 20k affixes for 8 languages
WordNet 3.0 (English)	Miller, Fellbaum	145k synonym sets

4.1.7. Корпуси іншомовних текстів

NLTK містить засоби роботи з іншомовними корпусами текстів (не англійськими). Для роботи з цими корпусами потрібно попередньо ознайомитися з питаннями кодування символів в Python (див. Лабораторну роботу № 2).

```
>>> nltk.corpus.cess_esp.words()
['El', 'grupo', 'estatal', 'Electricit\xe9_de_France', ...]
>>> nltk.corpus.floresta.words()
['Um', 'revivalismo', 'refrescante', 'O', '7_e_Meio', ...]
>>> nltk.corpus.indian.words('hindi.pos')
['\xe0\xa4\xaa\xe0\xa5\x82\xe0\xa4\xb0\xe0\xa5\x8d\xe0\xa4\xa3',
'\xe0\xa4\xaa\xe0\xa5\x8d\xe0\xa4\xb0\xe0\xa4\xa4\xe0\xa4\xbf\xe0\xa4\
xac\xe0\xa4
\x82\xe0\xa4\xa7', ...]
>>> nltk.corpus.udhr.fileids()
['Abkhaz-Cyrillic+Abkh', 'Abkhaz-UTF8', 'Achehnese-Latin1', 'Achuar-
Shiwiar-Latin1',
'Adja-UTF8', 'Afaan_Oromo_Oromiffa-Latin1', 'Afrikaans-Latin1',
'Aguaruna-Latin1',
'Akuapem_Twi-UTF8', 'Albanian_Shqip-Latin1', 'Amahuaca', 'Amahuaca-
Latin1', ...]
>>> nltk.corpus.udhr.words('Javanese-Latin1')[11:]
[u'Saben', u'umat', u'manungsa', u'lair', u'kanthi', ...]
```

Останній з розглянутих в попередньому прикладів корпусів (udhr) – це набір текстів різними мовами (300 мов) Декларації прав людини.

4.1.8. Структура корпусів текстів

Розглянувши приклади корпусів текстів можна зробити висновок, що всі вони мають різну структуру (рис. 4.1). Найпростіший корпус текстів не має структури, це набір текстів. Інші корпуси – це набори текстів, поділених за категоріями мови, жанру, автора. У багатьох випадках категорії текстів можуть перетинатися між собою, оскільки тексти можуть належати різним категоріям. Окремий випадок, коли набори текстів розподілені за часовими параметрами.

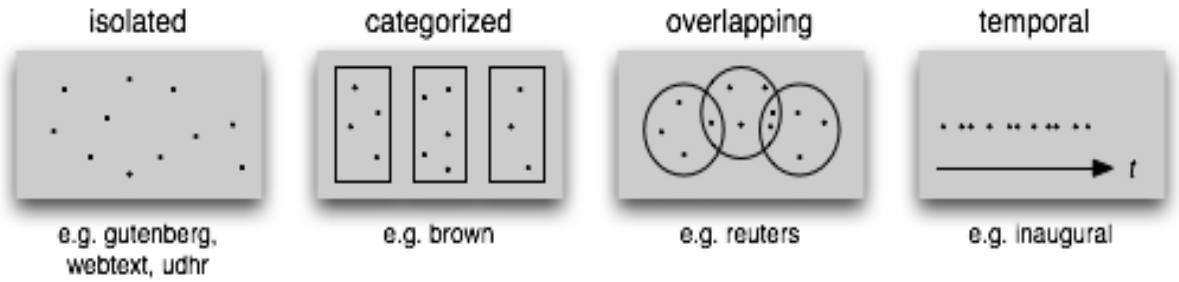


Рисунок 4.1 – Загальні структури корпусів текстів

Засоби NLTK забезпечують ефективні способи доступу до різних корпусів та роботи з існуючими та новими корпусами. Таблиця 4.3 містить набір функцій, які підтримує NLTK для роботи з корпусами.

Таблиця 4.3 – Основні функції NLTK для роботи з корпусами

Приклад використання функції	**Опис**
fileids()	Файли корпуса
fileids([categories])	Файли корпуса, що відповідають цій категорії
categories()	Категорії корпуса
categories([fileids])	Категорії корпуса, що відповідають цим файлам
raw()	Корпус, як послідовність символів
raw(fileids=[f1,f2,f3])	Послідовність символів з наступних файлів
raw(categories=[c1,c2])	Послідовність символів з наступних категорій
words()	Слова корпуса
words(fileids=[f1,f2,f3])	Слова з наступних файлів
words(categories=[c1,c2])	Слова з наступних категорій
sents()	Речення корпуса
sents(fileids=[f1,f2,f3])	Речення корпуса з наступних файлів
sents(categories=[c1,c2])	Речення корпуса з наступних категорій
abspath(fileid)	Місцезнаходження даного файлу на диску
encoding(fileid)	Кодування файлу (якщо відоме)
open(fileid)	Відкриття файла з корпуса для читання
root()	Шлях до місця де встановлено корпус
readme()	Вміст файла README корпуса текстів

Відмінності між методами доступу до корпусів можна проілюструвати наступним прикладом.

```
>>> raw = gutenberg.raw("burgess-busterbrown.txt")
>>> raw[1:20]
'The Adventures of B'
>>> words = gutenberg.words("burgess-busterbrown.txt")
>>> words[1:20]
['The', 'Adventures', 'of', 'Buster', 'Bear', 'by', 'Thornton', 'W', '.',
```

```
'Burgess', '1920', ']', 'I', 'BUSTER', 'BEAR', 'GOES', 'FISHING', 'Buster',
'Bear']
>>> sents = gutenberg.sents("burgess-busterbrown.txt")
>>> sents[1:20]
[['I'], ['BUSTER', 'BEAR', 'GOES', 'FISHING'], ['Buster', 'Bear', 'yawned',
'as',
'he', 'lay', 'on', 'his', 'comfortable', 'bed', 'of', 'leaves', 'and', 'watched',
'the', 'first', 'early', 'morning', 'sunbeams', 'creeping', 'through', ...], ...]
```

4.1.9. Доступ до власних корпусів текстів

За наявності власного набору текстових файлів, до них також можна організувати доступ як до корпусу текстів. Для цього використовують перелічені вище методи, попередньо застосувавши клас NLTK *PlaintextCorpusReader*. Потрібно знати розміщення файлів на диску, наприклад, якщо маємо шлях *C:\NLTK\Texts\Corpus_1*, то змінній *corpus_root* присвоюється це значення (#1). Клас *PlaintextCorpusReader* має два параметри – шлях до файлів та шаблон вибору файлів (#2) і повертає список назв файлів.

```
>>> import nltk
>>> from nltk.corpus import PlaintextCorpusReader
>>> corpus_root =
'C:\NLTK\Texts\Corpus_1'                                    #1
>>> wordlists = PlaintextCorpusReader(corpus_root,
'.*')                    #2
>>> wordlists.fileids()
['004556.html', '036_link.doc', '048.htm', '2.doc', '2.rtf', '240-0679.rar',
'36.pdf', '41.htm', '48.htm', '57224_1.rtf', '7.doc', 'about_pc-kimmo.html',
'ai00011f.htm', 'archive_article.asp.htm']
>>> wordlists.words('about_pc-kimmo.html')
['<!', 'DOCTYPE', 'HTML', 'PUBLIC', '"-//', 'W3C', ...]
```

У наступному прикладі показано, яким чином можна доступитися до локальної копії корпуса *PennTreebank*, використовуючи клас *BracketParseCorpusReader*.

```
>>> from nltk.corpus import BracketParseCorpusReader
>>> corpus_root = r"C:\corpora\penntreebank\parsed\mrg\wsj"
>>> file_pattern = r".*/wsj_.*\.mrg"
>>> ptb = BracketParseCorpusReader(corpus_root, file_pattern)
>>> ptb.fileids()
['00/wsj_0001.mrg', '00/wsj_0002.mrg', '00/wsj_0003.mrg',
'00/wsj_0004.mrg', ...]
>>> len(ptb.sents())
49208
>>> ptb.sents(fileids='20/wsj_2013.mrg')[19]
['The', '55-year-old', 'Mr.', 'Noriega', 'is', "n't", 'as', 'smooth', 'as', 'the',
```

```
'shah', 'of', 'Iran', ',', 'as', 'well-born', 'as', 'Nicaragua', "'s", 'Anastasio',
'Somoza', ',', 'as', 'imperial', 'as', 'Ferdinand', 'Marcos', 'of', 'the',
'Philippines',
'or', 'as', 'bloody', 'as', 'Haiti', "'s", 'Baby', Doc', 'Duvalier', '.']
```

4.2. Умовний частотний розподіл. Клас ConditionalFreqDist

Якщо тексти у корпусі поділено на різні категорії (за жанром, тематикою, авторами), то можна побудувати частотні розподіли для кожної з категорій. Такі дані дозволяють досліджувати відмінності між жанрами. Умовний частотний розподіл – це набір частотних розподілів, кожний з яких відповідає певній «умові». Однією з таких умов може бути категорія тексту.

4.2.1. Умови і події

Частотний розподіл визначає чисельні значення для кожної події, якими можемо вважати вживання слів в тексті. Умовний частотний розподіл поєднує в пари кожну подію та умову. Замість обробки послідовності слів (#1) обробляються послідовності пар (#2).

```
>>> text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
            #1
>>> pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...]
  #2
```

Кожна пара відповідає шаблону *(condition, event)*. Якщо розглядати корпус Brown за жанрами, то отримаємо 15 умов (одна для жанру) і 1161192 подій (одна для слова).

4.2.2. Підрахунок слів для окремих жанрів

Використовуючи клас *ConditionalFreqDist* можна визначити частоту слів для різних жанрів. У випадку модальних дієслів програма буде виглядати таким чином.

```
>>> cfd = nltk.ConditionalFreqDist(
...         (genre, word)
...         for genre in brown.categories()
...         for word in brown.words(categories=genre))
>>> genres = ['news', 'religion', 'hobbies', 'science_fiction', 'romance',
'humor']
>>> modals = ['can', 'could', 'may', 'might', 'must', 'will']
>>> cfd.tabulate(conditions=genres, samples=modals)
                 can could  may might must will
           news   93   86   66   38   50  389
       religion   82   59   78   12   54   71
        hobbies  268   58  131   22   83  264
science_fiction   16   49    4   12    8   16
        romance   74  193   11   51   45   43
          humor   16   30    8    8    9   13
```

Якщо для класу *FreqDist()* вхідними даними є список, то для класу *ConditionalFreqDist()* вхідними даними є список пар.

Розглянемо окремо тільки два жанри – новини і романтика. Для кожного жанру (#2) в циклі обробляємо кожне слово цього жанру (#3) і отримуємо пари, які містять жанр і слово (#1).

```
>>> genre_word = [(genre, word)                                  #1
...           for genre in ['news', 'romance']                   #2
...           for word in brown.words(categories=genre)]         #3
>>> len(genre_word)
170576
```

Пари на початку списку *genre_word* будуть мати форму *('news', word)*, тоді як з кінця списку їх форма буде такою *('romance', word)*.

```
>>> genre_word[:4]
[('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ('news', 'Grand')] #
[_start-genre]
>>> genre_word[-4:]
[('romance', 'afraid'), ('romance', 'not'), ('romance', """), ('romance', '.')] #
[_end-genre]
```

Можна використати цей список пар для побудови умовного частотного розподілу. Результати побудови збережемо в окремій змінній *cfd*. Перевіривши значення змінної (#1), дізнаємося про кількість умов, а також можемо переглянути ці умови (#2) та пересвідчитись, що для кожної з умов побудовано частотний розподіл (#3).

```
>>> cfd = nltk.ConditionalFreqDist(genre_word)
>>> cfd                                                          #1
<ConditionalFreqDist with 2 conditions>
>>> cfd.conditions()                                             #2
['news', 'romance'] # [_conditions-cfd]
>>> cfd['news']                                                  #3
<FreqDist with 100554 outcomes>
>>> cfd['romance']
<FreqDist with 70022 outcomes>
>>> list(cfd['romance'])
[',', '.', 'the', 'and', 'to', 'a', 'of', '``', """, 'was', 'I', 'in', 'he', 'had', '?', 'her', 'that',
'it', 'his', 'she', 'with', 'you', 'for', 'at', 'He', 'on', 'him', 'said', '!', '--', 'be', 'as', ';',
'have', 'but', 'not', 'would', 'She', 'The', ...]
>>> cfd['romance']['could']
193
```

За допомогою умовного частотного розподілу можна дослідити вживання слів в часовому проміжку. Наприклад, досліджуємо слова *America* та *citizen*. Спочатку перетворюємо всі слова корпусу промов президентів США до одного вигляду (#1), потім перевіряємо початкові літери слів для врахування різних форм *American's* and *Citizens*. Далі будуємо умовний частотний розподіл, а результати подаємо в графічному вигляді (рис. 4.2).

```
>>> cfd = nltk.ConditionalFreqDist(
...           (target, fileid[:4])
...           for fileid in inaugural.fileids()
...           for w in inaugural.words(fileid)
...           for target in ['america', 'citizen']
...           if w.lower().startswith(target))              #1
>>> cfd.plot()
```

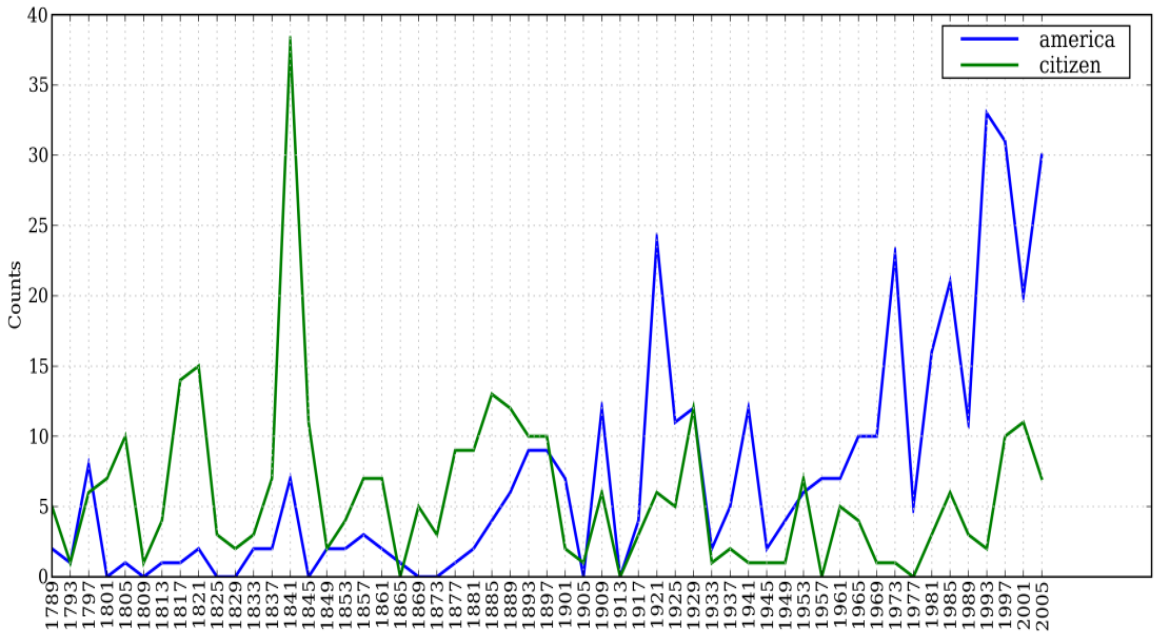


Рисунок 4.2 – Умовний частотний розподіл для визначення частоти вживання слів у різні частові проміжки

Подібний графік (рис. 4.3) можна побудувати для порівняння довжин слів у різних мовах. Для цього також використовуємо умовний частотний розподіл, але аналізуємо корпус *udhr*.

```
>>> from nltk.corpus import udhr
>>> languages = ['Chickasaw', 'English', 'German_Deutsch',
...    'Greenlandic_Inuktikut', 'Hungarian_Magyar', 'Ibibio_Efik']
>>> cfd = nltk.ConditionalFreqDist(
...           (lang, len(word))
...           for lang in languages
...           for word in udhr.words(lang + '-Latin1'))
>>> cfd.plot(cumulative=True)
```

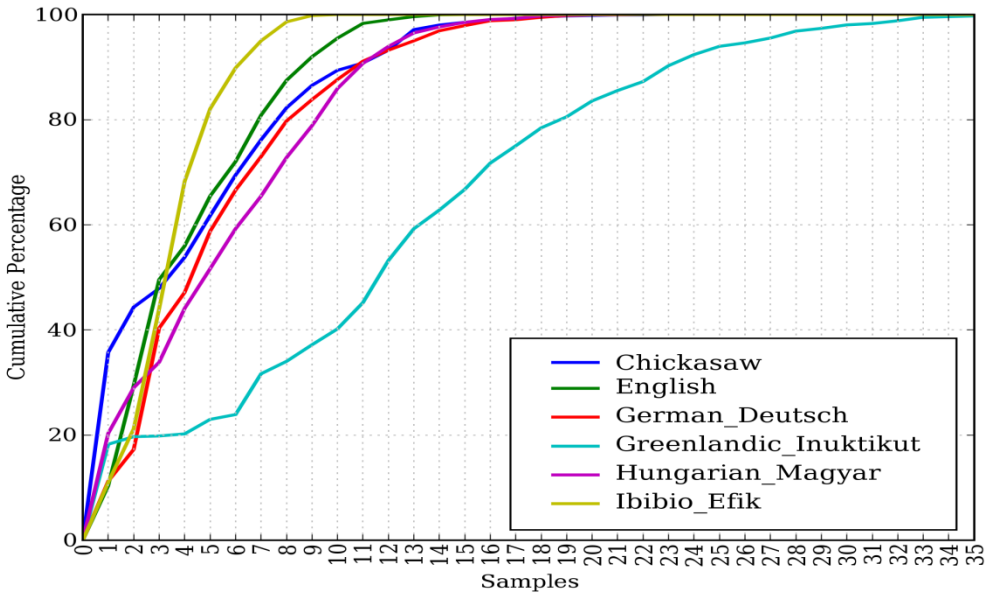


Рисунок 4.3 – Умовний частотний розподіл довжин слів для різних мов

Метод *plot()* а також метод *tabulate()* дозволяють визначати, які з умов будуть відображатися на екрані за допомогою параметра *conditions = parameter*. Так само визначають кількість прикладів для відображення за допомогою параметра *samples = parameter*. Для прикладу, в наступній табличній формі відображаються довжини слів до 10 символів для двох мов.

```
>>> cfd.tabulate(conditions=['English', 'German_Deutsch'],
...              samples=range(10), cumulative=True)
                   0    1    2    3    4    5    6    7    8    9
       English     0  185  525  883  997 1166 1283 1440 1558 1638
German_Deutsch     0  171  263  614  717  894 1013 1110 1213 1275
```

Виконати самостійно. Здійснити аналіз двох жанрів корпусу Brown (новини, романтика) для визначення, які з днів тижня є більш романтичні, а які містять більше новин. Для вирішення цієї задачі потрібно побудувати умовний частотний розподіл, де умовами є жанри, а подіями – дні тижня. Результати подати в табличній і графічній формах.

4.3. Використання текстового редактора в процесі створення програм

Інтерпретатор виконує оператори програми відразу після їх введення. Часто виникає потреба спочатку написати програму, яка складається з багатьох рядків (застосувавши текстовий редактор), а потім запустити її на виконання. Використовуючи IDLE можна це зробити, відкривши нове вікно за допомогою пункту меню File, зберігши текст програми у файлі **.py* і запустивши програму на виконання за допомогою команди Run Module пункту меню Run.

4.3.1. Поняття функції та модуля

Під час програмування доволі часто частину програми необхідно виконати (використати) декілька разів. Наприклад, потрібно написати програму, яка здійснює утворення множини з однини іменників, і так має бути виконано в різних місцях програми. Швидше, ніж повторювати той самий код декілька разів, більш ефективно і надійно організувати цю роботу через функцію. Функція – це програмна конструкція, яку можна викликати з одним або більше вхідними параметрами, і отримувати результат на виході. Визначаємо функцію, використовуючи ключове слово *def*, далі потрібно дати назву функції і визначити вхідні параметри, після двокрапки записується тіло функції. Ключове слово *return* використовується для відображення значення, яке ми хочемо отримати на виході функції. Розглянемо приклад – функція *plural()* отримує на вході однину іменника і формує множину на виході.

```
def plural(word):
    if word.endswith('y'):
        return word[:-1] + 'ies'
    elif word[-1] in 'sx' or word[-2:] in ['sh', 'ch']:
        return word + 'es'
    elif word.endswith('an'):
        return word[:-2] + 'en'
    else:
        return word + 's'
```

```
>>> plural('fairy')
'fairies'
>>> plural('woman')
'women'
```

4.3.2. Додаткові приклади

Розробляючи ту чи іншу програму протягом довшого періоду, додаючи до неї нові функції і змінюючи існуючі (або розробляючи декілька версій однієї програми), потрібно зберігати тексти програм в окремих файлах і організувати доступ до відповідних функцій у цих програмах. Збережемо текст останньої функції *plural()* в окремому файлі *textproc.py*. Тепер завжди можна доступитися до цієї функції, імпортувавши її з файлу.

```
>>> from textproc import plural
>>> plural('wish')
wishes
>>> plural('fan')
fen
```

Множина змінних і функцій, збережених у файлі, має в Python назву модуля. Множину змістовно пов’язаних між собою модулів називають пакетом. Програма обробки корпусу Brown є прикладом модуля, а множина

програм для роботи зі всіма корпусами – це приклад пакета. Тоді, власне, NLTK – це множина пакетів, яку називають бібліотекою.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями.
2. Виконати приклади, які використовуються в теоретичних відомостях.
3. Виконати такі вправи.

3.1. Використовуючи модуль *corpus* прочитайте текст *austin-persuasion.txt*. Визначте, скільки *tokens* (слів) і *type* (унікальних слів) містить ця книжка.

3.2. Напишіть, використовуючи модуль читання корпусу текстів *Brown nltk.corpus.brown.words()* програму, яка дозволяє доступитися до фрагментів текстів у двох різних жанрах корпусу *Brown*, і назва яких відповідає першій літері прізвища та імені студента.

3.3. Прочитайте тексти з корпусу *State of the Union addresses*, використовуючи *state_union* модуль читання. Визначте частоту вживання слів *men, women, people* в кожному з документів. Як змінилася частота вживання цих слів з часом?

3.4. Використовуючи конкорданси поясніть відмінності у вживанні слова however на початку речення ("in whatever way", "to whatever extent", або "nevertheless”).

3.5. Виберіть пару текстів і дослідіть відмінності між ними (кількість оригінальних слів, багатство мови, жанр). Знайдіть слова, які мають різний зміст в цих текстах, подібно до слова monstrous в *Moby Dick* та у *Sense and Sensibility.*

3.6. Проаналізуйте таблицю частот модальних дієслів для різних жанрів. Спробуйте її пояснити. Знайдіть інші класи слів, вживання яких також відрізняються в різних жанрах.

3.7. Напишіть програму для знаходження всіх слів в корпусі Brown, які зустрічаються не менш ніж три рази.

3.8. Напишіть програму генерації таблиці відношень кількість слів / кількість оригінальних слів для всіх жанрів корпусу Brown. Проаналізуйте отримані результати та поясніть їх.

3.9. Напишіть програму для знаходження 50 найчастотніших слів в тексті, за виключенням незначущих слів.

3.10. Напишіть програму, яка виводить на екран 50 найчастотніших біграмів тексту, за виключенням біграмів, до складу яких входять незначущі слова.

3.11. Напишіть програму для створення таблиці частот слів для різних жанрів. Знайдіть слова, чия присутність або відсутність є характерною для певних жанрів (подібно до модальних дієслів).

3.12. Напишіть функцію *word_freq()*, яка приймає слово і назву частини корпусу Brown як аргументи і визначає частоту слова в заданій частині корпусу.

3.13. Визначіть функцію *hedge(text)*, яка обробляє текст і створює нову версію цього тексту, додаючи слово ‘like’ перед кожним третім словом.

4. Підготувати і оформити звіт відповідно до варіанта індивідуального завдання за своїм порядковим номером у списку групи (таблиця 4.4).

Таблиця 4.4

Варіант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Номери завдань	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3
	5	6	5	6	5	6	5	6	5	6	5	6	5	5	6
	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
	9	10	11	9	10	11	9	10	11	9	10	11	9	10	11
	12	13	12	13	12	13	12	13	12	13	12	13	12	12	13
Варіант	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Номери завдань	1	2	3	4	1	2	3	4	1	2	3	1	2	3	4
	5	6	5	6	5	6	5	6	5	6	5	6	5	5	6
	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	9	10	9	10	11	9	10	11	9	10	11	9	9	10	11
	13	12	13	12	13	12	13	12	13	12	13	12	13	12	13
	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8

Контрольні запитання

1. Операція ділення дає цілочисельні результати, як одержати результат без округлення?
2. Як називається функція, що надає доступу до тексту книжки без його поділу на окремі слова?
3. Яка функція ділить текст на окремі речення, причому кожне речення подає як список рядків?
4. Як можна визначити частоту слів для різних жанрів?
5. Назвіть методи, що дозволяють визначати, які з умов будуть відображатися на екрані за допомогою параметра *conditions = parameter*.
6. Як можна організувати доступ до власних наборів текстових файлів, який клас потрібно використати?
7. Що таке функція?
8. За допомогою якого ключового слова визначають функцію?
9. Яка послідовність запису потрібна для визначення функції?
10. Як імпортувати збережену функцію з файлу?
11. Що таке модуль?
12. Як називається множина пов’язаних між собою модулів?

Лабораторна робота № 5. Комплексне оброблення лексичних ресурсів

Мета роботи: вивчення методів і засобів доступу до лексичних ресурсів та їх комплексної обробки

Теоретичні відомості

Поняття лексичних ресурсів

Лексичні ресурси – це узагальнене поняття для всіх програмно доступних а) певним чином упорядкованих і збережених сховищ текстової інформації та б) існуючих технологічних засобів їх обробки на різних платформах, зокрема функцій, модулів, бібліотек тощо. Комплексна обробка лексичних ресурсів передбачає використання набору потрібних з них з метою розв'язання певної задачі комп'ютерної лінгвістики. Зокрема йде мова про застосування вже відомих з попередніх лабораторних робіт корпусів текстів, функцій для обробки символьних типів даних, функцій, що реалізують суто лінгвістичні поняття (біграма, конкорданс, різного роду словники, зокрема частотні та компаративні) тощо. Важливе місце у цій предметній області також займає популярна лексична база даних англійської мови WordNet.

5.1. Генерація випадкового тексту за допомогою біграм

Умовний частотний розподіл можна використати для побудови таблиці біграм (пар слів). Функція NLTK *bigrams()* як аргумент бере список слів і повертає список послідовних пар слів.

```
>>> sent = ['In', 'the', 'beginning', 'God', 'created', 'the', 'heaven',
...   'and', 'the', 'earth', '.']
>>> nltk.bigrams(sent)
[('In', 'the'), ('the', 'beginning'), ('beginning', 'God'), ('God', 'created'),
('created', 'the'), ('the', 'heaven'), ('heaven', 'and'), ('and', 'the'),
('the', 'earth'), ('earth', '.')]
```

В наступному прикладі кожне слово розглядається як умова, і для кожного з них будується частотний розподіл по словах, які йдуть після нього. Функція *generate_model()* містить простий цикл для генерації тексту. Коли ця функція викликається, то одним з її аргументів є слово – початковий контекст (у прикладі *'living'*). У циклі поточне значення змінної word виводиться на екран, а її значення замінюється на слово, яке найчастіше є наступним словом *(max())*. На наступному кроці циклу вже це слово буде поточним контекстом. Запропонований підхід генерації тексту швидко призводить до зациклювання, якого можна уникнути, якщо вибирати наступні слова випадковим чином.

```
def generate_model(cfdist, word, num=15):
    for i in range(num):
```

```
        print word,
        word = cfdist[word].max()
text = nltk.corpus.genesis.words('english-kjv.txt')
bigrams = nltk.bigrams(text)

cfd = nltk.ConditionalFreqDist(bigrams)
    >>> print cfd['living']
    <FreqDist: 'creature': 7, 'thing': 4, 'substance': 2, ',': 1, '.': 1, 'soul': 1>
    >>> generate_model(cfd, 'living')
    living creature that he said , and the land of the land of the land
```

5.2. Словники як лексичні ресурси NLTK

Лексичний ресурс (часто вживають більш зрозумілий термін словник) – це набір слів та/або словосполучень, що асоціюються з такою інформацією, як частина мови та опис значення. Лексичні ресурси є вторинними відносно текстів і зазвичай створюються та вдосконалюються з використанням текстів. Наприклад, якщо визначити текст *my_text,* тоді *vocab = sorted(set(my_text))* побудує словник тексту *my_text,* а *word_freq = FreqDist(my_text)* визначить частоту кожного слова в тексті. Тут *vocab* та *word_freq* – приклади простих лексичних ресурсів. Так само конкорданс дає інформацію про використання слів і ця інформація може бути використана в процесі побудови словників. Стандартну термінологію для словників (англ. мова) подано на рис. 5.1. Словникова стаття містить основне слово (лему) та відповідну інформацію (частина мови значення слова).

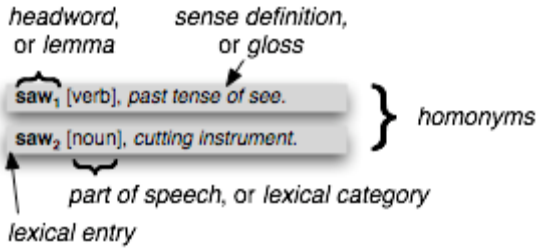


Рисунок 5.1 – Термінологія англійської мови для записів словників.

Найпростіший словник – це відсортований список слів. Досконаліші словники містять складну структуру записів та зв’язків між ними. У цій лабораторній роботі будуть розглянуті такі лексичні ресурси, які афілюються та розповсюджуються разом з NLTK.

5.3. Корпуси слів

NLTK розповсюджується з деякими корпусами, які насправді є списками слів. Корпус *words* це файл з *Unix*, який використовується для перевірки правопису. Цей список можна використати для знаходження незвичних та написаних з помилками слів в корпусі текстів, як показано в наступному прикладі:

```
def unusual_words(text):
    text_vocab = set(w.lower() for w in text if w.isalpha())
    english_vocab = set(w.lower() for w in nltk.corpus.words.words())
    unusual = text_vocab.difference(english_vocab)
    return sorted(unusual)
      >>> unusual_words(nltk.corpus.gutenberg.words('austen-sense.txt'))
      ['abbeyland', 'abhorrence', 'abominably', 'abridgement', 'accordant',
 'accustomary',
      'adieus', 'affability', 'affectedly', 'aggrandizement', 'alighted', 'allenham',
      'amiably', 'annamaria', 'annuities', 'apologising', 'arbour', 'archness', ...]
      >>> unusual_words(nltk.corpus.nps_chat.words())
      ['aaaaaaaaaaaaaaaaa', 'aaahhhh', 'abou', 'abourted', 'abs', 'ack', 'acros',
      'actualy', 'adduser', 'addy', 'adoted', 'adreniline', 'ae', 'afe', 'affari', 'afk',
      'agaibn', 'agurlwithbigguns', 'ahah', 'ahahah', 'ahahh', 'ahahha', 'ahem',
 'ahh', ...]
```

Ця програма працює за принципом фільтра. Спочатку створюється набір (словник) слів тексту, а далі з цього списку видаляються всі слова, які є в корпусі words.

В NLTK також включений корпус стоп-слів (незначущі слова). Ці слова часто зустрічаються в текстах, але, переважно, не мають окремого лексичного значення і тому видаляються з тексту під час його подальшої обробки.

```
>>> from nltk.corpus import stopwords
>>> stopwords.words('english')
['a', "a's", 'able', 'about', 'above', 'according', 'accordingly', 'across',
'actually', 'after', 'afterwards', 'again', 'against', "ain't", 'all', 'allow',
'allows', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', ...]
```

Можна визначити функцію для дослідження, який відсоток слів тексту не належить до незначущих слів.

```
>>> def content_fraction(text):
...     stopwords = nltk.corpus.stopwords.words('english')
...     content = [w for w in text if w.lower() not in stopwords]
...     return len(content) / len(text)
...
>>> content_fraction(nltk.corpus.reuters.words())
0.65997695393285261
```

За допомогою списку стоп-слів відкинуто третину слів з тексту. Ця програма працює з двома видами корпусів: текстовим корпусом і лексичним ресурсом-словником.

Список слів (корпус words) можна використати для розв’язування головоломки, що зображена на рис. 5.2.

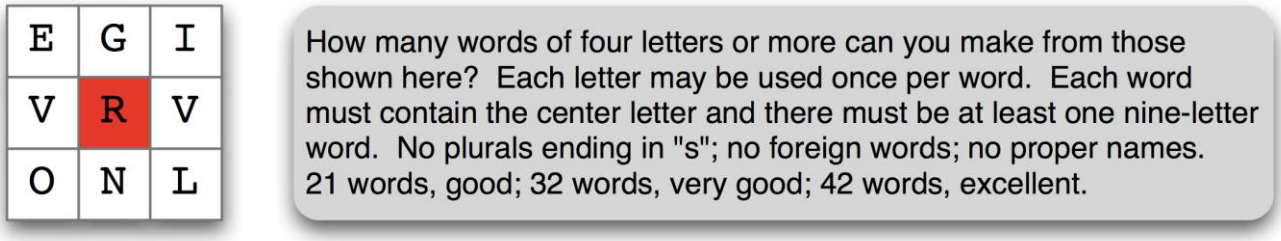


Рисунок 5.2 – Приклад головоломки

Наступна програма в циклі переглядає всі слова і перевіряє їх на відповідність умовам задачі. Перевірки чи є в слові обов'язкова літера #2 та обмеження довжини слова #1 реалізувати просто. Складніше перевірити інші літери слова, особливо з врахуванням того, що одна з них може зустрічатися два рази (*V*). Така задача вирішується використанням методу порівняння класу FreqDist #3. Частота букв у слові має бути меншою або дорівнювати частоті букв з умови задачі.

```
>>> puzzle_letters = nltk.FreqDist('egivrvonl')
>>> obligatory = 'r'
>>> wordlist = nltk.corpus.words.words()
>>> [w for w in wordlist if len(w) >= 6                    #1
...                  and obligatory in w                  #2
...                  and nltk.FreqDist(w) <= puzzle_letters]   #3
['glover', 'gorlin', 'govern', 'grovel', 'ignore', 'involver', 'lienor',
'linger', 'longer', 'lovering', 'noiler', 'overling', 'region', 'renvoi',
'revolving', 'ringle', 'roving', 'violer', 'virole']
```

У складі NLTK є окремий корпус імен обсягом 8000 одиниць, які поділені на категорії за родами (жіночим і чоловічим). Розглянемо приклад, в якому здійснюється пошук імен, що зустрічаються в обох категоріях, тобто імен, які належать і жінкам, і чоловікам.

```
>>> names = nltk.corpus.names
>>> names.fileids()
['female.txt', 'male.txt']
>>> male_names = names.words('male.txt')
>>> female_names = names.words('female.txt')
>>> [w for w in male_names if w in female_names]
['Abbey', 'Abbie', 'Abby', 'Addie', 'Adrian', 'Adrien', 'Ajay', 'Alex', 'Alexis',
'Alfie', 'Ali', 'Alix', 'Allie', 'Allyn', 'Andie', 'Andrea', 'Andy', 'Angel',
'Angie', 'Ariel', 'Ashley', 'Aubrey', 'Augustine', 'Austin', 'Averil', ...]
```

Переважно імена, які закінчуються на букву «а», належать жінкам – щоб пересвідчитися у цьому побудуємо умовний частотний розподіл (рис. 5.3).

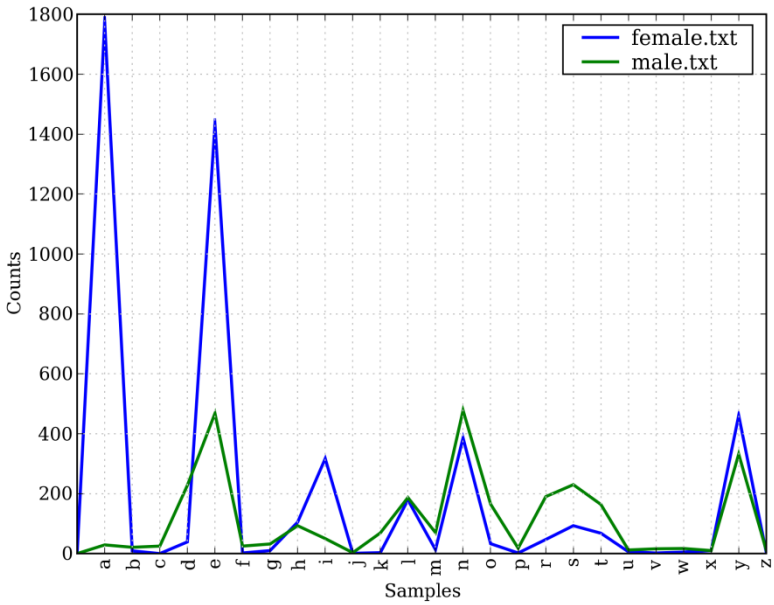


Рисунок 5.3 – Умовний частотний розподіл останніх букв чоловічих та жіночих імен

Програмний код для побудови умовного частотного розподілу:

```
 >>> cfd = nltk.ConditionalFreqDist(
...           (fileid, name[-1])
...           for fileid in names.fileids()
...           for name in names.words(fileid))
>>> cfd.plot()
```

5.4. Словник із позначенням вимови

Більш багатим лінгвістичним ресурсом може бути словник, де кожному слову поставлена у відповідність певна інформація. NLTK містить *CMU Pronouncing Dictionary* американського варіанта англійської, який розроблено для використання в синтезаторах мови.

```
>>> entries = nltk.corpus.cmudict.entries()
>>> len(entries)
127012
>>> for entry in entries[39943:39951]:
...     print entry
...
('fir', ['F', 'ER1'])
('fire', ['F', 'AY1', 'ER0'])
('fire', ['F', 'AY1', 'R'])
('firearm', ['F', 'AY1', 'ER0', 'AA2', 'R', 'M'])
('firearm', ['F', 'AY1', 'R', 'AA2', 'R', 'M'])
('firearms', ['F', 'AY1', 'ER0', 'AA2', 'R', 'M', 'Z'])
('firearms', ['F', 'AY1', 'R', 'AA2', 'R', 'M', 'Z'])
('fireball', ['F', 'AY1', 'ER0', 'B', 'AO2', 'L'])
```

Кожному слову в цьому словнику відповідає список фонетичних кодів – окремих позначень для кожного звуку (фонеми). Так, бачимо, що слово "*fire*" в американській англійській має два варіанти вимови (односкладний та двоскладний). Позначення в *CMU Pronouncing Dictionary* детально описані у http://en.wikipedia.org/wiki/Arpabet.

Кожний запис у словнику складається з двох частин, кожна з яких може оброблятися індивідуально завдяки більш складному варіанту *for* оператора. Замість *entry* в операторі *for entry in entries:* записуємо дві змінні *word*, *pron* (#1). У циклі на кожному кроці *word* відповідає першій частині запису, а *pron* – другій. Програма переглядає словник у пошуку записів, вимова яких містить три частини (#2). Якщо ця умова справджується, значення змінної *pron* присвоюється новим змінним *ph1*, *ph2*, *ph3* (#3).

```
>>> for word, pron in entries:                    #1
...     if len(pron) == 3:                         #2
...         ph1, ph2, ph3 = pron                   #3
...         if ph1 == 'P' and ph3 == 'T':
...             print word, ph2,
...
pait EY1 pat AE1 pate EY1 patt AE1 peart ER1 peat IY1 peet IY1 peete
IY1 pert ER1
pet EH1 pete IY1 pett EH1 piet IY1 piette IY1 pit IH1 pitt IH1 pot AA1
pote OW1
pott AA1 pout AW1 puett UW1 purt ER1 put UH1 putt AH1
```

Наступний приклад демонструє використання аналогічного оператора *for* у спискових висловлюваннях. Ця програма шукає слова, у вимові яких на кінці є звуки, аналогічні до *nicks*. Таку програму можна використовувати для пошуку рим для заданих слів.

```
>>> syllable = ['N', 'IH0', 'K', 'S']
>>> [word for word, pron in entries if pron[-4:] == syllable]
["atlantic's", 'audiotronics', 'avionics', 'beatniks', 'calisthenics', 'centronics',
'chetniks', "clinic's", 'clinics', 'conics', 'cynics', 'diasonics', "dominic's",
'ebonics', 'electronics', "electronics'", 'endotronics', "endotronics'", 'enix', ...]
```

Результати роботи програми показують відмінності у написанні деяких закінчень з однаковою вимовою (*nics*, *niks*, *nix*, *ntic's*).

Виконати самостійно. Здійснити аналіз таких прикладів.

```
>>> [w for w, pron in entries if pron[-1] == 'M' and w[-1] == 'n']
['autumn', 'column', 'condemn', 'damn', 'goddamn', 'hymn', 'solemn']
>>> sorted(set(w[:2] for w, pron in entries if pron[0] == 'N' and w[0] != 'n'))
['gn', 'kn', 'mn', 'pn']
```

У позначеннях звуків (фонем) використовуються цифри для позначення наголосів. Наступний приклад дозволяє доступитися до цифрових позначень і знайти у словнику слова за відповідними шаблонами наголосів.

У цьому прикладі потрібно звернути увагу на подвійне використання оператора *for*.

```
>>> def stress(pron):
...     return [char for phone in pron for char in phone if char.isdigit()]
>>> [w for w, pron in entries if stress(pron) == ['0', '1', '0', '2', '0']]
['abbreviated', 'abbreviating', 'accelerated', 'accelerating', 'accelerator',
'accentuated', 'accentuating', 'accommodated', 'accommodating',
'accommodative',
'accumulated', 'accumulating', 'accumulative', 'accumulator', 'accumulators',
...]
>>> [w for w, pron in entries if stress(pron) == ['0', '2', '0', '1', '0']]
['abbreviation', 'abbreviations', 'abomination', 'abortifacient', 'abortifacients',
'academicians', 'accommodation', 'accommodations', 'accreditation',
'accreditations',
'accumulation', 'accumulations', 'acetylcholine', 'acetylcholine',
'adjudication', ...]
```

Використовуючи умовний частотний розподіл можна знайти слова, які мають подібну вимову. Знайдемо слова, перша літера яких *P*, а складаються вони з трьох звуків (#2) – згрупуємо їх за першим і останнім звуками (#1).

```
>>> p3 = [(pron[0]+'-'+pron[2], word)
...       for (word, pron) in entries
...       if pron[0] == 'P' and len(pron) == 3]
>>> cfd = nltk.ConditionalFreqDist(p3)
>>> for template in cfd.conditions():
...     if len(cfd[template]) > 10:
...         words = cfd[template].keys()
...         wordlist = ' '.join(words)
...         print template, wordlist[:70] + "..."
...
P-CH perch puche poche peach petsche poach pietsch putsch pautsch piche pet...
P-K pik peek pic pique paque polk perc poke perk pac pock poch purk pak pa...
P-L pil poehl pille pehl pol pall pohl pahl paul perl pale paille perle po...
P-N paine payne pon pain pin pawn pinn pun pine paign pen pyne pane penn p...
P-P pap paap pipp paup pape pup pep poop pop pipe paape popp pip peep pope...
P-R paar poor par poore pear pare pour peer pore parr por pair porr pier...
P-S pearse piece posts pasts peace perce pos pers pace puss pesce pass pur...
P-T pot puett pit pete putt pat purt pet peart pott pett pait pert pote pa...
P-Z pays p.s pao's pais paws p.'s pas pez paz pei's pose poise peas paiz p...
```

Словник вимови можна використати для обробки тексту. У наступному прикладі програма знаходить вимову всіх зазначених слів.

```
>>> text = ['natural', 'language', 'processing']
>>> [ph for w in text for ph in prondict[w][0]]
['N', 'AE1', 'CH', 'ER0', 'AH0', 'L', 'L', 'AE1', 'NG', 'G', 'W', 'AH0', 'JH',
'P', 'R', 'AA1', 'S', 'EH0', 'S', 'IH0', 'NG']
```

5.5. Порівняльні (компаративні) словники

Ще одим популярним словником у NLTK є порівняльний словник (*Swadesh wordlists*), який містить 200 спільних слів для 24 мов. Мови ідентифікуються за двосимвольними кодами (ISO 639).

```
>>> from nltk.corpus import swadesh
>>> swadesh.fileids()
['be', 'bg', 'bs', 'ca', 'cs', 'cu', 'de', 'en', 'es', 'fr', 'hr', 'it', 'la', 'mk',
'nl', 'pl', 'pt', 'ro', 'ru', 'sk', 'sl', 'sr', 'sw', 'uk']
>>> swadesh.words('en')
['I', 'you (singular), thou', 'he', 'we', 'you (plural)', 'they', 'this', 'that',
'here', 'there', 'who', 'what', 'where', 'when', 'how', 'not', 'all', 'many', 'some',
'few', 'other', 'one', 'two', 'three', 'four', 'five', 'big', 'long', 'wide', ...]
```

До подібних слів з різних мов можна доступитися за допомогою методу entries(), аргументом якого є список мов.

```
>>> fr2en = swadesh.entries(['fr', 'en'])
>>> fr2en
[('je', 'I'), ('tu, vous', 'you (singular), thou'), ('il', 'he'), ...]
```

В наступному прикладі порівнюються слова романських і германських мов.

```
>>> languages = ['en', 'de', 'nl', 'es', 'fr', 'pt', 'la']
>>> for i in [139, 140, 141, 142]:
...     print swadesh.entries(languages)[i]
...
('say', 'sagen', 'zeggen', 'decir', 'dire', 'dizer', 'dicere')
('sing', 'singen', 'zingen', 'cantar', 'chanter', 'cantar', 'canere')
('play', 'spielen', 'spelen', 'jugar', 'jouer', 'jogar, brincar', 'ludere')
('float', 'schweben', 'zweven', 'flotar', 'flotter', 'flutuar, boiar', 'fluctuare')
```

5.6. WordNet – лексична база даних англійської мови

WordNet – це семантично орієнтований словник англійської мови, подібний до традиційних тезаурусів, але з більш багатою структурою. У *WordNet* слова групуються у набори синонімів або синсети, кожний із своїм визначенням і зв’язками з іншими синсетами. *WordNet 3.0* розповсюджується разом з NLTK і містить 155287 слів та 117659 синсетів. Хоча *WordNet* розроблявся для психолінгвістики, наразі словник широко використовується в NLP та в задачах інформаційного пошуку. Аналогічні

розробки для інших мов проводяться на основі документації з ресурсу http://www.globalwordnet.org/.

5.7. Значення та синоніми

Розглянемо таке речення:

(1) Benz is credited with the invention of the motorcar.

Якщо замінити слово *motorcar* на *automobil*e зміст речення не зміниться.

(2) Benz is credited with the invention of the automobile.

Можна вважати, що, оскільки заміна слів не вплинула на зміст речень, то ці слова – синоніми. Для одержання значення слова потрібно вибрати, до якої частини мови воно належить. *WordNet* містить чотири словники (іменники, дієслова, прикметники, прислівники). Знайдемо слово *motorcar* у словнику іменників:

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('motorcar')
[Synset('car.n.01')]
```

Слово motorcar має одне можливе значення, і воно ідентифікується як *car.n.01* – перший сенс іменника *car*. У *WordNet* структури типу *car.n.01* називають синсетами – це скінченні множини синонімічних слів.

```
>>> wn.synset('car.n.01').lemma_names()
['car', 'auto', 'automobile', 'machine', 'motorcar']
```

Слова в синсет об’єднані за спільним значенням, яке є однаковим для всіх слів. В синсеті вказується текстовий опис цього значення та приклад вживання слів з синсету.

```
>>> wn.synset('car.n.01').definition()
'a motor vehicle with four wheels; usually propelled by an internal
combustion engine'
>>> wn.synset('car.n.01').examples()
['he needs a car to get to work']
```

Для уникнення двозначності слова з синсету можна ідентифікувати як *car.n.01.automobile*, *car.n.01.motorcar*. Такі пари, як синсет і слово називають лемою. Можна переглянути всі леми цього синсету #1, переглянути окрему лему #2, відповідний лемі синсет #3 та ім’я леми #4.

```
>>> wn.synset('car.n.01').lemmas()                    #1
[Lemma('car.n.01.car'), Lemma('car.n.01.auto'),
Lemma('car.n.01.automobile'),
Lemma('car.n.01.machine'), Lemma('car.n.01.motorcar')]
>>> wn.lemma('car.n.01.automobile')                   #2
Lemma('car.n.01.automobile')
>>> wn.lemma('car.n.01.automobile').synset()          #3
Synset('car.n.01')
>>> wn.lemma('car.n.01.automobile').name()            #4
'automobile'
```

Слова *automobile* та *motorcar* є однозначні і входять тільки в один синсет. Слово *car* – багатозначне і входить у п’ять синсетів.

```
>>> wn.synsets('car')
[Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'), Synset('car.n.04'),
Synset('cable_car.n.01')]
>>> for synset in wn.synsets('car'):
...     print (synset.lemma_names())
...
['car', 'auto', 'automobile', 'machine', 'motorcar']
['car', 'railcar', 'railway_car', 'railroad_car']
['car', 'gondola']
['car', 'elevator_car']
['cable_car', 'car']
```

До всіх лем слова *car* можна доступитися наступним чином:

```
>>> wn.lemmas('car')
[Lemma('car.n.01.car'), Lemma('car.n.02.car'), Lemma('car.n.03.car'),
Lemma('car.n.04.car'), Lemma('cable_car.n.01.car')]
```

5.8. Ієрархія в WordNet

Синсети відповідають абстрактним поняттям, які можуть мати або не мати відповідних слів. Ці поняття зв’язуються разом в ієрархії. Деякі поняття, наприклад, *Entity*, *State*, *Event* – загальні, їх ще називають унікальними початковими поняттями. Інші є більше специфічними. Частковий приклад ієрархії понять наведено на рис. 5.4. Лінії між вузлами вказують на зв’язки (гіперонім\гіпонім), пунктирна лінія вказує, що *arteface* не є безпосереднім гіперонімом *motorcar*.

WordNet дозволяє легко переміщатися між поняттями. Наприклад, для поняття *motorcar* ми можемо переглянути поняття, які є більш специфічними (гіпонім):

```
>>> motorcar = wn.synset('car.n.01')
>>> types_of_motorcar = motorcar.hyponyms()
>>> types_of_motorcar[26]
Synset('ambulance.n.01')
>>> sorted([lemma.name() for synset in types_of_motorcar for lemma in
synset.lemmas])
['Model_T', 'S.U.V.', 'SUV', 'Stanley_Steamer', 'ambulance',
'beach_waggon',
'beach_wagon', 'bus', 'cab', 'compact', 'compact_car', 'convertible',
'coupe', 'cruiser', 'electric', 'electric_automobile', 'electric_car',
'estate_car', 'gas_guzzler', 'hack', 'hardtop', 'hatchback', 'heap',
'horseless_carriage', 'hot-rod', 'hot_rod', 'jalopy', 'jeep', 'landrover',
'limo', 'limousine', 'loaner', 'minicar', 'minivan', 'pace_car', 'patrol_car',
'phaeton', 'police_car', 'police_cruiser', 'prowl_car', 'race_car', 'racer',
'racing_car', 'roadster', 'runabout', 'saloon', 'secondhand_car', 'sedan',
```

'sport_car', 'sport_utility', 'sport_utility_vehicle', 'sports_car', 'squad_car',
'station_waggon', 'station_wagon', 'stock_car', 'subcompact',
'subcompact_car',
'taxi', 'taxicab', 'tourer', 'touring_car', 'two-seater', 'used-car', 'waggon',
'wagon']

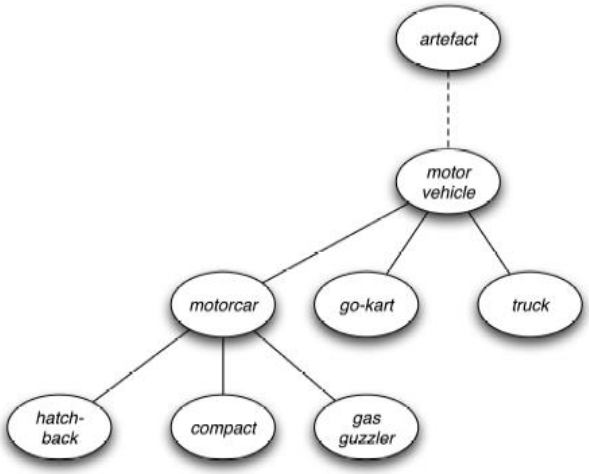


Рисунок 5.4 – Фрагмент ієрархії понять

Аналогічно можна піднятися по ієрархії і переглянути більш широкі поняття ніж *motorcar* (гіпероніми). Деякі слова мають декілька шляхів вверх, ці слова можуть класифікуватися більш ніж одним способом. Від *car.n.01* до *entity.n.01* є два шляхи, оскільки wheeled_vehicle.n.01 може розглядатися як vehicle та container.

```
>>> motorcar.hypernyms()
[Synset('motor_vehicle.n.01')]
>>> paths = motorcar.hypernym_paths()
>>> len(paths)
2
>>> [synset.name() for synset in paths[0]]
['entity.n.01', 'physical_entity.n.01', 'object.n.01', 'whole.n.02', 'artifact.n.01',
'instrumentality.n.03', 'container.n.01', 'wheeled_vehicle.n.01',
'self-propelled_vehicle.n.01', 'motor_vehicle.n.01', 'car.n.01']
>>> [synset.name() for synset in paths[1]]
['entity.n.01', 'physical_entity.n.01', 'object.n.01', 'whole.n.02', 'artifact.n.01',
'instrumentality.n.03', 'conveyance.n.03', 'vehicle.n.01',
'wheeled_vehicle.n.01',
'self-propelled_vehicle.n.01', 'motor_vehicle.n.01', 'car.n.01']
```

Початкове кореневе поняття для синсета можна переглянути наступним чином.

```
>>> motorcar.root_hypernyms()
[Synset('entity.n.01')]
```

5.9. Лексичні зв’язки у WordNet

Таблиця 5.1 містить список найбільш важливих типів зв’язків, реалізованих у *WordNet*.

Таблиця 5.1

Hypernym	more general	*animal* is a hypernym of *dog*
Hyponym	more specific	*dog* is a hypernym of *animal*
Meronym	part of	*door* is a meronym of *house*
Holonym	has part	*house* is a holonym of *door*
Synonym	similar meaning	*car* is a synonym of *automobile*
Antonym	opposite meaning	*like* is an antonym of *dislike*
Entailment	necessary action	*step* is an entailment of *walk*

Таблиця 5.2 містить повний список зв’язків іменників.

Таблиця 5.2

Relation	Also Called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	*breakfast*^{1} → *meal*^{1}
Hyponym	Subordinate	From concepts to subtypes	*meal*^{1} → *lunch*^{1}
Instance Hypernym	Instance	From instances to their concepts	*Austen*^{1} → *author*^{1}
Instance Hyponym	Has-Instance	From concepts to concept instances	*composer*^{1} → *Bach*^{1}
Member Meronym	Has-Member	From groups to their members	*faculty*^{2} → *professor*^{1}
Member Holonym	Member-Of	From members to their groups	*copilot*^{1} → *crew*^{1}
Part Meronym	Has-Part	From wholes to parts	*table*^{2} → *leg*^{3}
Part Holonym	Part-Of	From parts to wholes	*course*^{7} → *meal*^{1}
Substance Meronym		From substances to their subparts	*water*^{1} → *oxygen*^{1}
Substance Holonym		From parts of substances to wholes	*gin*^{1} → *martini*^{1}
Antonym		Semantic opposition between lemmas	*leader*^{1} ⟺ *follower*^{1}
Derivationally Related Form		Lemmas w/same morphological root	*destruction*^{1} ⟺ *destroy*^{1}

Гіперніми та гіпоніми називають лексичними зв’язками тому, що вони пов’язують один синсет з іншим. Ці два зв’язки вказують на рух вверх–вниз в ієрархії «is-a». Інший можливий шлях в ієрархії *WordNet* – це від предмета до його складових (меронім) або до поняття, яке містить предмет в собі (голоніми). Наприклад, частини дерева – стовбур, крона та ін. *part_meronyms()*. Речовина, з якого дерево зроблено, містить *heartwood* та *sapwood* – це *substance_meronyms()*. Багато дерев утворюють ліс – *member_holonyms()*.

```
>>> wn.synset('tree.n.01').part_meronyms()
[Synset('burl.n.02'), Synset('crown.n.07'), Synset('stump.n.01'),
Synset('trunk.n.01'), Synset('limb.n.02')]
>>> wn.synset('tree.n.01').substance_meronyms()
[Synset('heartwood.n.01'), Synset('sapwood.n.01')]
>>> wn.synset('tree.n.01').member_holonyms()
[Synset('forest.n.01')]
```

Прикладом складних випадків лексичних зв’язків може бути слово *mint*, яке має декілька близьких значень. Бачимо, що *mint.n.04* є частиною *mint.n.02* та речовиною, з якої зроблено *mint.n.05*.

```
>>> for synset in wn.synsets('mint', wn.NOUN):
...     print (synset.name(), ':', synset.definition())
...
batch.n.02: (often followed by `of') a large number or amount or extent
mint.n.02: any north temperate plant of the genus Mentha with aromatic leaves and  small mauve flowers
mint.n.03: any member of the mint family of plants
mint.n.04: the leaves of a mint plant used fresh or candied
mint.n.05: a candy that is flavored with a mint oil
mint.n.06: a plant where money is coined by authority of the government
>>> wn.synset('mint.n.04').part_holonyms()
[Synset('mint.n.02')]
>>> wn.synset('mint.n.04').substance_holonyms()
[Synset('mint.n.05')]
```

Для дієслів характерні зв’язки, подібні до таких:

```
>>> wn.synset('walk.v.01').entailments()
[Synset('step.v.01')]
>>> wn.synset('eat.v.01').entailments()
[Synset('swallow.v.01'), Synset('chew.v.01')]
>>> wn.synset('tease.v.03').entailments()
[Synset('arouse.v.07'), Synset('disappoint.v.01')]
```

Дія walking передбачає дію stepping – walking спричиняє stepping.
Ще один тип зв’язків, поданий в WordNet, – це антоніми:

```
>>> wn.lemma('supply.n.02.supply').antonyms()
[Lemma('demand.n.02.demand')]
>>> wn.lemma('rush.v.01.rush').antonyms()
[Lemma('linger.v.04.linger')]
>>> wn.lemma('horizontal.a.01.horizontal').antonyms()
[Lemma('vertical.a.01.vertical'), Lemma('inclined.a.02.inclined')]
>>> wn.lemma('staccato.r.01.staccato').antonyms()
[Lemma('legato.r.01.legato')]
```

5.10. Оцінення подібності в WordNet

Отже, синсети пов’язані між собою складною мережею лексичних зв’язків. Для певного синсету можна переглянути зв’язки у WordNet і знайти ті синсети, які з ним пов’язані за змістом. Інформація про семантичні взаємозв’язки між словами цінна для класифікації текстів.

Кожен синсет має один або більше шляхів, за яким він зводиться до ключового поняття *entity.n.01*. Два синсети можуть мати спільне ключове поняття і, чим нижче за ієрархією це ключове поняття, тим ближчі між собою ці два синсети.

```
>>> right = wn.synset('right_whale.n.01')
>>> orca = wn.synset('orca.n.01')
>>> minke = wn.synset('minke_whale.n.01')
>>> tortoise = wn.synset('tortoise.n.01')
>>> novel = wn.synset('novel.n.01')
>>> right.lowest_common_hypernyms(minke)
[Synset('baleen_whale.n.01')]
>>> right.lowest_common_hypernyms(orca)
[Synset('whale.n.02')]
>>> right.lowest_common_hypernyms(tortoise)
[Synset('vertebrate.n.01')]
>>> right.lowest_common_hypernyms(novel)
[Synset('entity.n.01')]
```

Зрозуміло, що *whale* – це чітко визначене поняття, тоді як *vertebrate* є більш загальним а *entity* найбільш загальним. Можна оцінити «загальність» поняття, визначивши глибину кожного синсета.

```
>>> wn.synset('baleen_whale.n.01').min_depth()
14
>>> wn.synset('whale.n.02').min_depth()
13
>>> wn.synset('vertebrate.n.01').min_depth()
8
>>> wn.synset('entity.n.01').min_depth()
0
```

Семантична подібність двох понять пов’язана з довжиною шляху між цими поняттями у *WordNet*, який містить багато засобів для здійснення таких вимірювань (*Leacock-Chodorow*, *Wu-Palmer*, *Resnik*, *Jiang-Conrath*, *Lin*). Наприклад, *path_similarity* (присвоює значення від 0 до 1) базується на найкоротшому шляху, який поєднує поняття за ієрархією гіперонімів (-1 означає, що шлях (спільний гіперонім) не знайдено).

```
>>> right.path_similarity(minke)
0.25
>>> right.path_similarity(orca)
0.16666666666666666
>>> right.path_similarity(tortoise)
0.07692307692307927
```

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями.
2. Виконати приклади, які використовуються в теоретичних відомостях.
3. Виконати такі вправи.

3.1. Дослідити зв'язки голонім – меронім для іменників. Знайти іменники для демонстрації таких зв'язків: *member_meronyms(), part_meronyms(), substance_meronyms(), member_holonyms(), part_holonyms(), та substance_holonyms()*.

3.2. Використовуючи компаративний словник, знайти близькі слова для німецької, італійської та англійської мов. Чи можуть отримані результати використовуватися для здійснення перекладу?

3.3. Побудувати умовний частотний розподіл для корпусу імен. Знайти, які перші літери частіше використовуються в чоловічих та жіночих іменах.

3.4. Здійснити аналіз словника вимов. Знайти, скільки різних слів він містить. Який відсоток слів з цього словника можуть мати різну вимову?

3.5. Який відсоток синсетів іменників не мають гіпонімів? До всіх синсетів можна доступитися за допомогою *wn.all_synsets('n')*.

3.6. Визначити функцію *supergloss(s)*, яка буде приймати синсет *s* як аргумент і повертати рядок, у якому будуть поєднані всі описи всіх значень синсету *s* та описи всіх гіпернімів та гіпонімів *s*.

3.7. Модифікувати програму генерації випадкового тексту для того, щоб зберігати можливі наступні слова у списку та вибирати їх за допомогою *random.choice()*, попередньо виконавши *import random*.

3.8. Модифікувати програму генерації випадкового тексту для того, щоб тренувати програму на текстах різних жанрів та різних корпусів. Генерацію тексту провести з 5-ма різними початковими словами. Результати проаналізувати та порівняти.

3.9. Модифікувати програму генерації випадкового тексту для того, щоб тренувати програму на текстах двох різних жанрів та, на основі цього, згенерувати текст об'єднаного жанру.

3.10. Полісемія – це явище, коли одне слово має декілька значень (наприклад, іменник *dog* має 7 значень, кількість яких визначити можна за допомогою *len(wn.synsets('dog', 'n')))*. Знайдіть середнє значення полісемії для іменників.

3.11. Полісемія – це явище, коли одне слово має декілька значень (наприклад, іменник *dog* має 7 значень, кількість яких визначити можна за допомогою *len(wn.synsets('dog', 'n')))*. Знайдіть середнє значення полісемії для прикметників.

3.12. Полісемія – це явище, коли одне слово має декілька значень (наприклад, іменник *dog* має 7 значень, кількість яких визначити можна

за допомогою *len(wn.synsets('dog', 'n')))*. Знайдіть середнє значення полісемії для дієслів.

3.13. Полісемія – це явище, коли одне слово має декілька значень (наприклад, іменник *dog* має 7 значень, кількість яких визначити можна за допомогою *len(wn.synsets('dog', 'n')))*. Знайдіть середнє значення полісемії для прислівників.

3.14. Використовуючи один з методів визначення подібності слів, побудуйте відсортований за спаданням список значень подібності для таких пар слів: *car-automobile, gem-jewel, journey-voyage, boy-lad, coast-shore, asylum-madhouse, magician-wizard, midday-noon, furnace-stove, food-fruit, bird-cock*.

3.15. Використовуючи один з методів визначення подібності слів, побудуйте відсортований за спаданням список значень подібності для таких пар слів: *bird-crane, tool-implement, brother-monk, lad-brother, crane-implement, journey-car, monk-oracle, cemetery-woodland*.

3.16. Використовуючи один з методів визначення подібності слів, побудуйте відсортований за спаданням список значень подібності для таких пар слів: *monk-oracle, cemetery-woodland, food-rooster, coast-hill, forest-graveyard, shore-woodland, monk-slave, coast-forest, lad-wizard, chord-smile, glass-magician, rooster-voyage, noon-string*.

3.17. Використовуючи один з методів визначення подібності слів, побудуйте відсортований за спаданням список значень подібності для таких пар слів: *monk-oracle, cemetery-woodland, food-rooster, coast-hill, forest-graveyard, crane-implement, journey-car, coast-shore, asylum-madhouse, magician-wizard, midday-noon, furnace-stove, food-fruit, bird-cock*.

4. Підготувати і оформити звіт відповідно до варіанта індивідуального завдання:

Варіант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Номери завдань	1	2	1	2	1	2	1	2	1	2	1	1	2	1	2
	3	5	5	3	3	5	5	3	3	5	5	3	5	5	3
	4	6	4	6	6	4	4	4	6	4	6	6	4	4	4
	7	8	9	9	7	8	7	8	9	9	7	8	9	7	8
	10	11	12	13	10	11	12	13	10	11	12	10	11	12	13
	14	15	16	17	14	15	16	17	14	15	16	14	15	16	17
Варіант	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Номери завдань	1	2	1	2	2	1	2	1	2	1	2	1	2	1	2
	3	5	5	3	5	5	3	3	5	5	3	3	5	5	3
	4	6	4	6	6	4	4	6	6	4	6	6	4	4	6
	7	8	9	9	7	8	7	8	9	9	7	8	8	7	9
	10	11	12	13	11	12	13	10	11	12	13	10	11	12	13
	14	15	16	17	15	16	17	14	15	16	17	14	15	16	17

Контрольні запитання

1. Як працює функція NLTK *bigrams()*?
2. Для чого застосовується функція *generate_model()*?
3. Як під час генерації тексту уникнути зациклювання?
4. Що таке лексичний ресурс?
5. Що можна вважати прикладом лексичного ресурсу?
6. Які засоби NLTK використовуються для перевірки правопису?
7. Що таке стоп-слова?
8. Як за допомогою словників можна відокремити множину стоп-слів?
9. Який словник призначений для опрацювання жіночих та чоловічих імен?
10. Як за допомогою словників можна визначати вимову слів?
11. За допомогою якого методу можна порівняти слова в різних мовах?
12. Що таке синсет?
13. Що таке лема?
14. Яким чином переглянути лему, що відповідає певному синсету?
15. Як оцінити «загальність» поняття, використовуючи синсет?
16. Що таке гіпоніми та гіпероніми?
17. Які типи лексичних зв’язків підтримує WordNet?
18. Як визначити семантичну подібність слів за допомогою WordNet?

Лабораторна робота № 6. Оброблення довільної текстової інформації

Мета роботи: вивчення методів методів роботи з файлами, нормалізація текстів, стемінг, лематизація та сегментація

Теоретичні відомості

Розглянемо методи та поняття комп'ютерної лінгвістики, які дозволяють опрацьовувати довільну текстову інфорацію. Виконання цієї лабораторної роботи необхідно розпочати з:

```
>>> from __future__ import division
>>> import nltk, re, pprint
```

Корпуси текстів та тексти з Інтернету є важливими джерелами даних для здійснення лінгвістичних досліджень. Звичайно, якщо дослідник має власноруч зібрані тексти, то потрібні засоби для доступу до них. Як результат виконання лабораторної роботи буде отримано відповіді на такі запитання:

1. Як написати програму для доступу до текстів з локальних файлів та з Інтернету, які є необмеженими джерелами лінгвістичних даних?
2. Як поділити текст на окремі слова та розділові знаки, з метою одержання можливості проводити подальший аналіз тексту?
3. Як написати програму для подання результатів роботи в певному форматі та зберегти їх у файлі?

6.1 Доступ до текстів з Інтернету та локальних дисків

6.1.1. Електронні книжки

Частина електронних книжок з Project Gutenberg розповсюджується разом з NLTK у вигляді корпусу текстів. Для використання інших текстів з цього проекту можна переглянути каталог 25000 електронних книжок за адресою http://www.gutenberg.org/catalog/ та встановити адресу (URL) потрібного текстового файлу в ASCII кодуванні. 90% текстів в Project Gutenberg викладено англійською мовою, але він містить також тексти більше, ніж 50-ма іншими мовами (каталонська, китайська, датська, фінська, французька, німецька, італійська, португальська, іспанська тощо).

Текст за номером 2554 – це переклад англійською *Crime and Punishment* (Злочин і кара), і отримати доступ до тексту можна таким чином:

```
from bs4 import BeautifulSoup
import urllib3

http = urllib3.PoolManager()

url = 'http://www.gutenberg.org/files/2554/2554-0.txt'
```

```
response = http.request('GET', url)
raw = str(response.data)
type(raw)
<type 'str'>
>>> len(raw)
1176831
>>> raw[:75]
'The Project Gutenberg EBook of Crime and Punishment,
by Fyodor Dostoevsky\r\n'
```

Виконання *request()* займає певний час, протягом якого відбувається завантаження цієї великої книжки.

Текст книжки збережений як значення змінної *raw*. Змінна *raw* фактично містить рядок довжиною 1,176,831 символів (перевірити тип змінної можна, скориставшись *type(raw)*). Рядок, що відповідає вмісту книжки, містить багато нецікавої для аналізу інформації: пропуски, пусті рядки, межі рядка. Символи *\r* та *\n*, які є в тексті, – це символи переводення каретки та початку нового рядка. Для подальшої роботи з текстом потрібно розділити текст на окремі слова та виділити розділові знаки (цей процес має назву токенізація). Під час застосування засобів токенізації у NLTK отримуємо список слів і розділових знаків.

```
>>> tokens = nltk.word_tokenize(raw)
>>> type(tokens)
<type 'list'>
>>> len(tokens)
255809
>>> tokens[:10]
['The', 'Project', 'Gutenberg', 'EBook', 'of', 'Crime', 'and',
'Punishment', ',', 'by']
```

Бібліотека NLTK використовувалась тільки на етапі токенізації, але не в процесі доступу за адресою в Інтернеті та під час читання рядка. Для подальшої роботи список засобами NLTK перетворюється у текст, тоді з ним можна виконувати різноманітні операції:

```
>>> text = nltk.Text(tokens)
>>> type(text)
<type 'nltk.text.Text'>
>>> text[1020:1060]
['CHAPTER', 'I', 'On', 'an', 'exceptionally', 'hot', 'evening',
'early', 'in',
'July', 'a', 'young', 'man', 'came', 'out', 'of', 'the', 'garret',
'in',
'which', 'he', 'lodged', 'in', 'S', '.', 'Place', 'and', 'walked',
'slowly',
```

```
    ',', 'as', 'though', 'in', 'hesitation', ',', 'towards', 'K', '.',
'bridge', '.']
    >>> text.collocations()
    Katerina Ivanovna; Pulcheria Alexandrovna; Avdotya
Romanovna; Pyotr
    Petrovitch; Project Gutenberg; Marfa Petrovna; Rodion
Romanovitch;
    Sofya Semyonovna; Nikodim Fomitch; did not; Hay Market;
Andrey
    Semyonovitch; old woman; Literary Archive; Dmitri
Prokofitch; great
    deal; United States; Praskovya Pavlovna; Porfiry Petrovitch;
ear rings
```

Як ми бачимо, у побудованих колокаціях зустрічається *Project Gutenberg*. Це словосполучення не міститься в тексті книжки. Завантажений текст з сайту містить метатекстову розмітку (інформацію про автора, про текст, про людей, які готували електронний варіант тощо). Ця інформація може бути як на початку тексту, так і в його кінці. Для роботи власне з текстом книжки потрібно в ручному режимі знайти межі цих додаткових даних і, за допомогою зрізів, доступитися до тексту.

```
    >>> raw.find("PART I")
    5303
    >>> raw.rfind("End of Project Gutenberg's Crime")
    1157681

    >>> raw = raw[5303:1157681]
    >>> raw.find("PART I")
    0
```

Методи *find()* та *rfind()* ("пошук з кінця") допомагають знайти потрібні індекси для їх подальшого використання у зрізах. Значення зрізу переприсвоюється змінній raw.

6.1.2. Робота з HTML файлами.

Більшість текстів в Інтернеті збережено у форматі *HTML* документів (файлів). Інтернет-сторінки можна зберігати на диску у вигляді файлів і доступатися до них. Python також дозволяє працювати з Інтернет−сторінками, безпосередньо використовуючи функцію *request*. Для прикладу переглянемо текст з *BBC News story* з назвою *Blondes to die out in 200 years*:

```
    url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
    response = http.request('GET', url)
    html = str(response.data)
    html[:60]
```

```
'<!doctype html public "-//W3C//DTD HTML 4.0
Transitional//EN'
```

Текст, який вивели на екран, містить *HTML* розмітку (метатеги, JavaScript, форми, таблиці). Вилучення тексту з *HTML* файлу доволі поширена задача, яка в *NLTK* вирішується за допомогою функції *nltk.clean_html()*. Ця функція на цей момент вважається застарілою, замість неї використовується *BeautifulSoup*.

```
>>> raw = BeautifulSoup(html)
>>>raw = raw.get_text()
>>> tokens = nltk.word_tokenize(raw)
>>> tokens
['BBC', 'NEWS', '|', 'Health', '|', 'Blondes', "'", 'to', 'die',
'out', ...]
```

Видалення іншої небажаної інформації проводиться в ручному режимі, аналогічно до попередніх прикладів з електронними книжками.

```
>>> tokens = tokens[96:399]
>>> text = nltk.Text(tokens)
>>> text.concordance('gene')
 they say too few people now carry the gene for blondes to last beyond the next tw
t blonde hair is caused by a recessive gene . In order for a child to have blonde
to have blonde hair , it must have the gene on both sides of the family in the gra
there is a disadvantage of having that gene or by chance . They don ' t disappear
ondes would disappear is if having the gene was a disadvantage and I do not think
```

6.1.3. Обробка результатів пошукових запитів

Виконати самостійно. Здійсніть аналіз результатів пошуку в Інтернеті такого словосполучення "*the of*". Чи можна аналогічним чином знайти найчастотніші колокації англійської мови?

6.1.4. Обробка RSS рядків

Блогосфера – важливе джерело текстів, як формальних, так і неформальних. За допомогою бібліотеки Python *Universal Feed Parser* (http://feedparser.org/) можна отримати доступ до вмісту блогів, як показано у цьому прикладі:

```
>>> import feedparser
>>> llog =
feedparser.parse("http://languagelog.ldc.upenn.edu/nll/?feed=atom")
>>> llog['feed']['title']
u'Language Log'
>>> len(llog.entries)
15
```

```
>>> post = llog.entries[2]
>>> post.title
u"He's My BF"
>>> content = post.content[0].value
>>> content[:70]
u'<p>Today I was chatting with three of our visiting graduate students f'
>>> raw = BeautifulSoup(llog.entries[2].content[0].value).get_text()
>>> nltk.word_tokenize(raw)
[u'Today', u'I', u'was', u'chatting', u'with', u'three', u'of', u'our', u'visiting',
u'graduate', u'students', u'from', u'the', u'PRC', u'.', u'Thinking', u'that', u'I',
u'was', u'being', u'au', u'courant', u',', u'I', u'mentioned', u'the', u'expression',
u'DUI4XIANG4', u'\u5c0d\u8c61', u'("', u'boy', u'/', u'girl', u'friend', u'"', ...]
```

6.1.5. Читання локальних файлів.

Для читання локальних файлів необхідно використовувати вбудовану функцію Python *open()* та *read()* метод. Якщо існує файл *document.txt*, то змінній *raw* можна присвоїти його вміст:

```
>>> f = open('document.txt')
>>> raw = f.read()
```

Якщо інтерпретатор не знайде файл, то видасть помилку, подібну до такої:

```
>>> f = open('document.txt')
Traceback (most recent call last):
File "<pyshell#7>", line 1, in -toplevel-
f = open('document.txt')
IOError: [Errno 2] No such file or directory: 'document.txt'
```

Для перевірки, чи дійсно файл є в потрібній директорії, у графічному інтерфейсі *IDLE* використовується команда *Open* з пункту меню *File*. Можна також перевірити вміст директорії таким чином:

```
>>> import os
>>> os.listdir('.')
```

Інша можлива проблема під час читання текстових файлів – це різні способи маркування нового рядка у файлах різних операційних систем. Під час виклику функція *open()* може містити другий параметр для контролю відкривання файлу *open('document.txt', 'rU')*. Тут „*r*” позначає файл для читання, а ”*U*” (*universal*) дозволяє ігнорувати різні способи, які використовуються для маркування нового рядка (на цей момент звернення за універсальним модифікатором вважається застарілим).

Для читання вмісту файлу можна використати багато різних методів. Метод *read()*, використаний до об'єкта файл (*f*), читає вміст файлу і подає його рядком:

```
>>> f.read()
'Time flies like an arrow.\nFruit flies like a banana.\n'
```

Символ '*n*' – це символ нового рядка.

Файл можна читати рядок за рядком, використовуючи *for*-цикл і застосовувати зріз *[:-1]* або метод *strip()* для видалення символів нового рядка:

```
>>> f = open('document.txt', 'rU')
>>> for line in f:
...     print line.strip()
Time flies like an arrow.
Fruit flies like a banana.
```

За допомогою цих методів також можна доступитися і до файлів з корпусів, які розповсюджуються з *NLTK*. Потрібно використати *nltk.data.find()* для одержання шляху до будь-якого файлу корпусу, а далі відкривати та читати файл, як показано:

```
>>> path = nltk.data.find('corpora/gutenberg/melville-moby_dick.txt')
>>> raw = open(path, 'rU').read()
```

6.1.6. Уведення тексту з клавіатури

Для уведення тексту з клавіатури (в процесі взаємодії користувача з програмою) потрібно використати функцію *raw_input()*. Після збереження введеного тексту у змінній з ним можна працювати як зі звичайним рядком.

```
>>> s = raw_input("Enter some text: ")
Enter some text: On an exceptionally hot evening early in July
>>> print "You typed", len(nltk.word_tokenize(s)), "words."
You typed 8 words.
```

6.1.7. Загальна схема роботи з текстами під час їх початкової обробки

На рис. 6.1. подано узагальнену, на основі вищевикладеного, схему початкової обробки текстів природною мовою. Початкова обробка текстів може містити такі етапи: відкривання та читання тексту з Інтернету, видалення розмітки, токенізація, перетворення тексту в NLTK текст.

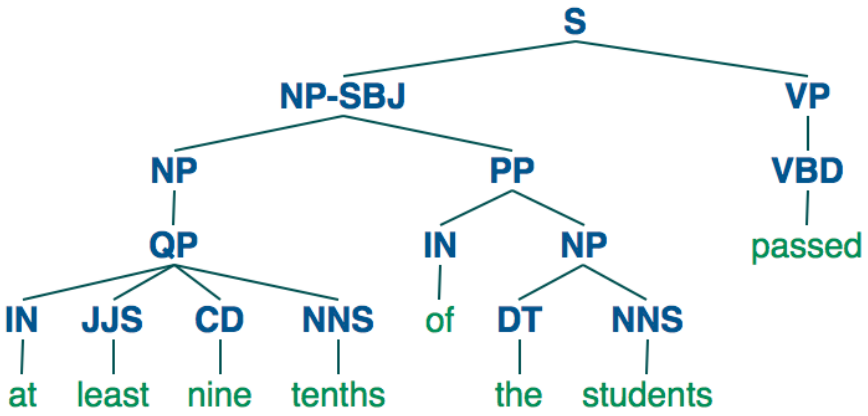


Рисунок 6.1 – Узагальнена схема початкової обробки текстів природною мовою

Коли відбувається доступ до вмісту файлу чи вмісту вебсторінки і коли видаляється HTML розмітка, то відбувається обробка рядка:

```
>>> raw = open('document.txt').read()
>>> type(raw)
<type 'str'>
```

Результат токенізації – це список, в який входять всі слова з тексту. Нормалізація та сортування цього списку приводить до отримання інших списків:

```
>>> tokens = nltk.word_tokenize(raw)
>>> type(tokens)
<type 'list'>
>>> words = [w.lower() for w in tokens]
>>> type(words)
<type 'list'>
>>> vocab = sorted(set(words))
>>> type(vocab)
<type 'list'>
```

Тип об’єкта визначає, які операції можуть бути здійснені з цим об’єктом. Наприклад, можна додавати елементи до списку, але не можна до рядка:

```
>>> vocab.append('blog')
>>> raw.append('blog')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no attribute 'append'
```

Рядки та списки можна поєднувати з іншими рядками та списками, але не можна поєднувати рядки зі списками:

```
>>> query = 'Who knows?'
>>> beatles = ['john', 'paul', 'george', 'ringo']
>>> query + beatles
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'list' objects
```

6.2. Використання Unicode в процесі оброблення текстів

Програми обробки природної мови мають працювати з різними мовами та з різними наборами символів. Твердження «1 байт = 1 символ» є застарілим і, у переважній більшості практичних випадків, є хибним. В англомовному світі переважно використовується ASCII кодування символів. В Європі використовується розширений *Latin* набір символів, який містить такі символи датської та норвежської, як "ø", угорської – "ő", іспанської та бретонської –"ñ" та "ň" чеської і словацької мов. Розглянемо, як використовується Unicode під час оброблення текстів, що містять відмінні від ASCII символи.

6.2.1. Поняття Юнікод (Unicode)

Юнікóд (англ. *Unicode*) – це промисловий стандарт, розроблений, щоб зробити можливим для текстів і символів (графічних знаків) всіх писемних систем світу узгоджене подання і обробку комп'ютерами. Юнікод підтримує більш ніж мільйон символів. Кожному символу ставиться у відповідність число, яке називають кодовою точкою. В Python кодові точки записуються у вигляді *\uXXXX*, де *XXXX* – чотири символи шістнадцяткового числа.

У межах програми обробка рядків *Unicode* відбувається аналогічно до звичайних рядків. Однак, коли *Unicode* символи зберігаються у файл або виводяться на екран, вони мають бути закодовані як потік байтів. Деякі кодування (такі як *ASCII* та *Latin-2*) використовують один байт для подання одної кодової точки і, відповідно, підтримують невеликий набір символів *Unicode*, достатній для одної мови. Інші кодування (такі як *UTF-8*) використовують послідовності байтів і можуть подати весь набір символів *Unicode*.

Текст у файлах зберігається у певному кодуванні, тому потрібен певний механізм для перетворення його до *Unicode*. Такий механізм називають декодуванням. Навпаки, записати *Unicode* символи у файл або вивести на екран можна, тільки попередньо перетворивши їх у потрібне кодування. Таке перетворення називають кодуванням (рис. 6.2).

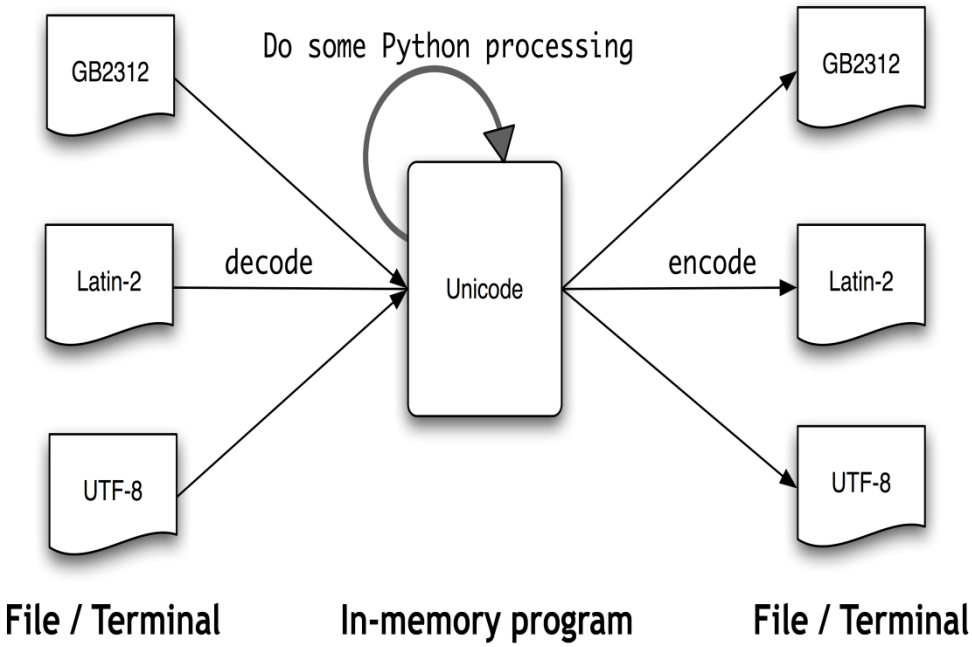


Рисунок 6.2 – Кодування і декодування Unicode

З точки зору Unicode, символи є абстрактними сутностями, які можуть бути реалізовані як один або більше піктограм (символьних знаків). Такі піктограми можуть з'являтися на екрані або друкуватися на папері. Тоді шрифт алфавіту певної мови – це відображення символів цього алфавіту у піктограми.

6.2.2. Одержання закодованого тексту з файлів

Нехай існують невеликі текстові файли відомого кодування. Це файл *Ukrainian1-Cyrillic* (перше речення декларації прав людини) з текстом українською мовою в кодуванні *Cyrillic* та файл *polish-lat2.txt* з текстом польською мовою у кодуванні *Latin-2* (ISO-8859-2). За допомогою функції *nltk.data.find()* знайдемо місцезнаходження цих файлів:

```
>>> path = nltk.data.find('corpora/udhr/Ukrainian1-Cyrillic')
>>> path1 = nltk.data.find('corpora/unicode_samples/polish-lat2.txt')
```

Модуль Python *codecs* забезпечує функції читання кодованих даних в *Unicode* рядку і запису *Unicode* рядка в кодовану форму. Функція *codecs.open()* потребує параметра кодування файлу для читання чи записування. Перед використанням модуля потрібно його імпортувати і, під час читання, вказувати тип кодування:

```
>>> import codecs
>>> f = codecs.open(path, encoding='cyrillic')
>>> f1 = codecs.open(path1, encoding='latin2')
```

Список параметрів кодування модуля *codecs* можна переглянути за адресою http://docs.python.org/lib/standard-encodings.html. Для записування даних у файл потрібно скористатись такою конструкцією: *f = codecs.open(path, 'w', encoding='utf-8')*.

Текст, прочитаний з *f*, буде в *Unicode*. Для подання цього тексту на екрані потрібно його закодувати. В Python кодування *unicode_escape* перетворює всі не *ASCII* символи в їх подання *\uXXXX*. Кодові точки вище ASCII = 127, але до 256 подаються у двоцифровій формі *\xXX*.

```
>>> for line in f1:
...     line = line.strip()
...     print line.encode('unicode_escape')
Pruska Biblioteka Pa\u0144stwowa. Jej dawne zbiory znane pod
nazw\u0105
"Berlinka" to skarb kultury i sztuki niemieckiej. Przewiezione
przez
Niemc\xf3w pod koniec II wojny \u015bwiatowej na Dolny
\u015bl\u0105sk, zosta\u0142y
odnalezione po 1945 r. na terytorium Polski. Trafi\u0142y do
Biblioteki
Jagiello\u0144skiej w Krakowie, obejmuj\u0105 ponad 500 tys.
zabytkowych
archiwali\xf3w, m.in. manuskrypty Goethego, Mozarta,
Beethovena, Bacha.
```

У першому рядку послідовність *\u0144* починається з символу *\u*. Відповідний *Unicode* символ буде відображатися на екрані як *ń*. В третьому рядку є символ *\xf3*, який відповідає *ó*, оскільки він є з проміжку 128–255.

У Python *Unicode* рядок записують, вказавши на початку символ u, (*u'hello'* – Юнікод рядок). Довільний *Unicode* символ може бути визначений всередині *Unicode* рядка, використовуючи подання *\u*XXXX. Можна знайти чисельне значення кодової точки, використовуючи функцію *ord()*:

```
>>> ord('a')
97
>>> ord('à')
224
```

В шістнадцятковій формі 97 це 0061, а 224 – це 00E0 , що дозволяє представити ці символи відповідними кодовими точками:

```
>>> a = u'\u0061'
>>> b = u'\u00E0'
>>> a
u'a'
```

```
>>> print a
a
>>> b
u'\xe0'
>>> print b
à
```

6.2.3. Використання локального кодування в Python

Під час роботи з символами певного локального кодування з метою використання стандартних методів уведення і редагування рядків, у файлі Python потрібно записати рядок *'# -*- coding: <coding> -*-'* першим або другим рядком у файлі. Тут *<coding>* – це рядок на зразок *'latin-1', 'big5' or 'utf-8'*.

6.3. Нормалізація тексту

У попередніх лабораторних роботах перед обробкою тексту всі літери слів перетворювались у малі літери (*set(w.lower() for w in text)*.). За допомогою *lower()* текст нормалізується для усунення розбіжностей між «The» та «the». Часто потрібно відділити від слів афікси. Така задача називається стемінгом. Для перевірки чи словоформа є словом у словнику, потрібно здійснити лематизацію (отримати канонічну форму лексеми):

```
>>> raw = """DENNIS: Listen, strange women lying in ponds distributing swords
... is no basis for a system of government.  Supreme executive power derives from
... a mandate from the masses, not from some farcical aquatic ceremony."""
>>> tokens = nltk.word_tokenize(raw)
```

6.3.1. Стемінг

NLTK містить декілька стандартних програм для здійснення стемінгу. *Porter* та *Lancaster* програми здійснюють стемінг (відкидання афіксів) на основі наборів правил. *Porter* стемер коректно обробляє слово *lying* (*lie*), а *Lancaster* стемер це слово обробляє з помилкою.

```
>>> porter = nltk.PorterStemmer()
>>> lancaster = nltk.LancasterStemmer()
>>> [porter.stem(t) for t in tokens]
['DENNI', ':', 'Listen', ',', 'strang', 'women', 'lie', 'in', 'pond',
'distribut', 'sword', 'is', 'no', 'basi', 'for', 'a', 'system', 'of',
'govern',
'.', 'Suprem', 'execut', 'power', 'deriv', 'from', 'a', 'mandat',
'from',
```

```
    'the', 'mass', ',', 'not', 'from', 'some', 'farcic', 'aquat',
'ceremoni', '.']
    >>> [lancaster.stem(t) for t in tokens]
    ['den', ':', 'list', ',', 'strange', 'wom', 'lying', 'in', 'pond',
'distribut',
    'sword', 'is', 'no', 'bas', 'for', 'a', 'system', 'of', 'govern', '.',
'suprem',
    'execut', 'pow', 'der', 'from', 'a', 'mand', 'from', 'the', 'mass',
',', 'not',
    'from', 'som', 'farc', 'aqu', 'ceremony', '.']
```

Результати стемінгу не є однозначними, а тому стемер вибирається той, який дає кращі результати для зазначеної задачі. *Porter* стемер найкраще використовувати для індексування деякого тексту з метою підтримки пошукових операцій (забезпечення пошуку всіх форм слова). Порівняйте результати побудови конкордансу для слова "*lie*", без індексування та з попереднім індексуванням (*class IndexedText(object)*).

```
class IndexedText(object):

    def __init__(self, stemmer, text):
        self._text = text
        self._stemmer = stemmer
        self._index = nltk.Index((self._stem(word), i)
                                 for (i, word) in enumerate(text))

    def concordance(self, word, width=40):
        key = self._stem(word)
        wc = width/4              # words of context
        for i in self._index[key]:
            lcontext = ' '.join(self._text[i-wc:i])
            rcontext = ' '.join(self._text[i:i+wc])
            ldisplay = '%*s'  % (width, lcontext[-width:])
            rdisplay = '%-*s' % (width, rcontext[:width])
            print ldisplay, rdisplay

    def _stem(self, word):
        return self._stemmer.stem(word).lower()
```

```
>>> porter = nltk.PorterStemmer()
>>> grail = nltk.corpus.webtext.words('grail.txt')
>>> text = IndexedText(porter, grail)
>>> text.concordance('lie')
r king ! DENNIS : Listen , strange women lying in ponds distributing swords is no
      beat a very brave retreat . ROBIN : All lies ! MINSTREL : [ singing ] Bravest of
          Nay . Nay . Come . Come . You may lie here . Oh , but you are wounded !
      doctors immediately ! No , no , please ! Lie down . [ clap clap ] PIGLET : Well
      ere is much danger , for beyond the cave lies the Gorge of Eternal Peril , which
        you . Oh ... TIM : To the north there lies a cave -- the cave of Caerbannog --
      h it and lived ! Bones of full fifty men lie strewn about its lair . So , brave k

not stop our fight ' til each one of you lies dead , and the Holy Grail returns t
```

```
>>> grail = nltk.Text(nltk.corpus.webtext.words('grail.txt'))
>>> grail.concordance('lie')
      Nay . Nay . Come . Come . You may lie here . Oh , but you are wounded !
doctors immediately ! No , no , please ! Lie down . [ clap clap ] PIGLET : Well
h it and lived ! Bones of full fifty men lie strewn about its lair . So , brave k
```

6.3.2 Лематизація

WordNet лематизатор видаляє афікси тільки якщо слово, яке отримується в процесі лематизації, наявне в його словнику. Ця процедура робить лематизацію повільнішою за стемінг. Слово *lying* оброблено з помилкою, але слово *women* перетворено у *woman*.

```
>>> wnl = nltk.WordNetLemmatizer()
>>> [wnl.lemmatize(t) for t in tokens]
['DENNIS', ':', 'Listen', ',', 'strange', 'woman', 'lying', 'in',
'pond',
'distributing', 'sword', 'is', 'no', 'basis', 'for', 'a', 'system',
'of',
'government', '.', 'Supreme', 'executive', 'power', 'derives',
'from', 'a',
'mandate', 'from', 'the', 'mass', ',', 'not', 'from', 'some',
'farcical',
'aquatic', 'ceremony', '.']
```

WordNet лематизатор можна використовувати для побудови словника деякого тексту. Внаслідок цього можна отримати список лем.

6.3.3. Сегментація

У комп’ютерній лінгвістиці сегментація – це найбільш загальний випадок токенізації.

6.3.4. Сегментація тексту на окремі речення

Робота з текстами на рівні окремих слів часто передбачає можливість поділу тексту на окремі речення. Деякі корпуси забезпечують можливість доступу на рівні окремих речень. В цьому прикладі визначається середня довжина речення в корпусі Brown:

```
>>> len(nltk.corpus.brown.words()) /
len(nltk.corpus.brown.sents())
20.250994070456922
```

У випадку, якщо текст подано як послідовність символів, то перед токенізацією необхідно поділити текст на окремі речення. NLTK забезпечує таку можливість за допомогою програми *Punkt* сегментації тексту на речення. У прикладі показано використання цієї програми:

```
>>> sent_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
>>> text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
>>> sents = sent_tokenizer.tokenize(text)
>>> pprint.pprint(sents[171:181])
['"Nonsense!',
 '" said Gregory, who was very rational when anyone else\nattempted paradox.',
 '"Why do all the clerks and navvies in the\nrailway trains look so sad and tired,...',
 'I will\ntell you.',
 'It is because they know that the train is going right.',
 'It\nis because they know that whatever place they have taken a ticket\nfor that ...',
 'It is because after they have\npassed Sloane Square they know that the next stat...',
 'Oh, their wild rapture!',
 'oh,\ntheir eyes like stars and their souls again in Eden, if the next\nstation w...'
 '"\n\n"It is you who are unpoetical," replied the poet Syme.']
```

Зауважимо, що цей приклад виведення одного речення у квадратних скобках – інформація про промову Mr Lucian Gregory. Промова за фактом містить декілька речень, тому доцільно їх виділити як окремі рядки.

Сегментація тексту на речення є складною процедурою, оскільки навіть такий явний розділовий знак межі речення, як крапка, може використовуватись у скороченнях, абревіатурах та їх комбінаціях.

6.3.5. Сегментація тексту на окремі слова

Для деяких писемностей токенізація тексту є складною процедурою, оскільки складно встановити межі слова. Наприклад, три символи рядка китайською 爱国人 (ai4 "love" (verb), guo3 "country", ren2 "person") можуть

бути поділені і як 爱国 / 人, "country-loving person", так і як 爱 / 国人, "love country-person."

Подібна проблема виникає в процесі обробки усного мовлення, коли слухач змушений сегментувати безперервний потік звуків на окремі слова. Особливо складно це зробити, якщо слова слухачу попередньо невідомі (під час вивчення мови або, якщо немовля сприймає мову батьків). Розглянемо наступний приклад, в якому видалені межі слів.

(1) a. doyouseethekitty
b. seethedoggy
c. doyoulikethekitty
d. likethedoggy

Спочатку маємо описати задачу: потрібно знайти спосіб відділити текст від власне сегментації. Цього можна досягнути, промаркувавши кожен символ булевим значенням, яке буде вказувати, що після символу йде межа слова. Подібний результат може отримати людина на слух, сприймаючи паузи під час вимови. Результат подамо таким чином: seg1 – початкова сегментація, seg2 – результуюча сегментація.

```
>>> text =
"doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
>>> seg1 =
"0000000000000001000000000100000000000000010000000000"
>>> seg2 =
"0100100100100001001001000010100100010010000100010010000"
```

Рядки сегментації містять нулі та одиниці, а їх довжина менша на один символ від початкового тексту (текст довжиною *N* елементів може мати тільки *N-1* місць розділу). Визначимо функцію *segment()*, яка в наступному прикладі дозволяє отримати оригінальний сегментований текст на основі поданого вище подання.

```
def segment(text, segs):
    words = []
    last = 0
    for i in range(len(segs)):
        if segs[i] == '1':
            words.append(text[last:i+1])
            last = i+1
    words.append(text[last:])
    return words
```

```
>>> text =
"doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
```

```
>>> seg1 =
"0000000000000001000000000100000000000000010000000000"
>>> seg2 =
"0100100100100001001001000010100100010010000100010010000"
>>> segment(text, seg1)
['doyouseethekitty', 'seethedoggy', 'doyoulikethekitty',
'likethedoggy']
>>> segment(text, seg2)
['do', 'you', 'see', 'the', 'kitty', 'see', 'the', 'doggy', 'do', 'you',
 'like', 'the', 'kitty', 'like', 'the', 'doggy']
```

Тепер задачу сегментації можна розглядати як пошукову задачу: знайти рядок бітів, який приводить до коректної сегментації рядка тексту на слова. Студент запам'ятовує слова і зберігає їх у словнику. Маючи відповідний словник, можливо здійснити реконструкцію початкового тексту як послідовність лексичних одиниць. Побудуємо цільову функцію (рис. 6.3), значення якої буде оптимізоване на основі розміру словника та розміру інформації, необхідної для реконструкції початкового тексту зі словника.

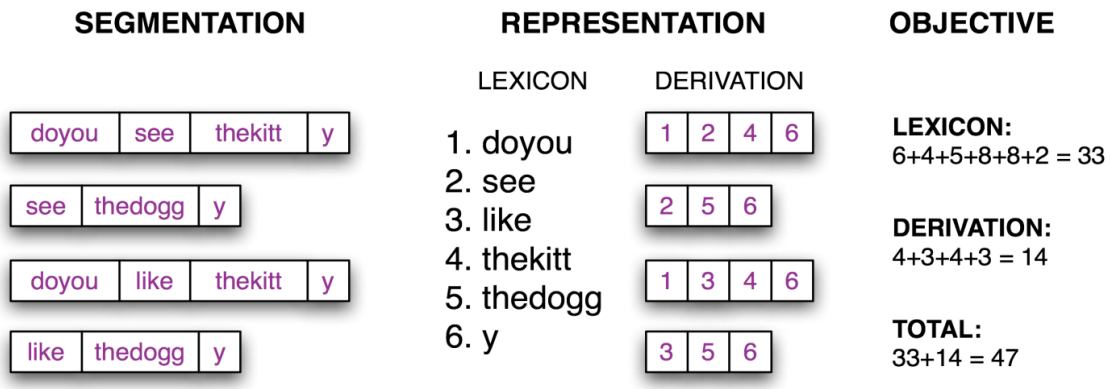


Рисунок 6.3 – Визначення цільової функції

На основі гіпотетично сегментованого початкового тексту (ліва колонка) отримуємо словник та таблицю (середня колонка), які забезпечують реконструкцію початкового тексту.

Загальна кількість символів у лексиконі (з врахуванням символу межі слова) та сума елементів в таблиці слугують чисельним значенням для оцінення якості сегментації (права колонка). Менше, за знайдене, значення вказує на кращий варіант сегментації.

Для розрахунку цієї цільової функції можна використати таку функцію:

```
def evaluate(text, segs):
    words = segment(text, segs)
    text_size = len(words)
    lexicon_size = len(' '.join(list(set(words))))
    return text_size + lexicon_size
```

```
>>> text =
"doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
>>> seg1 =
"000000000000001000000000100000000000000100000000000"
>>> seg2 =
"010010010010000100100100001010010001001000010001001000 0"
>>> seg3 =
"000010010000001100100000011000010001000000110001000000 1"
>>> segment(text, seg3)
['doyou', 'see', 'thekitt', 'y', 'see', 'thedogg', 'y', 'doyou', 'like',
 'thekitt', 'y', 'like', 'thedogg', 'y']
>>> evaluate(text, seg3)
46
>>> evaluate(text, seg2)
47
>>> evaluate(text, seg1)
63
```

Останній крок – це, власне, пошук послідовності нулів та одиниць, яка максимізує цю цільову функцію. Потрібно зазначити, що найкращий варіант сегментації містить «слово» *thekitty*, оскільки недостатньо даних для подальшої сегментації.

```
from random import randint

def flip(segs, pos):
    return segs[:pos] + str(1-int(segs[pos])) + segs[pos+1:]

def flip_n(segs, n):
    for i in range(n):
        segs = flip(segs, randint(0,len(segs)-1))
    return segs

def anneal(text, segs, iterations, cooling_rate):
    temperature = float(len(segs))
    while temperature > 0.5:
        best_segs, best = segs, evaluate(text, segs)
        for i in range(iterations):
            guess = flip_n(segs, int(round(temperature)))
            score = evaluate(text, guess)
            if score < best:
                best, best_segs = score, guess
        score, segs = best, best_segs
        temperature = temperature / cooling_rate
```

```
            print evaluate(text, segs), segment(text, segs)
        print
        return segs
```

```
    >>> text =
"doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
    >>> seg1 =
"000000000000001000000000100000000000000100000000000"
    >>> anneal(text, seg1, 5000, 1.2)
    60 ['doyouseetheki', 'tty', 'see', 'thedoggy', 'doyouliketh',
'ekittylike', 'thedoggy']
    58 ['doy', 'ouseetheki', 'ttysee', 'thedoggy', 'doy', 'o',
'ulikethekittylike', 'thedoggy']
    56 ['doyou', 'seetheki', 'ttysee', 'thedoggy', 'doyou', 'liketh',
'ekittylike', 'thedoggy']
    54 ['doyou', 'seethekit', 'tysee', 'thedoggy', 'doyou',
'likethekittylike', 'thedoggy']
    53 ['doyou', 'seethekit', 'tysee', 'thedoggy', 'doyou', 'like',
'thekitty', 'like', 'thedoggy']
    51 ['doyou', 'seethekittysee', 'thedoggy', 'doyou', 'like', 'thekitty',
'like', 'thedoggy']
    42 ['doyou', 'see', 'thekitty', 'see', 'thedoggy', 'doyou', 'like',
'thekitty', 'like', 'thedoggy']
    '000010010000000100100000001000010001000000010001000000
'
```

6.4. Форматування

6.4.1. Рядки і форматування

Відомо два шляхи виведення на екран вмісту об’єктів:

```
    >>> word = 'cat'
    >>> sentence = """hello
    ... world"""
    >>> print word
    cat
    >>> print sentence
    hello
    world
    >>> word
    'cat'
    >>> sentence
    'hello\nworld'
```

Команда *print* дозволяє отримати найбільш придатне для читання подання об’єкта. Другий спосіб – ввести назву змінної в командному рядку

– показує рядок, що може бути використаний для оновлення цього об’єкта. Важливо зрозуміти, ці два результати – це рядки, виведені на екран для користувача, які не дають ніякої інформації про фактичне внутрішнє подання об’єкта.

Існують інші шляхи для виведення на екран об’єкта як рядка символів. Вони можуть також використовуватися з метою експортування даних у файл певного формату, який буде використаний зовнішньою програмою.

Форматовані вихідні дані зазвичай містять комбінацію змінних та наперед визначених рядків. Наприклад, результати частотного розподілу можна подати як:

```
>>> fdist = nltk.FreqDist(['dog', 'cat', 'dog', 'cat', 'dog',
'snake', 'dog', 'cat'])
>>> for word in fdist:
...     print word, '->', fdist[word], ';',
dog -> 4 ; cat -> 3 ; snake -> 1 ;
```

Уникнути небажаних пропусків та покращити читабельність програми дозволяє використання згаданих уже у лабораторній роботі № 2 виразів форматування.

```
>>> for word in fdist:
...    print '%s->%d;' % (word, fdist[word]),
dog->4; cat->3; snake->1;
```

Тестування рядка, який містить вираз форматування, дозволяє глибше зрозуміти механізм форматованого виведення.

```
>>> '%s->%d;' % ('cat', 3)
'cat->3;'
>>> '%s->%d;' % 'cat'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: not enough arguments for format string
```

Спеціальні символи *%s* та *%d* вказують на тимчасові місця (позиції) для рядків і цілих (десяткових чисел). Ці символи записуються всередині рядка і використовується оператор *%* для їх комбінування, подібно до наступного прикладу:

```
>>> '%s->' % 'cat'
'cat->'
>>> '%d' % 3
'3'
>>> 'I want a %s right now' % 'coffee'
'I want a coffee right now'
```

У виразі можуть бути багато тимчасових позицій але після оператора *%* потрібно визначити кортеж з такою самою кількістю значень.

```
>>> "%s wants a %s %s" % ("Lee", "sandwich", "for
lunch")
'Lee wants a sandwich for lunch'
```

Значення для тимчасових позицій можна вказувати не безпосередньо, а так, як для прикладу зроблено у наступному циклі:

```
>>> template = 'Lee wants a %s right now'
>>> menu = ['sandwich', 'spam fritter', 'pancake']
>>> for snack in menu:
...     print template % snack
...
Lee wants a sandwich right now
Lee wants a spam fritter right now
Lee wants a pancake right now
```

Отже, символи *%s* та *%d* називають специфікаторами формату, які починаються з символу *%* і завершуються символом *s* (для рядка) або *d* (для цілих чисел). Рядок, який містить специфікатори формату, називають рядком форматування, його призначення – вивести вихідні дані на екран або на сторінку довільної ширини, такої як *%s* та *%d*.

Наприклад, можна визначити ширину як *%6s*, що дозволить отримати рядок, доповнений пропусками до ширини 6 символів. Значення змінної буде вирівняне по правому краю, а у випадку використання символу «мінус» – по лівому краю. У випадку, коли наперед невідомо, якої ширини може бути змінна, у рядку форматування використовується зірочка, яка потім визначається за допомогою окремої змінної.

```
>>> '%6s' % 'dog'
'   dog'
>>> '%-6s' % 'dog'
'dog   '
>>> width = 6
>>> '%-*s' % (width, 'dog')
'dog   '

>>> '%*s' % (15, "Monty Python")
'   Monty Python'
```

Інший спосіб контролю символів використовується для цілих чисел та чисел з плаваючою крапкою. Оскільки символ % має спеціальне значення в рядку форматування, то за необхідності використання цього символу у вихідних даних потрібно додати перед ним ще один символ % .

```
>>> count, total = 3205, 9375
>>> "accuracy for %d words: %2.4f%%" % (total, 100 * count / total)
'accuracy for 9375 words: 34.1867%'
```

Для подання даних у вигляді таблиці також необхідно використовувати рядки форматування. В наступному прикладі показано, яким чином працює функція, подана у вигляді таблиці результатів умовного частотного розподілу.

```
def tabulate(cfdist, words, categories):
    print '%-16s' % 'Category',
    for word in words:                          # column headings
        print '%6s' % word,
    print
    for category in categories:
        print '%-16s' % category,                # row heading
        for word in words:                       # for each word
            print '%6d' % cfdist[category][word],      # print table cell
        print                                    # end the row
```

```
>>> from nltk.corpus import brown
>>> cfd = nltk.ConditionalFreqDist(
...           (genre, word)
...           for genre in brown.categories()
...           for word in brown.words(categories=genre))
>>> genres = ['news', 'religion', 'hobbies', 'science_fiction', 'romance', 'humor']
>>> modals = ['can', 'could', 'may', 'might', 'must', 'will']
>>> tabulate(cfd, modals, genres)
Category          can  could   may  might  must  will
news               93     86    66     38    50   389
religion           82     59    78     12    54    71
hobbies           268     58   131     22    83   264
science_fiction    16     49     4     12     8    16
romance            74    193    11     51    45    43
humor              16     30     8      8     9    13
```

Використовуючи вираз *width = max(len(w) for w in words)* можна автоматично налаштувати ширину колонки, а кома в кінці операторів *print* не дозволяє колонкам перекриватися між собою.

6.4.2. Запис результатів у файл

Наступний приклад демонструє, яким чином відкрити файл *output.txt* для запису, та зберегти в ньому результати роботи певної програми.

```
>>> output_file = open('output.txt', 'w')
>>> words = set(nltk.corpus.genesis.words('english-kjv.txt'))
>>> for word in sorted(words):
...     output_file.write(word + "\n")
```

Виконати самостійно. Перевірити, навіщо перед записом у файл до кожного рядка додається символ *\n*. Спробуйте також використати *word + "\r\n"* вираз. Що станеться, якщо опустити ці символи *output_file.write(word)*?

У разу ззаписування у файл нетекстових даних, їх попередньо потрібно перетворити у рядок. Перетворення можна зробити, використавши рядок форматування, як було показано вище, або зробити перетворення таким чином:

```
>>> len(words)
2789
>>> str(len(words))
'2789'
>>> output_file.write(str(len(words)) + "\n")
>>> output_file.close()
```

Текстові вихідні дані часто корисно обробити для покращення їх відображення. Розглянемо наступний приклад зі складним виглядом оператора print:

```
>>> saying = ['After', 'all', 'is', 'said', 'and', 'done', ',',
...           'more', 'is', 'said', 'than', 'done', '.']
>>> for word in saying:
...     print word, '(' + str(len(word)) + '),',
After (5), all (3), is (2), said (4), and (3), done (4), , (1), more
(4), is (2), said (4), than (4), done (4), . (1),
```

Аналогічний результат, але з фіксованою довжиною рядка можна отримати з використанням модуля *textwrap*:

```
>>> from textwrap import fill
>>> format = '%s (%d),'
>>> pieces = [format % (word, len(word)) for word in
saying]
>>> output = ' '.join(pieces)
>>> wrapped = fill(output)
>>> print wrapped
After (5), all (3), is (2), said (4), and (3), done (4), , (1), more
(4), is (2), said (4), than (4), done (4), . (1),
```

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями.
2. Виконати приклади, які використовуються в теоретичних відомостях.
3. Виконати такі вправи.

3.1. Напишіть функцію, яка приймає адресу URL як аргумент, і повертає те, що міститься за цією адресою з видаленням HTML розмітки. Використовувати *urllib.urlopen* для доступу до контенту таким чином *raw_contents = urllib.urlopen('http://www.nltk.org/').read()*.

3.2. Збережіть деякий текст у файлі *corpus.txt*. Визначити функцію *load(f)* для читання файлу, назва якого є її аргументом; функція повертає рядок, що містить текст з файлу.

3.3. Перепишіть наступний цикл як *list comprehension*:

```
>>> sent = ['The', 'dog', 'gave', 'John', 'the', 'newspaper']
>>> result = []
>>> for word in sent:
...     word_len = (word, len(word))
...     result.append(word_len)
>>> result
[('The', 3), ('dog', 3), ('gave', 4), ('John', 4), ('the', 3), ('newspaper', 9)]
```

3.4. Перевірте різницю між рядками і цілими числами, виконавши такі дії: "3" * 7 та 3 * 7. Спробуйте здійснити конвертування між рядками і цілими, використавши *int("3")* та *str(3)*.

3.5. Що станеться, коли рядки форматування *%6s* та *%-6s* використовуються для відображення рядка, довшого за 6 символів?

3.6. Прочитайте деякий текст з корпусу, здійсніть його токенізацію і збережіть у список всі *wh*-слова, які в ньому зустрічаються.

3.7. Створіть файл, який буде містити слова та їх частоту, записані в окремих рядках через пропуск (наприклад, *fuzzy 53*). Прочитайте цей файл, використовуючи *open(filename).readlines()*. Розділіть кожний рядок на дві частини, використовуючи *split()*, і перетворіть число в ціле значення, використовуючи *int()*. Результат має бути у вигляді списку: *[['fuzzy', 53], ...]*.

3.8. Напишіть програму доступу до веб-сторінки з довільною адресою і вилучення з неї деякого тексту.

3.9. З тексту мовою, яка має гармонію голосних (Hungarian), вилучіть у словах послідовності голосних, а також створіть частотну таблицю біграмів голосних.

3.10. Модуль random містить функцію *choice()*, яка випадковим чином вибирає елементи послідовності. Наприклад, *choice("aehh")* буде вибирати один з чотирьох символів. Напишіть програму генерації рядка

з 500 випадково вибраних символів "aehh". Для поєднання елементів в рядок використовуйте *join()*. Нормалізуйте отриманий результат, застосувавши *split()* та *join()*.

3.11. Здійсніть аналіз чисельного виразу в такому реченні з корпусу *MedLine*: The corresponding free cortisol fractions in these sera were 4.53 +/- 0.15% and 8.16 +/- 0.23%, respectively. Чи можна сказати, що 4.53 +/- 0.15% – це три окремих слова? Чи це одне складне слово? Чи це дев'ять слів "four point five three, plus or minus fifteen percent"? Чи це взагалі не можна вважати словом? Під час вирішення яких задач потрібно вибирати ту чи іншу відповідь?

3.12. Міра оцінення читабельності використовується для оцінювання складності тексту для читання. Нехай, μ_w – середня кількість літер у слові, та μ_s – середнє значення кількості слів у реченні в певному тексті. *Automated Readability Index (ARI)* тексту визначається згідно з виразом: $4.71\ \mu_w + 0.5\ \mu_s - 21.43$. Визначіть значення *ARI* для різних частин корпусу *Brown Corpus*, включно й частину *f (popular lore)* та *j (learned)*. Використовуйте *nltk.corpus.brown.words()* для знаходження послідовності слів та *nltk.corpus.brown.sents()* для знаходження послідовності речень.

3.13. Використовуючи *Porter* стемер, нормалізуйте будь-який токенізований текст. До того самого тексту застосуйте *Lancaster* стемер. Результати порівняйте та поясніть.

3.14. Доступіться до текстів *ABC Rural News* та *ABC Science News* з корпуса (*nltk.corpus.abc*). Знайдіть значення для оцінення читабельності текстів (аналогічно задачі № 3.12). Використовуйте *Punkt* для поділу тексту на окремі речення.

3.15. Перепишіть цей цикл як *list comprehension*:

```
>>> words = ['attribution', 'confabulation', 'elocution',
...          'sequoia', 'tenacious', 'unidirectional']
>>> vsequences = set()
>>> for word in words:
...     vowels = []
...     for char in word:
...         if char in 'aeiou':
...             vowels.append(char)
...     vsequences.add(''.join(vowels))
>>> sorted(vsequences)
['aiuio', 'eaiou', 'eouio', 'euoia', 'oauaio', 'uiieioa']
```

4. Підготувати і оформити звіт відповідно до варіанта індивідуального завдання:

Варіант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Номери завдань	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	8	6	13	11	8	6	13	11	8	6	13	8	6	13	11
	9	10	12	9	10	12	9	10	12	9	10	12	9	10	12
	14	15	14	15	14	15	14	15	14	15	14	15	14	15	14

Варіант	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Номери завдань	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
	13	8	6	11	13	8	6	11	13	8	6	13	8	6	11
	9	10	12	9	10	12	9	10	12	9	10	12	9	10	12
	14	15	14	15	14	15	14	15	14	15	14	15	14	15	15

Контрольні запитання

1. Що таке токенізація?
2. Що забезпечує метод *rfind()*?
3. За допомогою якої функції можна вилучити текст з *HTML* файлу?
4. Які функції дають змогу читати локальні файли?
5. Як читати файл за принципом «рядок за рядком»?
6. Що виконує функція *raw_input()*?
7. До якого типу об’єкта можна додавати елементи?
8. Що таке лематизація?
9. Що таке нормалізація тексту?
10. Як забезпечити сегментацію тексту?
11. Що таке *Unicod*? Наведіть його основні характеристики.
12. Що таке декодування?
13. Що таке стемінг?
14. Які ви знаєте шляхи виведення вмісту об’єктів на екран?
15. Що виконує вираз *width = max(len(w) for w in words)*?

ЛІТЕРАТУРА

1. Steven Bird, Ewan Klein and Edward Loper. Natural Language Processing with Python, UPDATED FOR NLTK 3.0. – Copyright © 2014 the authors. – It is distributed with the Natural Language Toolkit [http://nltk.org/], Version 3.0, under the terms of the CC BY-NC-ND 3.0 US License.
2. Arumugam R., Shanmugamani R. Hands-On Natural Language Processing with Python. – Packt Publishing, 2018. – 312 p.
3. Саммерфилд М. Программирование на Python 3. Подробное руководство – М. , Символ-Плюс, 2009. – 608 с.
4. Дарчук Н. П. Комп'ютерна лінгвістика (автоматичне опрацювання тексту) : підручник.— К. : Видавничо-поліграфічний центр “Київський університет”, 2008. – 351 с.
5. Zsolt Nagy. Artificial Intelligence and Machine Learning Fundamentals. – Packt Publishing, 2018. – 330 p.
6. Волошин В. Г. Комп'ютерна лінгвістика. – Суми : Університетська книга, 2004. – 382 с.
7. Щербина Ю. М. Мови та автомати : методичні вказівки до лабораторної роботи № 7 / Щербина Ю. М., Висоцька В. А., Шестакевич Т. В. – Львів : Видавництво НУ ”Львівська політехніка”, 2007. – 19 с.
8. Бісікало О. В. Формальні методи образного аналізу та синтезу природно-мовних конструкцій : монографія / Бісікало О. В. – Вінниця, ВНТУ, 2013. – 316 с.
9. Хайрова Н. Ф. Информационно-лингвистические технологии экстракции и идентификации глубинных знаний в текстах : монография / Н. Ф. Хайрова, Н .В. Шаронова. – Харьков : ФЛП Коряк С. Ф., 2016. – 205 с.
10. Подоба В. Веб-розробка з Python і Django для початківців [Електронний ресурс]. – Режим доступу: http://www.vitaliypodoba.com/books/django-for-beginners/.
11. Романюк А. Б, Юрчак І. Ю. Методичні вказівки до циклу лабораторних робіт «Вивчення бібліотеки прикладних програм NLTK, для опрацювання текстів природною мовою» з дисципліни «Комп’ютерна лінгвістика» для студентів спеціальності 7.02030303 «Прикладна лінгвістика» – 2011.– Видавн. каф. САПР, 2011 р., зареєстровано у НМУ НУ «ЛП» №№3517-3525 від 02.06.2011 р. – 100 с.

Інформаційні ресурси

1. Computational Linguistic / From Wikipedia, the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Computational_linguistics.
2. Learn Python Programming Online / Copyright © 2019 Python Principles. – Режим доступу: https://pythonprinciples.com/landing/.

Електронне навчальне видання
комбінованого використання
Можна використовувати в локальному та мережному режимах

Бісікало Олег Володимирович
Севастьянов Володимир Миколайович
Богач Ілона Віталіївна

ЛАБОРАТОРНИЙ ПРАКТИКУМ

з дисципліни «Комп'ютерна лінгвістика»
для студентів спеціальності
126 – «Інформаційні системи та технології»

Лабораторний практикум

Рукопис оформив *О. Бісікало*

Редактор *Т. Старічек*

Оригінал-макет підготувала *Т. Старічек*

Підписано до видання 27.12.2022 р.
Гарнітура Times New Roman.
Зам. № Р2022-090.

Видавець та виготовлювач
Вінницький національний технічний університет,
Редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95, м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
E-mail: irvc.ed.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.