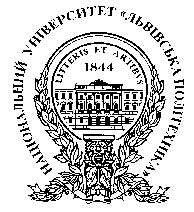
МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

Національний Університет “Львівська політехніка”



Лабораторна робота № 8

***ВИВЧЕННЯ БІБЛІОТЕКИ ПРИКЛАДНИХ ПРОГРАМ NLTK, ДЛЯ ОПРАЦЮВАННЯ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ.***

***СТРУКТУРНЕ ПРОГРАМУВАННЯ МОВОЮ PYTHON (частина2).***

Виконав:

студент групи ПРЛс-11

Форманюк А.А.

Прийняв:

Дупак Б.П.

Львів 2015

**МЕТА РОБОТИ**

* Вивчення основ програмування на мові *Python*.
* Вивчення основ структурного програмування мовою *Python*.
* Повторення та закріплення знань отриманих при виконанні попередніх лабораторних робіт.
* Покращення загальних навичок у програмуванні.

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

## *1.   Складні випадки використання функцій*

### *1.1 Функція, як аргумент*

Аргументи функцій, які розглядалися в попередніх лабораторних роботах, були простими об’єктами, такими як стрічка, або структурованими, такими як список. В Python аргументом функції також може бути і інша функція.В наступному прикладі показано, яким чином вбудована функція len() або розроблена функція last\_letter() передаються, як аргументи іншій функції:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> sent = ['Take', 'care', 'of', 'the', 'sense', ',', 'and', 'the',**  **... 'sounds', 'will', 'take', 'care', 'of', 'themselves', '.']**  **>>> def extract\_property(prop):**  **... return [prop(word) for word in sent]**  **...**  **>>> extract\_property(len)**  **[4, 4, 2, 3, 5, 1, 3, 3, 6, 4, 4, 4, 2, 10, 1]**  **>>> def last\_letter(word):**  **... return word[-1]**  **>>> extract\_property(last\_letter)**  **['e', 'e', 'f', 'e', 'e', ',', 'd', 'e', 's', 'l', 'e', 'e', 'f', 's', '.']** | |

Об’єкти len та last\_letter передаються у функцію як списки та словники (тип даних словник буде розглянуто пізніше). Зауважимо, що дужки після імені функції використовуються тільки при її виклику, а якщо функція трактується, як об’єкт (аргумент іншої функції) то дужки опускаються.

Python підтримує ще один спосіб визначення функцій як аргументів іншої функції , це так званий лямбда-вираз (анонімна функція). Спробуємо виконати дії функції last\_letter() без її створення і відповідно не використовуючи її імя. За допомогою лямбда-виразу отримаємо наступний результат:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> extract\_property(lambda w: w[-1])**  **['e', 'e', 'f', 'e', 'e', ',', 'd', 'e', 's', 'l', 'e', 'e', 'f', 's', '.']** | |

Наступний приклад ілюструє передавання функції до функції sorted(). У випадку виклику функції sorted() з одним аргументом #1 (сортування списку) ця функція використовує для порівняння вбудовану функцію cmp() (про що свідчить #2). Звичайно, можна, як аргумент, використати і власну функцію, наприклад функцію сортування в порядку спадання довжин елементів.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> sorted(sent) #1**  **[',', '.', 'Take', 'and', 'care', 'care', 'of', 'of', 'sense', 'sounds',**  **'take', 'the', 'the', 'themselves', 'will']**  **>>> sorted(sent, cmp) #2**  **[',', '.', 'Take', 'and', 'care', 'care', 'of', 'of', 'sense', 'sounds',**  **'take', 'the', 'the', 'themselves', 'will']**  **>>> sorted(sent, lambda x, y: cmp(len(y), len(x))) #3**  **['themselves', 'sounds', 'sense', 'Take', 'care', 'will', 'take', 'care',**  **'the', 'and', 'the', 'of', 'of', ',', '.']** | |

### *1.2 Функції накопичення*

Виконання функцій накопичення починається з виділення певного об’єму пям’яті. Під час здійснення ітерацій над вхідними даними, цей об’єм заповнюється і тільки після цього функція повертає результат (велику структуру або узагальнений результат). Стандартний спосіб реалізувати таку функцію це створити пустий список, в списку накопичити дані і повернути цей список , як показано у функції search1()(наступний приклад).

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **def search1(substring, words):**  **result = []**  **for word in words:**  **if substring in word:**  **result.append(word)**  **return result**  **def search2(substring, words):**  **for word in words:**  **if substring in word:**  **yield word**  **print "search1:"**  **for item in search1('zz', nltk.corpus.brown.words()):**  **print item**  **print "search2:"**  **for item in search2('zz', nltk.corpus.brown.words()):**  **print item** | |
|  |

Функція search2() – це генератор. Після першого виклику цієї функції, вона виконується аж до yield твердження і зупиняється. Програма, з якої викликалася ця функція, отримує перше слово і виконує всі необхідні дії. Тільки коли програма готова до обробки наступного слова, виконання функції продовжується з того самого місця в якому вона була зупинена і функція знову виконується аж до yield твердження. Такий підхід є типовим і більш ефективним, оскільки функція генерує дані тільки тоді коли їх потребує основна програма і не потребує виділення додаткового об’єму памяті для збереження результатів.

Розглянемо більш складний приклад генератора, який здійснює всі можливі перестановки слів у списку. Для одержання всіх можливих результатів викликаємо функцію permutations() як аргумент функції list() [#1](#listperm).

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> def permutations(seq):**  **... if len(seq) <= 1:**  **... yield seq**  **... else:**  **... for perm in permutations(seq[1:]):**  **... for i in range(len(perm)+1):**  **... yield perm[:i] + seq[0:1] + perm[i:]**  **...**  **>>> list(permutations(['police', 'fish', 'buffalo']))** [#1](#ref-listperm)  **[['police', 'fish', 'buffalo'], ['fish', 'police', 'buffalo'],**  **['fish', 'buffalo', 'police'], ['police', 'buffalo', 'fish'],**  **['buffalo', 'police', 'fish'], ['buffalo', 'fish', 'police']]** | |

### Функція використовує рекурсію – техніку яка буде розглядатися в подальшому. Перестановки слів у наборах слів корисно використовувати при створенні даних для тестування граматик.

### *1.3 Функції вищого рівня*

Python підтримує деякі функції вищого порядку, які є стандартними для мов функціонального програмування, таких як Haskell. Розглянемо ці функції разом з еквівалентними виразами , які використовують list comprehensions.

Визначимо функцію is\_content\_word(), яка перевіряє чи належить слово до відкритого класу слів. Ця функція використовується, як перший параметр функції вищого порядку filter(). Функція filter() застосовує функцію is\_content\_word() до кожного елементу послідовності, яка є її другим параметром і зберігає тільки ті елементи для яких функція is\_content\_word() повертає значення True.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> def is\_content\_word(word):**  **... return word.lower() not in ['a', 'of', 'the', 'and', 'will', ',', '.']**  **>>> sent = ['Take', 'care', 'of', 'the', 'sense', ',', 'and', 'the',**  **... 'sounds', 'will', 'take', 'care', 'of', 'themselves', '.']**  **>>> filter(is\_content\_word, sent)**  **['Take', 'care', 'sense', 'sounds', 'take', 'care', 'themselves']**  **>>> [w for w in sent if is\_content\_word(w)]**  **['Take', 'care', 'sense', 'sounds', 'take', 'care', 'themselves']** | |

Інша функція вищого порядку map(), також застосовує функцію до кожного з елементів послідовності. Наступний приклад ілюструє застосування цієї функції для одержання аналогічних з функцією extract\_property()(пункт 1.1) результатів. В прикладі показано простий спосіб знаходження середньої довжини речень в корпусі Brown Corpus, та еквівалентний варіант цих обчислень за допомогою list comprehension:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> lengths = map(len, nltk.corpus.brown.sents(categories='news'))**  **>>> sum(lengths) / len(lengths)**  **21.7508111616**  **>>> lengths = [len(w) for w in nltk.corpus.brown.sents(categories='news'))]**  **>>> sum(lengths) / len(lengths)**  **21.7508111616** | |

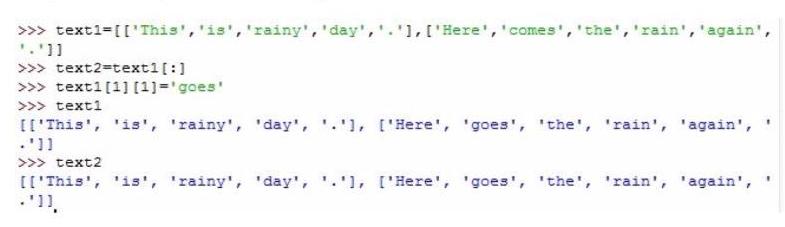
В попередніх прикладах використовувалась функція is\_content\_word() та вбудована функція len(). Наступні два приклади демонструють використання лямбда виразів в подібних програмах. В цих прикладах визначається кількість голосних в кожному зі слів вхідної послідовності.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | **>>> map(lambda w: len(filter(lambda c: c.lower() in "aeiou", w)), sent)**  **[2, 2, 1, 1, 2, 0, 1, 1, 2, 1, 2, 2, 1, 3, 0]**  **>>> [len([c for c in w if c.lower() in "aeiou"]) for w in sent]**  **[2, 2, 1, 1, 2, 0, 1, 1, 2, 1, 2, 2, 1, 3, 0]** | |

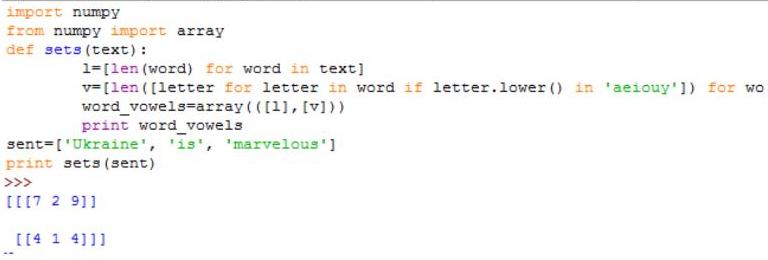
Рішення на основі list comprehensions є зазвичай більш зручні для читання і розуміння ніж рішення на основі функцій вищого порядку.

### Завдання

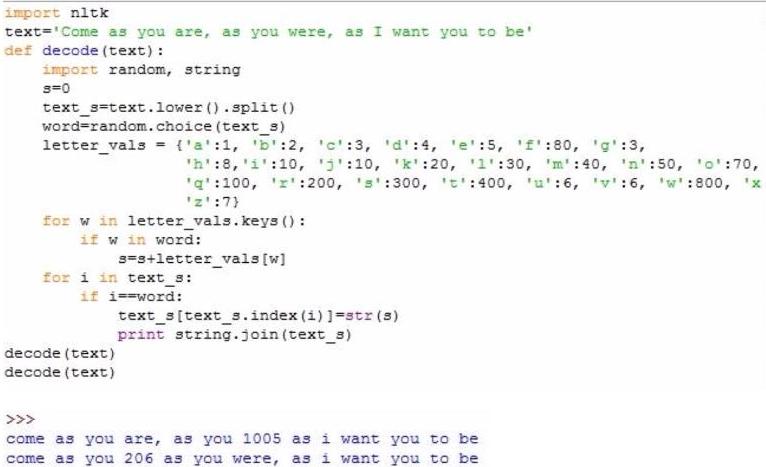
1. Створити список списків слів [ [‘’,’’,’’… ], [[‘’,’’,’’… ], [[‘’,’’,’’…],…] (наприклад текст складається з речень, які складаються з стрічок). Здійснити операцію присвоювання text2 = text1[:], та здійснити операцію присвоювання нового значення одному зі слів (text1[1][1] = 'Monty'). Перевірити як ці операції вплинули на text2. Результат письмово пояснити.



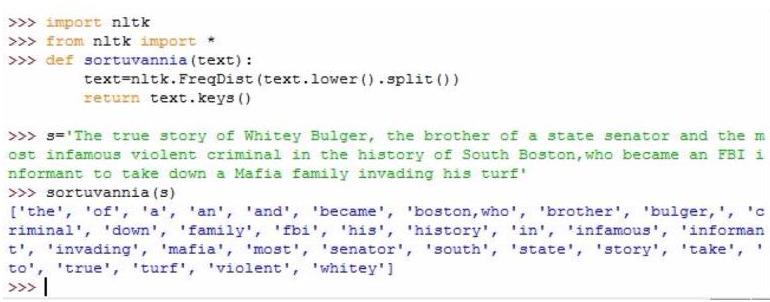
1. Написати програму для створення двовимірного масиву word\_vowels елементами якого є набори. Програма повинна обробити список слів і додати результати обробки до word\_vowels[l][v] де l – довжина слова, v – кількість голосних у слові.



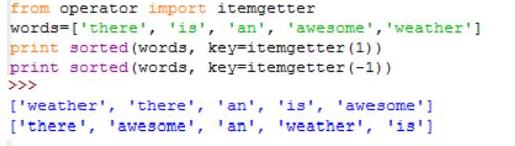
1. Гематрія – метод виявлення прихованого змісту слів на основі порівняння чисел, які відповідають словам. Слова з однаковими числами мають однаковий зміст. Число слова визначається сумуванням чисел, як відповідають його літерам. Написати функцію decode() для обробки тексту, яка випадковим чином замінює слова на їх Гематрія-еквіваленти. Чи вдалося виявити "прихований зміст" тексту? (Використовувати letter\_vals з попередньої задачі)



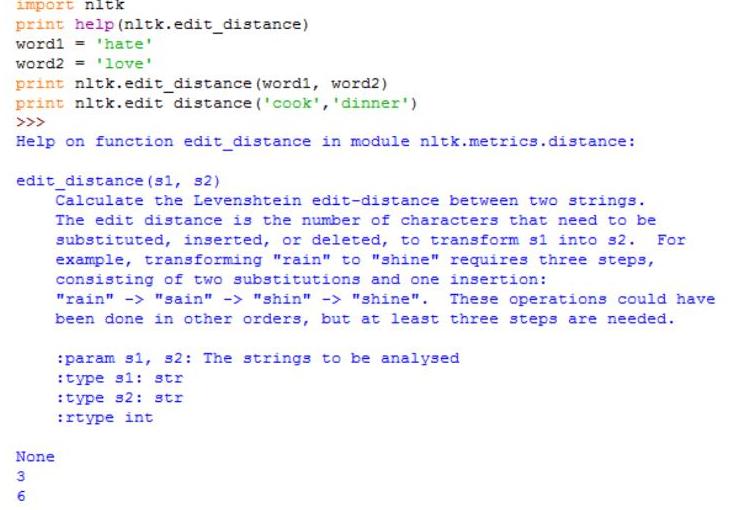
1. Написати функцію, яка обробляє список слів (з дублюванням слів) і повертає список слів (без дублювання) відсортований в порядку спадання їх частоти.



1. Імпортувати функцію itemgetter() модуля operator зі стандартної бібліотеки Python ( from operator import itemgetter). Створити список words , який містить декілька слів. Спробувати виконати: sorted(words, key=itemgetter(1)), та sorted(words, key=itemgetter(-1)). Пояснити письмово роботу функції itemgetter().



1. В NLTK реалізовано алгоритм Левінштейна для порівняння стрічок. Спробуйте скористатись цим модулем nltk.edit\_dist(). Яким чином в цьому модулі використовується динамічне програмування? Який підхід використовується знизу-вверх чи зверху-вниз? Пояснити письмово.



В цьому модулі використовується динамічне програмування . Підхід знизу-вверх. Спочатку визначається довжина стрічок, потім відстань редагування між ними.

**Висновок:** на лабораторній роботі я вивчив основи програмування на мові *Python*, основи структурного програмування мовою *Python* і повторив та закріпив знання отримані при виконанні попередніх лабораторних робіт.