МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Лабораторна робота № 7

ВИВЧЕННЯ БІБЛІОТЕКИ ПРИКЛАДНИХ ПРОГРАМ NLTK, ДЛЯ ОПРАЦЮВАННЯ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ. СТРУКТУРНЕ ПРОГРАМУВАННЯ МОВОЮ PYTHON (частина1).

Виконала:
студентка групи ПРЛс-11
Іваськів М.Є.

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:

Змінна, тип якої – кортеж, створюється за допомогою ком і переважно обмежується дужками. В попередніх лабораторних роботах кортежі використовувалися для представлення пар значень (елементів послідовності з двох членів). Зазвичай, кортежі можуть містити будь-яку кількість елементів та членів. Аналогічно до списків та стрічок, елементи кортежів можуть бути проіндексовані , до них можна доступитися за допомогою зрізів та визначити кількість елементів.

Кортежі створюються за допомогою ком (оператор - кома). Для створення кортежу, який містить єдиний елемент 'snark' , кома ставиться безпосередньо після цього елементу 'snark',. Пустий кортеж створюється за допомогою пустих дужок t=().

Ітерування елементів:

Вираз Python	**Пояснення**
for item in s	Проітерувати елементи s
for item in sorted(s)	Проітерувати впорядковані елементи s
for item in set(s)	Проітерувати унікальні елементи s
for item in reversed(s)	Проітерувати зворотньо впорядковані елементи s
for item in set(s).difference(t)	Проітерувати елементи s, які не входять в t
for item in random.shuffle(s)	Проітерувати випадково впорядковані елементи s

Послідовності різних типів можна перетворювати між собою. Наприклад, tuple(s) – перетворення послідовності будь-якого типу в кортеж, list(s) – перетворення послідовності будь-якого типу в список. Для перетворення списку стрічок в єдину стрічку потрібно використовувати функцію join(), наприклад, ':'.join(words).

Python, за допомогою таких функцій, як sorted() and reversed(), дозволяє змінювати порядок елементів у послідовностях. Також існують функції, які модифікують структуру послідовностей, що знаходить широке використання при обробці мови. Функція, zip() приймає елементи двох або більше послідовностей і "zip" переміщує їх, попарно, один з одним, в один список пар. Маючи послідовність s, і скориставшись вбудованою функцією мови програмування Python enumerate(s) отримуємо пари, які містять індекс та елемент послідовності, який відповідає цьому індексу.

Одне і те саме завдання може бути виконане різними шляхами з використанням різних змінних та з різною ефективністю. Інший критерій, який впливає та розробку програм, це стиль програмування.

При створенні добре структурованих програми, зазвичай, широко використовують функції. У випадку коли блок (окрема частина) програми довший за 10-20 стрічок то його доцільно оформити як окрему функцію.

Стрічка документування може містити **doctest блок**, в якому ілюструються застосування функції та одержання очікуваного результату. Цей блок дозволяє автоматично тестувати функцію за допомогою модуля Python's docutils, який розповсюджується окремо. В стрічці

документування документується тип кожного параметру в функції і тип параметрів які вона повертає. Як мінімум це має бути звичайний текстовий опис. Проте, в NLTK може використовуватися "epytext" – мова розмітки для документування параметрів. Опис в цьому форматі може автоматично конвертуватися в багато структуровану API документацію (дивитися http://www.nltk.org/), і містить спеціальну обробку певних полів, таких як @param , які дозволяють чітко документувати вхідні та вихідні дані функції. Наступний приклад ілюструє побудову повної стрічки документації.

ВИКОНАННЯ ЗАВДАНЬ:

Завдання 1. Знайти в Python's help додаткову інформацію про послідовності. В інтерпретаторі, набрати по черзі help(str), help(list), та help(tuple). На екрані буде відображено повний список функцій властивих кожному з типів. Деякі функції мають спеціальні імена з подвійними підкреслюваннями. Кожній такій функції відповідає і інший запис показаний в документації. Наприклад x.__getitem__(y) відповідає x[y].

```
class str(basestring)
 |  str(object) -> string
 |
 |  Return a nice string representation of the object.
 |  If the argument is a string, the return value is the same object.
 |
 |  Method resolution order:
 |      str
 |      basestring
 |      object
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
 |
 |  __eq__(...)
 |      x.__eq__(y) <==> x==y
 |
 |  __format__(...)
 |      S.__format__(format_spec) -> string
 |
 |      Return a formatted version of S as described by format_spec.
```

```
>>> help(list)
Help on class list in module __builtin__:

class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
 |
 |  __delitem__(...)
 |      x.__delitem__(y) <==> del x[y]
 |
 |  __delslice__(...)
 |      x.__delslice__(i, j) <==> del x[i:j]
 |
 |      Use of negative indices is not supported.
 |
 |  __eq__(...)
 |      x.__eq__(y) <==> x==y
 |
```

```
>>> help(tuple)
Help on class tuple in module __builtin__:

class tuple(object)
 |  tuple() -> empty tuple
 |  tuple(iterable) -> tuple initialized from iterable's items
 |
 |  If the argument is a tuple, the return value is the same object.
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
 |
 |  __eq__(...)
 |      x.__eq__(y) <==> x==y
 |
 |  __ge__(...)
 |      x.__ge__(y) <==> x>=y
 |
 |  __getattribute__(...)
 |      x.__getattribute__('name') <==> x.name
 |
```

Завдання 2. Знайти три операції, які можна здійснювати і зі списками та із кортежами. Знайти три операції, які не можна здійснювати над кортежами. Знайдіть коли використання списку замість кортежу приводить до Python помилки.

```
list = ['x', 'y']
list.hash()


#operations used both for list and for tuple:
    #__add__(...), __contains__, __len__(...)

#operations cannot be used for tuples:
    #__delitem__(...), __delslice__(...), append(...)

>>>

Traceback (most recent call last):
  File "E:\Моє\Університ\Комп'ютерна лінгвістика\лаба 7\7-2.py", line 2, in <module
>
    list.hash()
AttributeError: 'list' object has no attribute 'hash'
```

Завдання 3. Яким чином можна створити кортеж з одного елемента. Продемонструвати два різні способи.

```
#creating a one-element tuple by adding a coma
tup2 = 'we',
print tup2
#turning str into one-element tuple
str = 'element'
print tuple(str)[0:1]
#creating one-element tuple from 2 lists
word = ['Hello']
word2 = ['there']
print zip(word, word2)
>>>
('we',)
('e',)
[('Hello', 'there')]
```

Завдання 4. Створити список words = ['is', 'NLP', 'fun', '?']. Використовуючи операції присвоювання подібні до words[1] = words[2] та тимчасову змінну tmp перетворити цей список в список ['NLP', 'is', 'fun', '!']. Здійснити аналогічні перетворення використовуючи присвоювання в кортежах.

```
words = ['is', 'NLP', 'fun', '?']
print words

print 'list'
tmp = words[0]
words[0] = words[1]
words[1] = tmp
words[3] = '!'
new = words
print new

print 'tuple'
word = [(1, 'is'), (2, 'NLP'), (3, 'fun'), (4, '?')]
tmp = word[0]
word[0] = words[1]
word[1] = tmp
word[3] = '!'
news = word
print news
```

```
>>>
['is', 'NLP', 'fun', '?']
list
['NLP', 'is', 'fun', '!']
tuple
['is', (1, 'is'), (3, 'fun'), '!']
```

Завдання 5. Прочитати про вбудовану функцію здійснення порівнянь cmp, набравши help(cmp). Продемонструвати чим поведінка цієї функції відрізняється від поведінки операторів порівняння.

```
>>> help(cmp)
Help on built-in function cmp in module __builtin__:

cmp(...)
    cmp(x, y) -> integer

    Return negative if x<y, zero if x==y, positive if x>y.
```

```
print 'cmp: '
a = cmp(2,5)
print a
b = cmp(5,5)
print b
c = cmp('hello', 'bye')
print c
lst = ['hello'] *5
lst2 = ['bye']*4
print cmp (lst, lst2)

print 'other operations: '
d = 2 < 5
print d
e = 2 > 5
print e
f = 5==5
print f
g = 55
h = 65
print g is h
lst3 = ['hello']*5
lst4 = ['bye']*4
print lst3 is lst4
```

```
>>>
cmp:
-1
0
1
1
other operations:
True
False
True
False
False
```

Завдання 6. Написати програму для коректного виділення в тексті n-грамів з врахуванням граничних випадків: $n = 1$, та n = len(sent)?

```
sent = ['there','is','a','great','park','in','Lviv', '!']
print 'sentence: ', sent
n = 1
print 'n = 1: ', [sent[i:i+n] for i in range(len(sent)-n+1)]
n = 3
print 'n = 3: ', [sent[i:i+n] for i in range(len(sent)-n+1)]
n = len(sent)
print 'n = len(sent): ', [sent[i:i+n] for i in range(len(sent)-n+1)]
```

```
>>>
sentence:  ['there', 'is', 'a', 'great', 'park', 'in', 'Lviv', '!']
n = 1:  [['there'], ['is'], ['a'], ['great'], ['park'], ['in'], ['Lviv'], ['!']]
n = 3:  [['there', 'is', 'a'], ['is', 'a', 'great'], ['a', 'great', 'park'], ['g
reat', 'park', 'in'], ['park', 'in', 'Lviv'], ['in', 'Lviv', '!']]
n = len(sent):  [['there', 'is', 'a', 'great', 'park', 'in', 'Lviv', '!']]
```

Завдання 7. Використати оператори нерівності для порівняння стрічок, наприклад. 'Monty' < 'Python'. Що станеться, якщо виконати 'Z' < 'a'? Порівняти стрічки,як мають однаковий префікс, наприклад 'Monty' < 'Montague'. Спробувати порівняти структуровані об'єкти ,наприклад. ('Monty', 1) < ('Monty', 2). Чи отримали очікувані результати?

```
a = 'Monthy' < 'Python'
print a
b = 'Z' < 'a'
print b
c = 'Monthy' < 'Montague'
print c
d = ('Monty', 1) < ('Monty', 2)
print d
>>>
True
True
False
True
```

Завдання 8. Написати програму видалення пробілів на початку і в кінці стрічки та для видалення зайвих пробілів між словами. Використовувати split() та join(). Оформити у вигляді функції. Функція повинна містити повну стрічку документування.

```
str = '     I   live   in   Ukraine       '
print 'before any changes: ', str
def no_spaces(str):
    """
The funcion deletes unneeded spaces in the beginning and in the end of the sente
"""
    well = str.split()
    well = ' '.join(well)
    return well
print 'after deleting spaces: ', no_spaces(str)
print help(no_spaces)

>>>
before any changes:       I   live   in   Ukraine
after deleting spaces:  I live in Ukraine
Help on function no_spaces in module __main__:

no_spaces(str)
    The funcion deletes unneeded spaces in the beginning and in the end of the s
entence. Deletes unnecessary spaces between words.
```

Завдання 9. Написати програму видалення пробілів на початку і в кінці стрічки та для видалення зайвих пробілів між словами. Використовувати re.sub() . Оформити у вигляді функції. Функція повинна містити повну стрічку документування

```
import nltk
import re
str = '     I  live   in    Ukraine     '
def no_spaces(str):
    """
Deletes unnecessary spaces in the sentence
"""
    well = re.sub("^\s+","",str)
    well = re.sub("\s+$","",well)
    well = re.sub("\s{2,3}"," ", well)
    return well
print no_spaces(str)
print help(no_spaces)
>>>
I live in  Ukraine
Help on function no_spaces in module __main__:

no_spaces(str)
    Deletes unnecessary spaces in the sentence
```

Завдання 10. Написати програму сортування слів за їх довжиною. Визначити допоміжну функцію cmp_len, яка буде використовувати функцію cmp для порівняння довжин слів. Функція повинна містити повну стрічку документування.

```
def cmp_len(word1,word2):
    """Finds longest word"""
    return cmp(len(word1),len(word2))

def sort_by_len(input_list):
    """Sorts list"""
    while cmp_len(input_list[0], input_list[1])==-1:
            for i in range(len(input_list)-1):
                if cmp_len(input_list[i], input_list[i+1])==-1: tmp = input_list[i]
                elif cmp_len(input_list[i], input_list[i+1])==1: tmp = input_list[i];
>>>
>>> listout = ['never', 'have', 'I', 'visited','Holland']
>>> sort_by_len(listout)
>>> listout
['I','have', 'never', 'visited', 'Holland']
```