

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»  
Кафедра САПР

ЗВІТ

до лабораторної роботи № 7

на тему:

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

СТРУКТУРНЕ ПРОГРАМУВАННЯ МОВОЮ PYTHON (частина 1)

з дисципліни “Комп’ютерна лінгвістика”

Виконала:

Студентка групи ПРЛм-12

Рибчак Х. В.

Перевірив:

Асистент кафедри САПР

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

Виконання даної лабораторної роботи дозволить знайти відповіді на такі основні питання:

1. Яким чином писати гарно структуровані, чіткі, сприйнятливі (читабельні) програми, придатні для неодноразового використання?
2. Яким чином працюють основні конструкційні блоки, а саме: цикл, функція, присвоювання?
3. Які можуть бути пастки при програмуванні на Python та як їх уникати?

### 1. Основи програмування

#### 1.1 Присвоювання

Присвоювання – найпростіше поняття програмування, але навіть і йому властиві певні тонкощі.

#### 1.2 Порівняння

Python підтримує два способи порівняння. Оператор `is` перевіряє об'єкти на ідентичність. Створивши список з декількох копій одного і того самого об'єкту не складно переконатися, що елементи цього списку не тільки ідентичні, згідно `==`, але і є одним і тим самим об'єктом.

#### 1.3 Умовні твердження(висловлювання)

В частині умов `if` твердження, не пусті стрічки вважаються «true», а пусті стрічки чи списки вважаються «false» і не обробляються.

## 2. Послідовності

В попередніх прикладах використовувалися наступні послідовності: стрічки, списки та кортежі. Змінна, тип якої – кортеж, створюється за допомогою `ком` (#1) 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 (#2), до них можна доступитися за допомогою зрізів (#3) та визначити кількість елементів (#4).

## 3. Стиль програмування

Програмування це більше мистецтво ніж наука. Одна з найгрунтовніших книг по програмуванню, написана Дональдом Кнутом (Donald Knuth) так і називається *The Art of Computer Programming*. Приклади програм у більшості підручників написані таким чином, щоб не тільки комп'ютери але і люди змогли прочитати і зрозуміти текст програми. При програмуванні важливе значення має стиль програмування, який впливає на розуміння тексту програми і передбачає такі складові, як розміщення тексту програми, вибір процедурного чи декларативного стилю, використання змінних в циклах та багато інших.

# ТЕКСТИ ПРОГРАМ НА МОВІ PYTHON

## ВАРІАНТ №8

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]`.

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

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
|
```

```
>>> 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]
|
```

```
>>> 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
|
```

Рис. 1. Результат завдання №1 (фрагменти).

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.

```
spysok = ['Have', 'a', 'nice', 'day', '!']
kortezh = 'Have', 'a', 'nice', 'day', '!'
#операції, які можна здійснювати зі списками та кортежами
#1
a = spysok[2]
b = kortezh[2]
print '1) ', a, b
#2
a = spysok[1:4]
b = kortezh[1:4]
print '2) ', a, b
#3
print '3) ', 'a' in spysok, 'a' in kortezh
#операції, які не можна здійснювати над кортежами
#1
#kortezh[2] = 'bad'
#Traceback (most recent call last):
#  File "D:/5 курс/Комп'ютерна лінгвістика/Звіти/Лаба7/2.py", line 16, in <m
#    kortezh[2] = 'bad'
#TypeError: 'tuple' object does not support item assignment
#2
#print kortezh.append('today')
#Traceback (most recent call last):
#  File "D:/5 курс/Комп'ютерна лінгвістика/Звіти/Лаба7/2.py", line 22, in <m
#    print kortezh.append('today')
#AttributeError: 'tuple' object has no attribute 'append'
#3
#print kortezh.reverse()
#Traceback (most recent call last):
#  File "D:/5 курс/Комп'ютерна лінгвістика/Звіти/Лаба7/2.py", line 28, in <m
#    print kortezh.reverse()
#AttributeError: 'tuple' object has no attribute 'reverse'
#використання списку замість кортежу призводить до помилки
print kortezh + kortezh
print spysok + kortezh

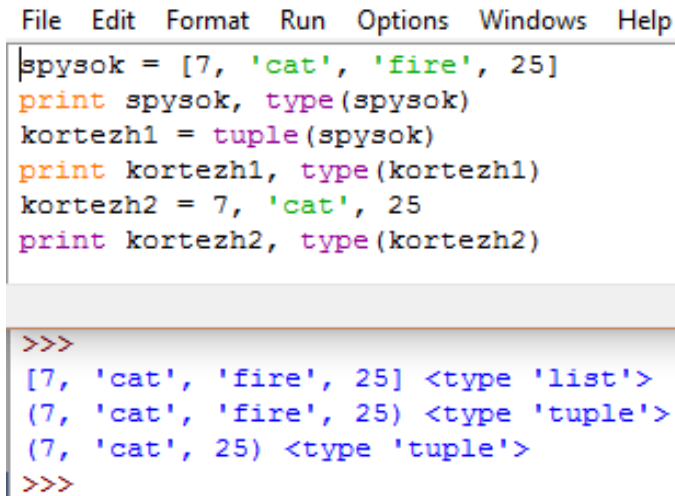
>>>
1) nice nice
2) ['a', 'nice', 'day'] ('a', 'nice', 'day')
3) True True
('Have', 'a', 'nice', 'day', '!', 'Have', 'a', 'nice', 'day', '!')

Traceback (most recent call last):
  File "D:/5 курс/Комп'ютерна лінгвістика/Звіти/Лаба7/2.py", line 35, in <module
>
    print spysok + kortezh
TypeError: can only concatenate list (not "tuple") to list
```

Рис. 2. Результат завдання №2.

3. Яким чином можна створити кортеж з одного елемента.

Продемонструвати два різні способи.

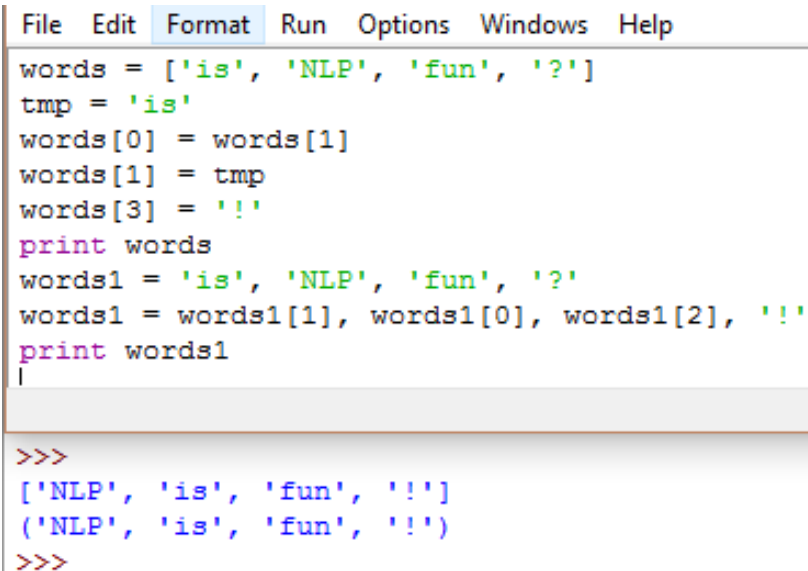


```
File Edit Format Run Options Windows Help
spysok = [7, 'cat', 'fire', 25]
print spysok, type(spysok)
kortezh1 = tuple(spysok)
print kortezh1, type(kortezh1)
kortezh2 = 7, 'cat', 25
print kortezh2, type(kortezh2)

>>>
[7, 'cat', 'fire', 25] <type 'list'>
(7, 'cat', 'fire', 25) <type 'tuple'>
(7, 'cat', 25) <type 'tuple'>
>>>
```

Рис. 3. Текст програми №3.

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.



```
File Edit Format Run Options Windows Help
words = ['is', 'NLP', 'fun', '?']
tmp = 'is'
words[0] = words[1]
words[1] = tmp
words[3] = '!'
print words
words1 = 'is', 'NLP', 'fun', '?'
words1 = words1[1], words1[0], words1[2], '!'
print words1

>>>
['NLP', 'is', 'fun', '!']
('NLP', 'is', 'fun', '!')
>>>
```

Рис. 4. Текст програми №4.

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

```
a = 45
b = 25
print 'a > b', a > b
print 'a > b', cmp(a, b)
print 'a < b', a < b
print 'a < b', cmp(b, a)
a = 25
print 'a = b', a == b
print 'a = b', cmp(a, b)
```

```
>>>
a > b True
a > b 1
a < b False
a < b -1
a = b True
a = b 0
>>>
```

Рис. 5. Текст програми №5.

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

File Edit Format Run Options Windows Help

```
sent = ['The', 'dog', 'gave', 'John', 'the', 'newspaper']
n = int(raw_input('Enter the number of elements in the n-gram! n = '))
grams = [sent[i:i+n] for i in range(len(sent)-n+1)]
print grams
```

```
>>>
Enter the number of elements in the n-gram! n = 6
[['The', 'dog', 'gave', 'John', 'the', 'newspaper']]
>>> ===== RESTART =====
==
>>>
Enter the number of elements in the n-gram! n = 1
[['The'], ['dog'], ['gave'], ['John'], ['the'], ['newspaper']]
>>> ===== RESTART =====
==
>>>
Enter the number of elements in the n-gram! n = 3
[['The', 'dog', 'gave'], ['dog', 'gave', 'John'], ['gave', 'John', 'the'],
['John', 'the', 'newspaper']]
>>>
```

Рис. 6. Текст програми №6, спосіб 1.



```
File Edit Format Run Options Windows Help
from nltk import ngrams
from nltk import re
raw = 'The cat and a dog are in the yard!'
text = re.split(r'\s+', raw)
n = int(raw_input('Enter the number of elements in the n-gram, n = '))
print ngrams(text, n)
|

Ln: 7 Col: 4

>>>
Enter the number of elements in the n-gram, n = 1
[('The',), ('cat',), ('and',), ('a',), ('dog',), ('are',), ('in',), ('the',), ('yard!',)]
>>> ===== RESTART =====
>>>
Enter the number of elements in the n-gram, n = 5
[('The', 'cat', 'and', 'a', 'dog'), ('cat', 'and', 'a', 'dog', 'are'), ('and', 'a', 'dog', 'are', 'in'), ('a', 'dog', 'are', 'in', 'the'), ('dog', 'are', 'in', 'the', 'yard!')]
>>> ===== RESTART =====
>>>
Enter the number of elements in the n-gram, n = 9
[('The', 'cat', 'and', 'a', 'dog', 'are', 'in', 'the', 'yard!')]
>>>

Ln: 49 Col: 4
```

Рис. 7. Текст програми №6, спосіб 2.

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?

```
File Edit Format Run Options Windows Help
print 'Monty' < 'Python'
print 'Z' < 'a'
print 'a' < 'b'
print 'Monty' < 'Montague'
print ('Monty', 1) < ('Monty', 2)
print ('Monty', 2) < ('Monty', 1)

>>>
True
True
True
False
True
False
>>>
```

Рис. 8. Текст програми №7.

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.

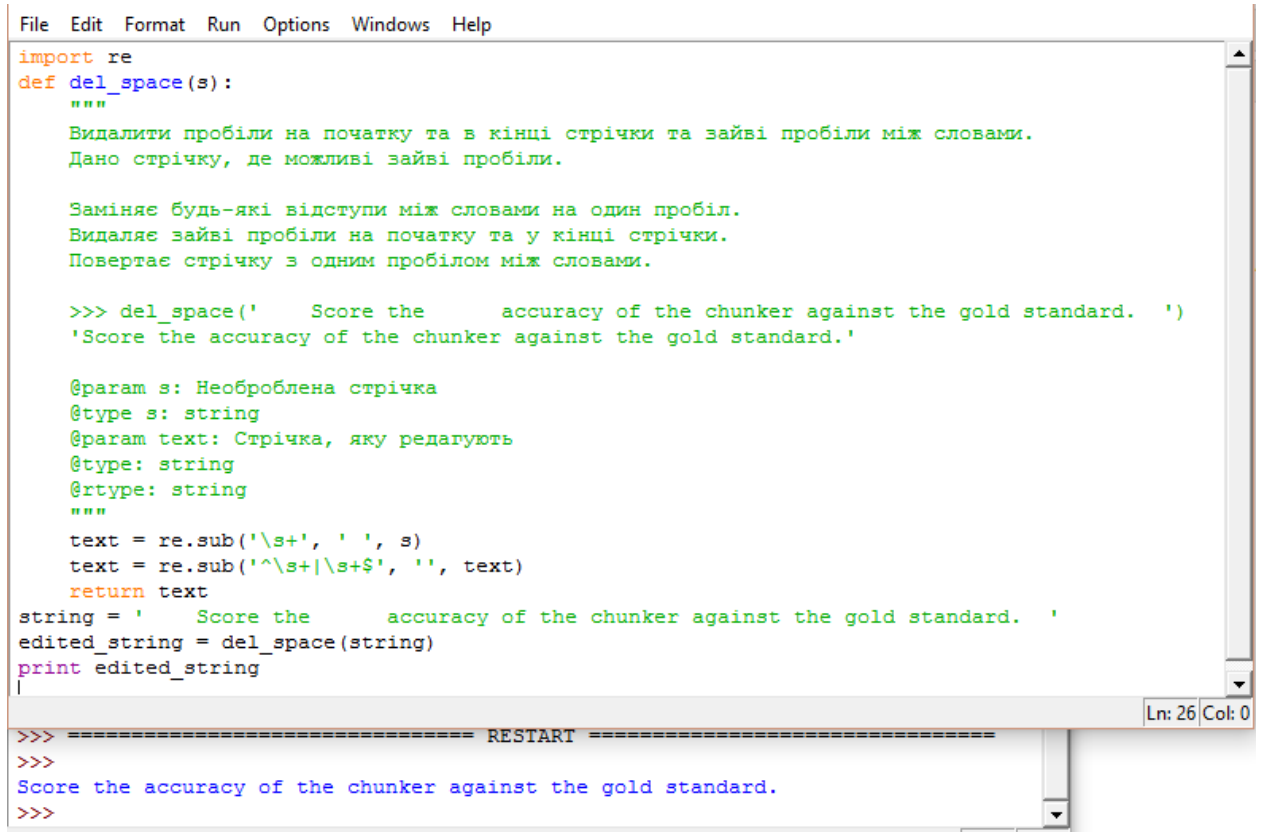
```
def backspace(s):  
    """  
    Видалити пробіли на початку та в кінці стрічки та зайві пробіли між словами.  
    Дано стрічку, де можливі зайві пробіли.  
  
    Видалити всі пробіли і вставити між словами рівно один пробіл.  
    Повертає стрічку з одним пробілом між словами.  
  
    >>> backspace('In the    morning')  
    'In the morning'  
  
    @param s: Стрічка/речення  
    @type s: string  
    @param s: Список, утворений зі слів стрічки/речення  
    @param s: list  
    @param new_s: Оброблена стрічка/речення  
    @type new_s: string  
    @rtype: string  
    """  
    s = s.split()  
    new_s = ' '.join(s)  
    return new_s  
string = 'I am a      student.'  
new_string = backspace(string)  
print new_string
```

Ln: 17 Col: 1

```
>>>  
I am a student.  
>>>
```

Рис. 9. Текст програми №8.

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.



```
File Edit Format Run Options Windows Help
import re
def del_space(s):
    """
    Видалити пробіли на початку та в кінці стрічки та зайві пробіли між словами.
    Дано стрічку, де можливі зайві пробіли.

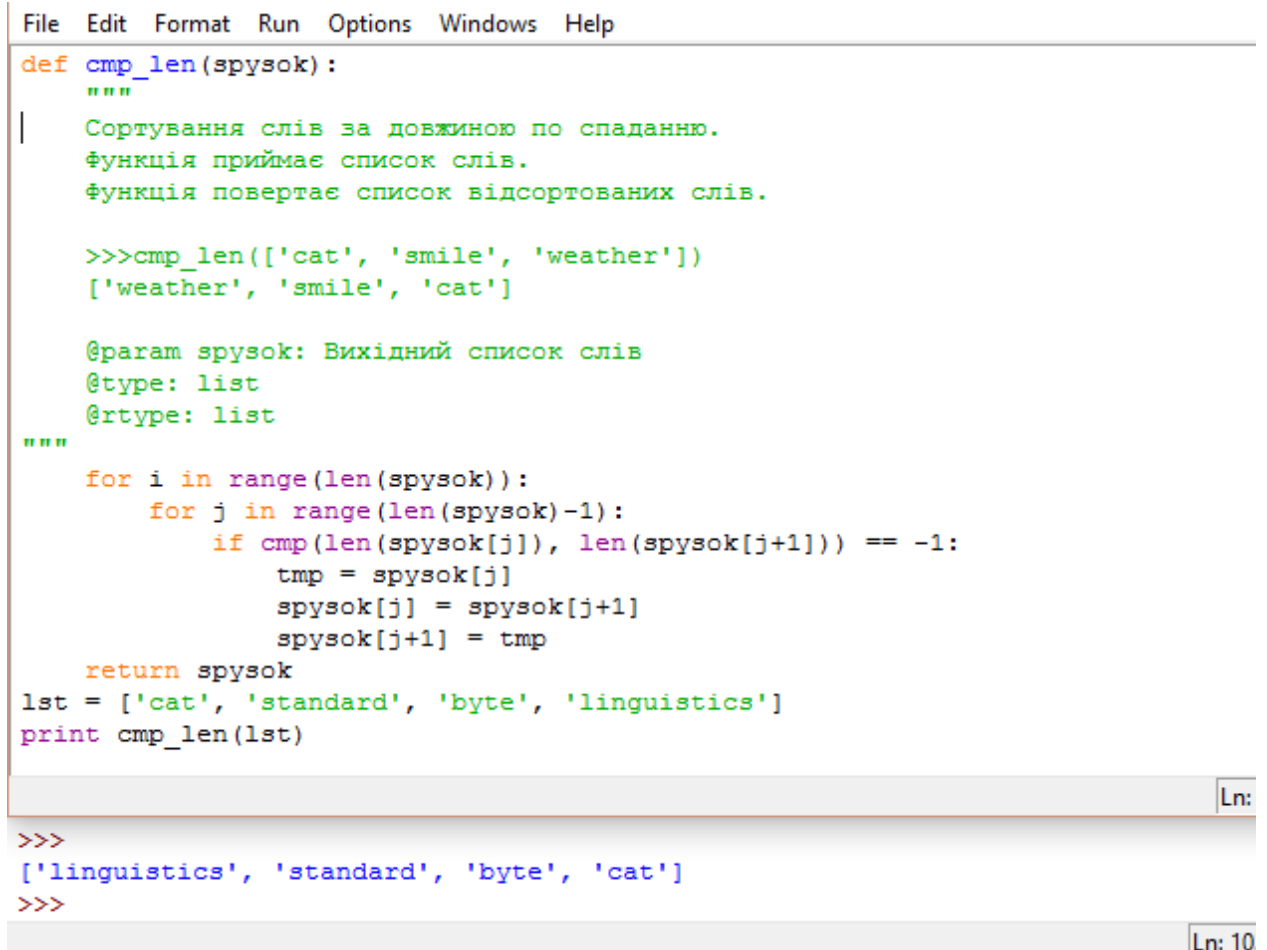
    Заміняє будь-які відступи між словами на один пробіл.
    Видаляє зайві пробіли на початку та у кінці стрічки.
    Повертає стрічку з одним пробілом між словами.

    >>> del_space('    Score the    accuracy of the chunker against the gold standard. ')
    'Score the accuracy of the chunker against the gold standard.'

    @param s: Необроблена стрічка
    @type s: string
    @param text: Стрічка, яку редагують
    @type: string
    @rtype: string
    """
    text = re.sub('\s+', ' ', s)
    text = re.sub('^\s+|\s+$', '', text)
    return text
string = '    Score the    accuracy of the chunker against the gold standard. '
edited_string = del_space(string)
print edited_string
|
Ln: 26 Col: 0
>>> ===== RESTART =====
>>>
Score the accuracy of the chunker against the gold standard.
>>>
```

Рис. 10. Текст програми №9.

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.



```
File Edit Format Run Options Windows Help
def cmp_len(spysok):
    """
    Сортування слів за довжиною по спаданню.
    Функція приймає список слів.
    Функція повертає список відсортованих слів.

    >>>cmp_len(['cat', 'smile', 'weather'])
    ['weather', 'smile', 'cat']

    @param spysok: Вихідний список слів
    @type: list
    @rtype: list
    """
    for i in range(len(spysok)):
        for j in range(len(spysok)-1):
            if cmp(len(spysok[j]), len(spysok[j+1])) == -1:
                tmp = spysok[j]
                spysok[j] = spysok[j+1]
                spysok[j+1] = tmp
    return spysok
lst = ['cat', 'standard', 'byte', 'linguistics']
print cmp_len(lst)
Ln:

>>>
['linguistics', 'standard', 'byte', 'cat']
>>>
Ln: 10
```

Рис. 11. Текст програми №10.

## ВИСНОВОК

У цій лабораторній роботі я вивчила основи структурного програмування мовою Python. Повторила та закріпила знання, отримані при виконанні попередніх лабораторних робіт. Покращила загальні навички у програмуванні.