

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

А. Г. Кривохата, О. В. Кудін

МОВА ПРОГРАМУВАННЯ PYTHON

Методичні рекомендації до лабораторних занять
для здобувачів ступеня вищої освіти бакалавра
спеціальності «Філологія»
спеціалізації «Прикладна лінгвістика»
освітньо-професійної програми «Прикладна лінгвістика»

Затверджено
вченою радою ЗНУ
Протокол № від

Запоріжжя
2026

УДК 004.432(076.5)
К821

Кривохата А. Г., Кудін О. В. Мова програмування Python : методичні рекомендації до лабораторних занять для здобувачів ступеня вищої освіти бакалавра спеціальності «Філологія» спеціалізації «Прикладна лінгвістика» освітньо-професійної програми «Прикладна лінгвістика». Запоріжжя : Запорізький національний університет, 2026. 94 с.

У методичних рекомендаціях викладено теоретичні відомості та зміст лабораторних робіт із курсу «Мова програмування Python». Розглянуто базові типи й структури даних, ключові алгоритмічні конструкції, а також теоретичні засади та практичні алгоритми попереднього опрацювання текстових даних як фундаментального етапу розв'язання задач автоматичного аналізу природної мови.

Основну увагу приділено поєднанню класичних лінгвістичних методів із сучасними інструментами автоматизації на базі мови Python та інтелектуальних мовних моделей. Кожна тема містить теоретичний матеріал із прикладами коду, детальні вказівки щодо виконання завдань та запитання для самоконтролю.

Для здобувачів ступеня вищої освіти бакалавра спеціальності «Прикладна лінгвістика», які навчаються за освітньо-професійною програмою «Прикладна лінгвістика».

Рецензент

О. С. Пишенична, кандидат педагогічних наук, доцент, зав. кафедри комп'ютерних наук

Відповідальний за випуск

А. О. Лісняк, кандидат фізико-математичних наук, доцент, завідувач кафедри програмної інженерії

Зміст

Вступ.....	4
Загальні рекомендації до виконання лабораторних робіт.....	6
Лабораторне заняття № 1. Робота з числами та даними в Python.....	7
Лабораторне заняття № 2. Програми лінійної та розгалуженої структури.....	18
Лабораторне заняття № 3. Програмування циклічних алгоритмів. Обробка текстових та числових послідовностей.....	28
Лабораторне заняття № 4 Програмний аналіз текстових даних: робота з рядковими та списковими структурами.....	38
Лабораторне заняття № 5. Кортежі, множини, словники	56
Лабораторне заняття № 6. Автоматизація обробки корпусів: функції та файли	68
Лабораторне заняття № 7. Попередня обробка тексту	79
Рекомендована література	92
Використана література	93

Вступ

Актуальність вивчення мови програмування Python майбутніми фахівцями з прикладної лінгвістики зумовлена стрімкою цифровізацією гуманітарної сфери. Мова сьогодні розглядається як масив цифрових даних, опрацювання яких вимагає від лінгвіста навичок алгоритмізації та володіння мовами програмування високого рівня. Вміння розробляти власні інструменти для аналізу тексту стає необхідною умовою для успішної професійної діяльності в галузях комп'ютерної лінгвістики, цифрової гуманітаристики та штучного інтелекту.

У навчально-методичному виданні подано стислий теоретичний матеріал та практичні приклади створення програмних проєктів мовою Python, яка є світовим стандартом у сфері аналізу природної мови (NLP). Розглянуто ключові аспекти, необхідні для формування фундаменту програмування:

- побудова лінійних, розгалужених та циклічних алгоритмів для автоматизації рутинних завдань;
- робота зі структурами даних (списками, словниками) для збереження лексичних одиниць та їхніх атрибутів;
- ефективна обробка рядкових даних (текстів);
- використання потужних лінгвістичних бібліотек, як-от NLTK, spaCy та Stanza, для токенізації, лематизації та морфологічного аналізу;
- основи процедурного програмування, взаємодія з файловими системами та візуалізація даних.

Метою курсу «Мова програмування Python» є формування у студентів практичних навичок використання мови Python для автоматизації базових лінгвістичних завдань: від очищення та нормалізації текстових корпусів до розрахунку їхніх статистичних характеристик і візуалізації результатів.

Основними завданнями дисципліни є:

1. Засвоєння базового синтаксису мови Python та принципів роботи з різними типами даних.
2. Опанування алгоритмів обробки текстової інформації (пошук, фільтрація, трансформація).
3. Набуття навичок роботи з файлами, що є основою для автоматизованого аналізу мовних корпусів.
4. Розробка власних прикладних програм для автоматизації рутинних лінгвістичних операцій (підрахунок частотності, виокремлення термінів тощо).

Згідно з вимогами освітньої-професійної програми здобувачі освіти мають досягти таких компетентностей та програмних результатів навчання:

- Здатність учитися й оволодівати сучасними знаннями
- Здатність до пошуку, опрацювання та аналізу інформації з різних джерел
- Здатність вільно оперувати спеціальною термінологією для розв'язання професійних завдань;
- Спроможність аналізувати, оцінювати, перетворювати й створювати тексти різних жанрів та стилів;

- Спроможність до текстової аналітики, зіставлення різнорівневих, різностильових і різномовних фактів;
- Спроможність створювати мовою програмування інструментальні засоби для лінгвістичного аналізу;
- Вміння організовувати процес свого навчання й самоосвіти
- Вміння працювати з інформацією: добирати необхідну інформацію з різних джерел, зокрема з фахової літератури та електронних баз, критично аналізувати й інтерпретувати її, впорядковувати, класифікувати й систематизувати
- Вміння організовувати й проводити прикладні дослідження у галузі філології
- Вміння використовувати інформаційні й комунікаційні технології для вирішення складних спеціалізованих задач і проблем професійної діяльності
- Спроможність використовувати різне програмне забезпечення для проведення лінгвістичних досліджень, аналізу текстової інформації, що пов'язані з лінгвістикою
- Здатність розуміти й використовувати нові досягнення програмного й технічного забезпечення в процесі роботи з лінгвістичним матеріалом

Структура та вміст методичних рекомендацій відповідають змісту силабуса дисципліни «Мова програмування Python».

Навчально-методичне видання містить сім лабораторних робіт, контрольні запитання, посилання на інформаційні джерела, які можна використовувати при розробці програм, переліки рекомендованої та використаної літератури.

Загальні рекомендації до виконання лабораторних робіт

Лабораторні заняття з дисципліни «Мова програмування Python» спрямовані на практичне підтвердження теоретичних положень курсу та набуття навичок програмування. Під керівництвом викладача студент виконує практичні завдання, що дозволяють опанувати алгоритми опрацювання даних на прикладі конкретних проєктів.

Завдання виконуються за індивідуальними варіантами (номер варіанта відповідає порядковому номеру студента у списку академічної групи). Звіт до кожної лабораторної роботи має містити опис етапів виконання та розгорнуті відповіді на контрольні запитання, наведені наприкінці кожного розділу.

Захист результатів відбувається безпосередньо на лабораторному занятті. Під час захисту студент демонструє реалізований програмний код, пояснює логіку виконання завдання та відповідає на запитання викладача. Обов'язковою складовою захисту є презентація результатів за допомогою комп'ютера, а також здатність модифікувати або відтворити окремі елементи коду в режимі реального часу.

За результатами роботи необхідно оформити звіт, який повинен містити:

- титульну сторінку;
- номер варіанта;
- умови завдань, лістинги програм та результати роботи;
- відповіді на контрольні запитання.

Загальний шаблон цього документу розташовано на сторінці курсу «Мова програмування Python» в системі Moodle.

Цей документ завантажується до СЕЗН Moodle у відведені викладачем терміни. Файли для завантаження слід іменувати за шаблоном: Прізвище_Ім'я_IrN (де N – номер лабораторної роботи). Звіти приймаються у форматі текстового документа з розширенням .doc, .docx, .pdf або .odt. Якщо розмір файлів перевищує 5Мб, то відповіді розміщуються на зовнішніх ресурсах, а до СЕЗН додається відповідне посилання.

Лабораторне заняття № 1. Робота з числами та даними в Python.

Мета: опанувати базові типи даних Python, набути навичок обробки числової інформації та реалізовувати введення-виведення даних.

План

1. Введення та виведення інформації.
2. Застосування розділювачів у функції `print()`
3. Форматоване виведення
4. Базові типи даних Python.

1.1 Введення та виведення інформації

Розглянемо приклад введення інформації з клавіатури

```
1 s=input()
Hello, world!

1 s
'Hello, world!'

1 type(s)
str
```

Рисунок 1.1 – Введення та виведення інформації

Наведемо декілька коментарів до даного скрипта (див. рис. 1.1):

1. Викликаємо функцію `input()` та результат її роботи привласнюємо змінній `s`.
2. Користувач вводить значення з клавіатури (в даному прикладі це фраза «Hello, world!») і натиснув Enter, тобто закінчив введення. Значення змінної вивели на екран.
3. Викликали функцію `type()`, за допомогою якої можна визначити тип об'єкта. В даному прикладі це тип `str`, тобто рядок.

Увага!!! Функція `input()` за замовчуванням повертає рядковий об'єкт!!!

Для введення числа необхідно здійснити перетворення типів (на рисунку 1.2 реалізовано перетворення до цілого типу, для чого використано функцію `int()`):

```
[4] 1 s = int(input("Введіть число: "))  
  
Введіть число: 55  
  
[5] 1 s  
  
55  
  
▶ 1 s+5  
  
60
```

Рисунок 1.2 – Введення цілого числа з клавіатури

Для виведення інформації на екран у Python використовується інструкція `print()`. В дужках вказується те, що потрібно надрукувати.

Приклад 1.1

```
print('Привіт')  
або  
print("Привіт")
```

Можна також вказувати в дужках число, ім'я змінної величини або арифметичний/логічний вираз. Для перетворення всіх букв уведеної послідовності у прописні використовують метод `upper()`. Дії для застосування методу такі:

`<ім'я об'єкта>.<ім'я методу>`

Рисунок 1.3 – Формат застосування методу

Наприклад, `famil.upper()`. Для виведення на екран застосуємо конструкцію `print(famil.upper())`.

```
[16] 1 famil='Пайтон'  
      2 famil.upper()  
  
'ПАЙТОН'  
  
▶ 1 print(famil.upper())  
  
📄 ПАЙТОН
```

Рисунок 1.4 – Застосування методу `upper()` до змінної `famil`

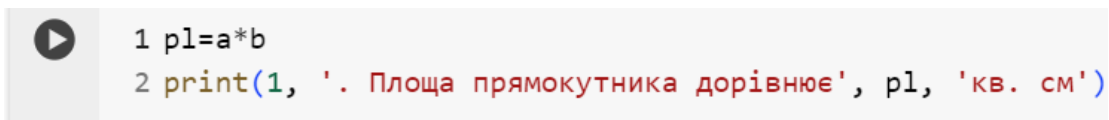
Результат виконання програми (див. рис. 1.4) для кожного з варіантів запису інструкції наведено в таблиці 1.1.

Таблиця 1.1. Приклади реалізації інструкції `print()`

У дужках	Приклад	На екрані	Приклад
Текст	<code>print('Привіт')</code>	Текст без лапок, включно з можливими пробілами	Привіт
Число	<code>print(-2)</code>	Відповідне число	-2
Ім'я змінної величини	<code>x1=1024</code> <code>print(x1)</code>	Значення величини	1024
Вираз	<code>a=10</code> <code>b=12</code> <code>print(a*b)</code>	Значення виразу	120
Метод	<code>print(famil.upper())</code>	Результат застосування методу	ПАЙТОН

1.2 Застосування розділювачів у функції `print()`

В разі потреби надрукувати декілька значень, зокрема і різних типів, вказують всі ідентифікатори через кому. Наприклад, у задачі визначення площі і периметра прямокутника може бути використано такий варіант інструкції:



```

1 p1=a*b
2 print(1, '. Площа прямокутника дорівнює', p1, 'кв. см')

```

Рисунок 1.5 – Обчислення та виведення на екран площі прямокутника

В даному прикладі площу трикутника обчислено за введеними раніше даними (див. таблиця 1.1). Надписи вбрано в лапки, а оскільки значення площі зберігається в змінній `p1` (в рядку 1 наведено відповідну формулу), то на екран виведено її значення. Тут указано число, два тексти (в лапках) та ім'я змінної. В результаті на екран буде виведено:

Площа прямокутника дорівнює 120 кв. см

Між вказаними в інструкції `print()` значеннями виводиться також один пробіл. Цей роздільний знак можна змінити на будь-який інший символ (послідовність символів). Це можна зробити так:

```
print(<список виводу>, sep = ' , ')
```

Результат виконання інструкції `print()` з різними типами розділювачів наведено в таблиці 1.2.

Таблиця 1.2 Приклади застосування розділювачів

Приклад	На екрані
<pre>a=10 b=20 c=a+b print(a, '+', b, '=', c, sep = ' ') #a i b - задані числові значення #c - їх сума</pre>	10 + 20 = 30
<pre>print("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun", sep="-")</pre>	Mon-Tue-Wed- Thu-Fri-Sat-Sun
<pre>print(1, 2, 3, sep='//')</pre>	1//2//3

На рисунку 1.6 наведено приклад об'єднання двох рядків `s1` та `s2`. Для цього використано знак «+». Результат об'єднання збережено в змінну `s3`.

```
1 s1='Hello'
2 s2='world'
3 s3=s1+s2
4 print(s3)
```

Hello world

Рисунок 1.6 – Об'єднання рядків

Виконайте самостійно. Модифікуйте код на рисунку 1.6: додайте необхідні знаки пунктуації та виведіть на екран значення змінної `s3`.

За допомогою параметру `end` можна вказати, що робити після виведення рядка. За замовчуванням відбувається перехід на новий рядок. Однак цю дію можна скасувати, якщо вказати будь-який інший символ або рядок:

```
print(<список виводу>, end = ' ')
```

або вказати обидва параметри

```
print(<список виводу >, sep = ', ', end = ' ')
```

або

```
print(<список виводу >, end = ' ', sep = ', ')
```

```
1 print(10, end="_")
```

10_

Рисунок 1.7 – Приклад застосування параметру `end`

Перехід на новий рядок позначається комбінацією символів `"\n"`. Якщо привласнити це значення параметру `end`, то жодних змін в роботі інструкції

`print()` не буде помітно, оскільки за замовчуванням і так відбувається переведення на новий рядок.

```
print(10, end='\n')
```

Якщо ж потрібно відступити на один додатковий рядок після виведення, то можна зробити так:

```
1 print(10, end='\n\n')
```

10

Рисунок 1.8 – Робота з параметром `end`

Інструкція `print()` без списку виведення видає порожній рядок.

1.3 Форматоване виведення

В останніх версіях Python для форматованого виведення даних використовується метод `format()`. Цей метод форматує значення-аргумент, вказаний в круглих дужках, за шаблоном (зразком), який вказаний в лапках і фігурних дужках

```
print('{<шаблон форматування>}'.format(<значення>))
```

```
1 print('{:10.2f}'.format(-4/3))
2 a = -4/3
3 print('{:10.2f}'.format(a))
4 z = 123
5 print('{:5d}'.format(z))
```

-1.33

-1.33

123

Рисунок 1.9 – Приклади роботи з методом `format`

Особливості оформлення шаблону виведення:

- 1) між першими (лівими) символами '{ пробілу бути не повинно. Інакше він також буде виведений;
- 2) після двокрапки пробіл може бути відсутнім; два і більше пробілів записувати не можна.

1.4 Базові типи даних Python

У Python кожна змінна – це об'єкт, який має свій тип. Оскільки Python є мовою з **динамічною типізацією**, розробнику не потрібно явно вказувати тип

при створенні змінної – інтерпретатор визначає його автоматично під час присвоєння значення. Далі в таблиці 1.3 наведено коротку інформацію про основні типи даних.

Таблиця 1.3 Класифікація типів даних Python

Категорія	Тип	Опис	Приклад
Числові	int	Цілі числа будь-якої довжини	42, -7
	float	Числа з плаваючою крапкою (десяткові дробы)	3.14, 2.0
	complex	Комплексні числа	3 + 5j
Логічні	bool	Логічні значення (Істина/Хибність)	True, False
Текстові	str	Рядки символів (Unicode)	"Лінгвістика", 'Python'

Числові типи int (цілі числа) у Python 3 мають «необмежену» точність (обмеження стосується лише обсягів оперативної пам'яті), float – використовується для представлення дійсних чисел. Логічний тип bool має лише два значення True (істина) та False (хибність), в свою чергу True еквівалентно 1, а False – 0. Рядки в Python – це незмінні послідовності символів Unicode. Вони можуть записуватися як в одинарних так і в подвійних лапках (див. таб. 1.3). Нижче наведено декілька прикладів з базовими типами даних та використання вбудованої функції type().

```
a = 5 # int
a = 5 # int
b = 7.0 # float
c = 2 > 4 # boolean
d = "World" # string
e = 1.5 + 0.5j # complex
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
print(a, b, c, d, e.real, e.imag)
```

Рисунок 1.10 – Базові типи даних. Неявне вказування

```

a = 5 # int
b = 7.0 # float
c = 1 + 2 # 3
d = 5 - 3 # 2
e = a * b # 35.0
f = 3.0 / 2 # 1.5
g = 3 / 2 # 1
h = 5 % 3 # 2
j = 10 ** 7.3 # 19952623.1497
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
print(type(f))
print(type(g))
print(type(h))
print(type(j))
print(a, b, c, d, e, f, g, h, j)

```

Рисунок 1.11 – Основні арифметичні операції

Для роботи із вбудованими математичними функціями необхідне підключення бібліотеки `math`

```

from math import *
a = 1
b = 2
x = sqrt(a*b)/(exp(a)*b)+a*exp((2*a)/b)
print(x)

```

Рисунок 1.12 – Вбудовані математичні функції

При виконанні операції ділення двох дійсних чисел потрібно обмежувати бажану кількість цифр у дробовій частині. Також можна надрукувати числа в науковому форматі:

```

print('% .3f' % <значення>)
print('% .2f' % <значення>)
print('% .2e' % <значення>)

```

Таблиця 1.4. Форматоване виведення дробних чисел

Вивід цілого числа b=5 print(b) print('% 5d' % b)	5 <table><tr><td></td><td></td><td></td><td></td><td>5</td></tr></table>					5
				5		
Вивід від'ємного числа a = -234.123 print(a) print(-a) print('% .2f' % a)	-234.123 234.123 -234.12					
Вказування загальної довжини числа						

<code>b = 30000/7</code>	
<code>print(b)</code>	4285.714285714285
<code>print('%7.3f'% b)</code>	4285.714
<code>print('%7.3e'% b)</code>	4.286e+02

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №1

В середовищі програмування Google Colab опрацюйте наступні завдання:

1. Перейдіть за посиланням <https://colab.research.google.com/>
2. У меню, що з'явиться натисніть Створити блокнот
3. Змініть назву блокнота відповідно до шаблону Прізвище_номер лабораторної_шифр групи.
4. Виконайте чотири завдання відповідно до свого варіанта:
 - Для кожного завдання створюйте окрему кодову комірку.
 - Формулювання завдань наводити в текстових комітках.

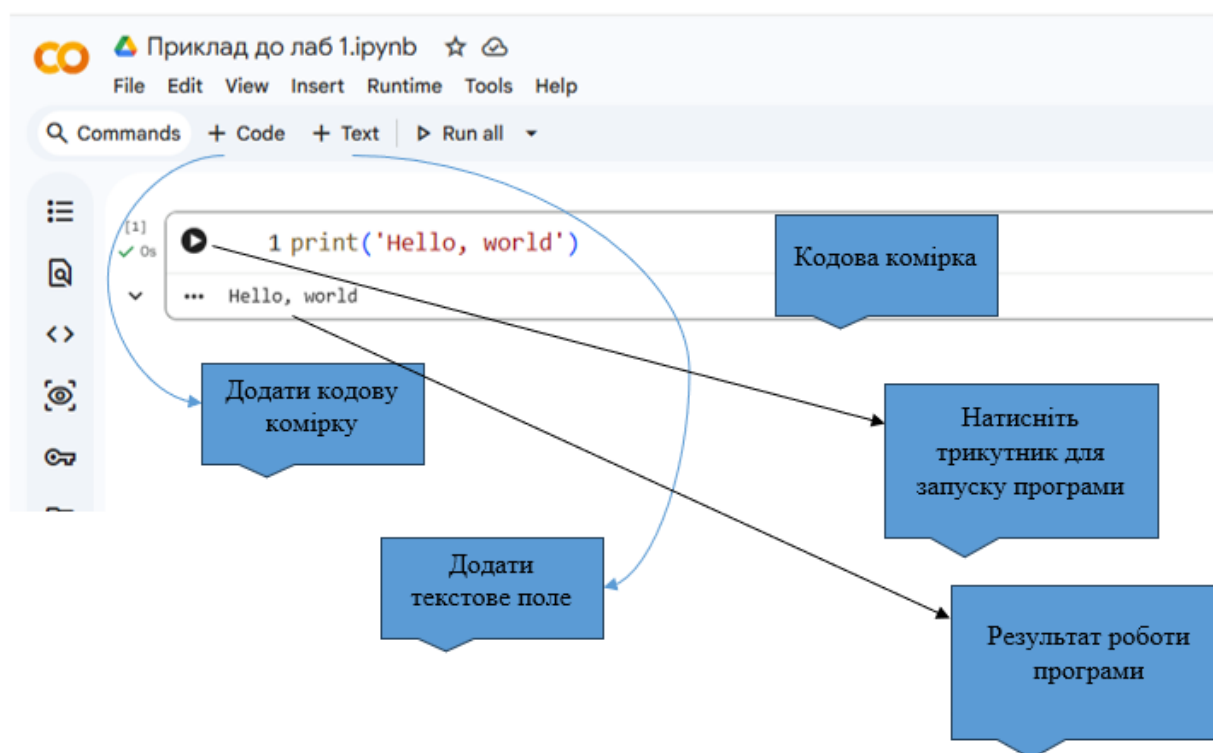


Рисунок 1.13 – Інтерфейс середовища Google Collab

- Відповіді на контрольні запитання оформіть наприкінці блокноту в текстовому полі.
- Налаштуйте доступ для коментування.
- За результатами роботи оформіть текстовий звіт (шаблон міститься на сторінці курсу та додайте до цього документа посилання на ваш блокнот).
- Захистіть роботу.

Варіанти завдань до лабораторної роботи №1

Завдання 1. Створіть в інтерактивному режимі (без додаткових повідомлень) три змінні: першу – для вашого імені, другу – для вашого прізвища, третю – для шифра групи. За допомогою однієї інструкції `print` виведіть їх на екран в один рядок.

Завдання 2. Напишіть програму, яка зчитує та відображає основні дані про студента: ім'я, прізвище та шифр групи. Інформацію виводити одним рядком або блоком тексту за допомогою функції `print()`.

Завдання 3. Напишіть скрипт, що зчитує ім'я в текстову змінну та виводить на екран рядок привітання у форматі: `Hello, Harry!`.

Завдання 4. Задано число 156.12459835. Вивести на екран:

Варіанти 1; 5; 9; 13; 17 Число з 3 цифрами у дробовій частині

Варіанти 2; 6; 10; 14; 18 Число з 2 цифрами у дробовій частині

Варіанти 3; 7; 11; 15 Число в «науковому» форматі

Варіанти 4; 8; 12; 16 Число з 2 цифрами у дробовій частині з протилежним знаком

Завдання 5. Отримати на екрані з використанням методу `format()` таке:

Варіанти 1; 5; 9; 13 Варіанти 2; 6; 10; 14

○○○○○124○○○○○13

○○○○○56○○○○○355

○○○○○587○○○○○○8

○○○○○1.24○○○○13.52

○○○○3.567○○○-355.1

○○○○○8.2○○○○○9.18

○○○○88 ○○-769.800

Варіанти 3; 7; 11; 15 Варіанти 4; 8; 12; 16

○○○○○7.240○○○-43.520

○○○○23.500○○○○55.107

○○○○88.203○○-769.800

○○○○○7○○○-43.520

○○○23○○○○55.107

○○○○88 ○○-769.800

Де ○ – порожня позиція на екрані

Контрольні запитання

1. У чому полягає різниця між типами `int` та `float` у Python? Який із цих типів ви б обрали для зберігання загальної кількості слів у корпусі, а який – для розрахунку середньої довжини речення? Поясніть свій вибір.

2. Як працюють оператори `//` (цілочисельне ділення) та `%` (остача від ділення)?

3. Які функції слід використати, щоб перетворити рядок «125» у число для подальших розрахунків, і що станеться, якщо рядок міститиме нецифрові символи?

4. Які можливості надає модуль `math` для статистичного аналізу текстів?

5. Як за допомогою логічних операторів (`>`, `<`, `==`, `and`, `or`) створити умову для фільтрації слів у частотному списку?

Корисні посилання:

1. Початкові відомості з Colab
(https://colab.research.google.com/notebooks/welcome.ipynb?hl=uk#scrollTo=GJBs_flRovLc)
2. Стил ь коду Python. URL : <https://peps.python.org/pep-0008/>
3. Підключення модуля pep8 URL : <https://github.com/nbper8/nbper8>
4. Філософія Python <https://peps.python.org/pep-0020/>
5. Python Tutorial. URL : <https://www.w3schools.com/python/default.asp>
6. Путівник мовою програмування Python. URL :
https://pythonguide.rozh2sch.org.ua/#_%D0%B2%D1%81%D1%82%D1%83%D0%BF
7. Метод format (перелік параметрів та ключів). URL :
https://www.w3schools.com/python/ref_string_format.asp

Лабораторне заняття № 2. Програми лінійної та розгалуженої структури

Мета: набути практичних навичок програмування лінійних та розгалужених алгоритмів в Python.

План

1. Програми лінійної структури
2. Оператор розгалуження `if`.
3. Пошук підрядка та нормалізація тексту
4. Перевірка закінчення рядка: метод `.endswith()`

2.1 Програми лінійної структури

Розглянемо програму лінійної структури в Python. Перш за все підключимо модуль `pep8` для контролю дотримання рекомендацій зі стилю кодування. Для цього в кодовому полі необхідно ввести:

```
!pip install pycodestyle
!pip install --index-url https://test.pypi.org/simple/ nbpep8
from nbpep8.nbpep8 import pep8
```

Рисунок 2.1 – Підключення модуля `pep8`

Розглянемо просту програму для обчислення значення математичного виразу для певного значення аргумента (вводить користувач з клавіатури). Виведення результату на екран здійснюватимемо у форматованому вигляді – у вигляді звичайного десяткового дробу та в науковому форматі. Отже, необхідно обчислити значення y :

$$y = \frac{\sin^3(x) - \cos(x-1)^2}{\arctg(x+1) + \sqrt[3]{\lg(x)}}$$

Перед розробкою програми розглянемо структуру виразу. Оскільки він містить спеціальні функції, необхідно підключити модуль `math` на початку програми. Загалом, один з варіантів програми виглядатиме так:

```

1 import math
2
3 xValue = float(input())
4 yValue = ((math.sin(xValue))**3 - math.cos((xValue - 1)**2)) / \
5 (math.atan(xValue + 1) + (math.log(xValue, 10))**1/3)
6
7 print("{:.3f}".format(yValue))
8 print("{:12.4e}".format(yValue))
9
10 print("%.3f" % (yValue))
11 print("%12.4e" % (yValue))
12
13 print(f"{yValue:.3f}")
14 print(f"Результат обчислення {yValue:12.4e}")
15 pep8(_ih)
16

```

Рисунок 2.2 – Обчислення значення виразу

Наведемо декілька коментарів до цієї програми.

1. В рядку №1 імпортується модуль `math` для використання тригонометричних функцій. Блок підключення додаткових модулів відокремлюємо від основної програми порожнім рядком.
2. В рядку №3 значення, яке вводить користувач з клавіатури перетворюється в тип `float`, та присвоюється змінній `xValue`.
3. Рядки 4-5 – це запис виразу алгоритмічною мовою. Проте, оскільки вираз занадто довгий, то в кінці рядка 4 використано символ «\» для перенесення другої частини виразу (тобто знаменника) на рядок 5. Для того, щоб програма працювала коректно і без помилок, пробіл на початку рядка 5 – обов'язковий. Всі знаки математичних операцій відокремлено пробілами. Знак «*» означає піднесення до степеня.
4. Рядки 7-8 – виведення на екран результату з трьома знаками після коми та в науковому форматі відповідно.
5. Рядки 10-11 – стандартне форматоване виведення на екран
6. Рядки 13-14 – використання f-рядка (f-string).
7. Рядок 15. Виклик модуля для перевірки стиля коду.

F-рядок – це рядок-шаблон, починається з символу `f` (або `F`). Кожного разу, коли необхідно додати в рядок значення будь-якої змінної, то ім'я змінної записують у фігурних дужках. Всередині фігурних дужок можна вказати спосіб форматування, який є аналогічним методу `format()`. У подальшому будемо використовувати один зі способів виведення на екран.

2.2 Оператор розгалуження `if`

Для керування послідовним виконанням команд у Python використовують оператори розгалуження. Конструкція `if-elif-else` надає можливість обирати потік виконання програми залежно від значень змінних у момент перевірки. Загальний синтаксис умовного оператора такий:

```

if <<boolean_expretion1>>:
    <<st1>>
elif
<<boolean_expretion2>>:
    <<st2>>
...
elif
<<boolean_expretionn>>:
    <<stn>>
else:
    <<str>>

```

Рисунок 2.3 – Загальний синтаксис умовного оператора

Вкладеність операторів у Python регулюється відступами (пробілами): логічний блок починається після двокрапки (:) і триває, доки зберігається однаковий рівень пробілів. Стандартний відступ – 4 пробіли. Критично важливо дотримуватися єдиного стилю відступів у межах усієї програми, щоб уникнути помилок.

Python – одна з небагатьох мов, де пробіли є частиною синтаксису, тому помилка `IndentationError` є дуже поширеною серед новачків (див. рис. 2.4-2.5).

```

1 x = 10
2 if x > 5:
3     print("Число більше за 5") # Відступ 4 пробіли
4     if x == 10:
5         print("Це саме десять") # Відступ 8 пробілів (вкладений блок)
6 print("Програма завершена")    # Повернення до краю – блок if закінчився

```

Рисунок 2.4 – Програма з коректними відступами

```

1 x = 10
2 if x > 5:
3 print("Цей рядок викличе помилку") # Помилка! Має бути відступ після ":"

```

Рисунок 2.5 – Програма без відступів після блоку `if`

В середовищі `Colab` відступ від краю рядка можна зробити клавішею `Tab`.

При введенні значення змінної з клавіатури необхідно застосувати функцію `int()` для перетворення введеного користувачем значення до числового формату (див. рис. 2.6).

```

1 a = int(input())
2 if a < -5 :
3     print('Low')
4 elif -5 <= a <= 5 :
5     print('Mid')
6 else :
7     print('High')

```

0
Mid

Рисунок 2.6 – Умовний оператор з користувацьким введенням

Приклад з рисунка 2.6 можна модифікувати для введення певної фрази користувачем, наприклад, певний пароль:

```

1 userPassword = input()
2 if userPassword == 'Python':
3     print('Ok')
4 elif userPassword == 'python' or userPassword == 'PYTHON':
5     print('Incorrect input')
6 else:
7     print('Repeat input')

```

Python
Ok

Рисунок 2.7 – Введення пароля користувачем

Конструкції `elif` та `else` є опціональними. Код у блоці `else` виконується лише в тому разі, якщо всі попередні перевірки повернули `False`. Оскільки в Python немає операторних дужок, межі логічних блоків визначаються виключно за допомогою відступів.

2.3 Пошук підрядка та нормалізація тексту

У прикладній лінгвістиці нормалізація даних (видалення пробілів, приведення до одного регістру) – це перший і найважливіший етап. Без цього жоден алгоритм аналізу тексту не працюватиме коректно, оскільки комп'ютер бачить «Слово» і «слово» як два абсолютно різні об'єкти.

Для перевірки наявності конкретного слова або символу в рядку використовується оператор входження `in`.

! **Важливо.** Перш ніж виконувати пошук, необхідно привести текст до єдиного регістру (зазвичай нижнього). Це дозволяє уникнути помилок, коли програма не розпізнає однакові слова, написані з великої чи малої літери (наприклад, «Лінгвістика» та «лінгвістика»). Звісно, це правило не розповсюджується на завдання, де регістр є принциповим (наприклад, пошук власних назв).

На рисунку 2.8 наведено фрагмент програми, яка запитує у користувача введення певного тексту. Основними етапами реалізації є: зчитування даних, нормалізація тексту за допомогою методу `.lower()` введена послідовність символів переводиться у нижній регістр, а далі через оператор `in` здійснюється перевірка наявності ключового слова (наприклад, «лінгвістика») у підготовленому тексті.

```
1 text = input("Введіть текст: ").lower()
2 if "лінгвістика" in text:
3     print("Скоріш за все текст відноситься до прикладної лінгвістики")
```

*** Введіть текст: Прикладна лінгвістика використовує програмування для розробки мовних програмних продуктів
Скоріш за все текст відноситься до прикладної лінгвістики

Рисунок 2.8 – Перевірка наявності «лінгвістика» у введеному тексті

Загалом обробка тексту завжди починається з його «очищення» або нормалізації. Окрім приведення до нижнього регістру, у Python часто використовуються методи для перетворення тексту у верхній регістр та видалення «шумових» символів (пробілів, знаків табуляції тощо). Метод `.upper()` перетворює всі літери рядка на великі. Це корисно для створення заголовків або акцентування уваги на певних лексемах.

```
1 org_name = input("Введіть назву університету (наприклад, зну): ").upper()
2
3 if org_name == "ЗНУ":
4     print("Ви вказали: Запорізький національний університет.")
```

Рисунок 2.9 – Застосування метода `.upper()`

Метод `.strip()` видаляє всі пробіли, символи табуляції (`\t`) та перенесення рядка (`\n`) з початку та кінця рядка. Це критично важливо при обробці даних, введених користувачем вручну, або при зчитуванні тексту з файлів, де часто з'являються випадкові пробіли.

```
1 # введемо "      лінгвістика      " із зайвими пробілами
2 word = input("Введіть слово для аналізу: ").strip()
3
4 print(f"Довжина очищеного слова '{word}': {len(word)} символів.")
5 # без .strip() довжина була б значно більшою через пробіли
```

Введіть слово для аналізу: лінгвістика
Довжина очищеного слова 'лінгвістика': 11 символів.

Рисунок 2.10 – Очищення тексту від зайвих пробілів

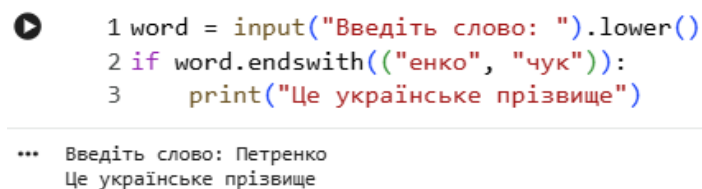
Зазвичай методи можна комбінувати («ланцюжком»), наприклад: `text.strip().lower()`. Це спочатку прибере зайві пробіли по краях, а потім зробить усі літери маленькими. Така комбінація є необхідним етапом попередньої обробки тексту.

2.4 Перевірка закінчення рядка: метод `.endswith()`

Метод `.endswith()` використовується для перевірки того, чи закінчується рядок певним символом або послідовністю символів (підрядком). Результатом є логічне значення `True`, якщо рядок закінчується вказаним суфіксом, і `False` – у протилежному випадку. Метод чутливий до регістру (наприклад, «`.txt`» та «`.TXT`» – це різні закінчення).

Цей метод зручно використовувати для автоматичного пошуку слів із певними флексіями (закінченнями) або для перевірки розділових знаків у кінці речень.

На рисунку 2.11 наведено фрагмент коду з використанням методу `.endswith("енко", "чук")`, який шукає рядок, що закінчується на символи: е, н, к, о та ч, у, к.



```
1 word = input("Введіть слово: ").lower()
2 if word.endswith(("енко", "чук")):
3     print("Це українське прізвище")
```

*** Введіть слово: Петренко
Це українське прізвище

Рисунок 2.11 – Перевірка закінчення слова

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №2

В середовищі програмування Google Colab виконайте три завдання відповідно варіанта.

- Для кожного завдання створюйте окрему кодову комірку. Формулювання завдань наводити в текстових комірках.

- Відповіді на контрольні запитання оформіть наприкінці блокноту в текстовому полі.

- Налаштуйте доступ для коментування.

- За результатами роботи оформіть текстовий звіт (шаблон міститься на сторінці курсу та додайте до цього документа посилання на ваш блокнот).

- Захистіть роботу.

Варіанти завдань до лабораторної роботи №2

Код має узгоджуватись зі стандартом pep8. Виконайте три завдання відповідно свого варіанта.

Завдання 1. Обчисліть та надрукуйте значення виразу. Значення x вводить користувач з клавіатури.

№Вар.	Завдання	№Вар.	Завдання
1	$y = \arctg(x) + \frac{e^{0,6x-1} - \sqrt{(x+6,1)^3}}{\ln(x) + tg^2(x)}$	9	$y = \sin\left(\frac{x + 2,3\lg(x+1)}{\sqrt{2\ln(x)} + \cos(x)}\right)$
2	$y = \arcsin\left(\frac{x \cdot \ln(x)}{1 + \cos(x)} + 1\right)$	10	$y = \frac{e^{-x} + tg(x-1)}{\ln x + \sqrt{\cos x+1 }}$
3	$y = \frac{\sin(x^2) - \cos^4(x-1)^2}{\arctg(x+2,6) + \sqrt[3]{\ln(x)}}$	11	$y = tg\left(\frac{x + 3 \cdot \lg(x+1)}{\sqrt{\ln(4x)} + \cos(x)}\right)$
4	$y = 0,3 \cdot \lg(e^{-x}) + \frac{\arctg(x) - \sin^2(x)}{4 \cdot \sqrt{\ln x-1 }}$	12	$y = \frac{\sin(x^3) - \cos(x-1)^2}{\arctg(x+1) + \sqrt[3]{\lg(x)}}$
5	$y = \frac{e^{-x} - 4 \cdot \lg(x)}{\ln(x) - \cos x+1 }$	13	$y = \frac{\sqrt{\sin(x+1)\arctg(\pi x)}}{\lg(x+2\pi)} + 1$
6	$y = \arccos\left(\frac{x - \lg(x)}{1 + \cos(3x)} + 1\right)$	14	$y = \arcsin^2(x-1) + \left(\frac{2\sqrt{x}}{1 + \cos(x)}\right)$
7	$y = \frac{e^{-3x} + 4 \cdot tg(4x-1)}{\cos x + \sqrt{\cos(2x)}}$	15	$y = \frac{e^{-x} - x \cdot \sin(x) - 4x}{1 + \sqrt{\cos(2x)}}$
8	$y = \frac{e^{-3x} + \ln^3(x-1)}{\ln x+1 + tg(x^2-1)}$	16	$y = \arcsin(x) + \frac{e^{x-1} - \sqrt{(x+1)^5}}{\lg^3(x) + tg(x)}$

Завдання 2. Розробіть програму відповідно до варіанта.

1. Напишіть програму, що просить користувача ввести ім'я з клавіатури і виводить іменне привітання, повторюючи введене ім'я.

2. Напишіть програму, що просить користувача ввести з клавіатури назву футбольного клубу і повторює його на екрані зі словами «...-чемпіон!».

3. Напишіть програму, що просить користувача ввести ім'я та прізвище з клавіатури і виводить їх у одному рядку.

4. Напишіть програму, що просить користувача ввести з клавіатури назву держави і столиці держави і виводить на екран рядок «Столиця держави ... - місто ...». Замість «...» виводяться введені користувачем рядки.

5. Напишіть програму, що просить користувача ввести назву роману та прізвище автора з клавіатури і виводить на екран рядок «Письменник ... - автор роману ...». Замість «...» виводяться введені користувачем рядки.

6. Напишіть програму, що просить користувача ввести назву товару та його ціну з клавіатури і виводить на екран рядок «Ціна товару ... становить ...». Замість «...» виводяться введені користувачем рядки.

7. Напишіть програму, що просить користувача ввести два імені з клавіатури і виводить на екран рядок «... + ... = Кохання». Замість «...» виводяться введені користувачем рядки.

8. Напишіть програму, що просить користувача ввести ім'я та назву міста з клавіатури і виводить на екран рядок «... мешкає в місті ...». Замість «...» виводяться введені користувачем рядки.

9. Напишіть програму, що просить користувача ввести ім'я з клавіатури і виводить на екран рядок «Шановний ..., вітаємо з успішним виконанням першого завдання лабораторної роботи!». Замість «...» виводиться введений користувачем рядок.

10. Напишіть програму, що просить користувача ввести ім'я з клавіатури і виводить на екран рядок «Шановний ..., ви зараз почали читати рядок, читання якого вже закінчуєте.» Замість «...» виводиться введений користувачем рядок.

11. Напишіть програму, що просить користувача ввести назву пісні та прізвище співака з клавіатури і виводить на екран рядок «Пісню ... виконує співак» Замість «...» виводяться введені користувачем рядки.

12. Напишіть програму, що просить користувача ввести шифр групи та прізвище студента з клавіатури і виводить на екран рядок «Студент ... навчається в групі» Замість «...» виводяться введені користувачем рядки.

13. Напишіть програму, що просить користувача ввести назву корабля та порт приписки з клавіатури і виводить на екран рядок «Корабель ... приписаний до порту» Замість «...» виводяться введені користувачем рядки.

14. Напишіть програму, що просить користувача ввести ім'я з клавіатури і виводить на екран рядок «Тут був» Замість «...» виводиться введений користувачем рядок.

15. Напишіть програму, що просить користувача ввести прізвище та назву професії з клавіатури і виводить на екран рядок «... працює за професією» Замість «...» виводяться введені користувачем рядки.

Завдання 3. Напишіть програму відповідно до варіанта, скористайтесь прикладами наведеними на рисунках 2.8-2.11.

1. Напишіть програму, яка визначає тип введеного користувачем речення. Якщо речення закінчується на «?» – виведіть «Це питальне речення», якщо на «!» – «Це окличне речення», якщо «.» – «Це розповідне речення».

2. Напишіть програму для аналізу введеного користувачем речення. Якщо в реченні є слово «мова» (використовуйте оператор `in` та метод `.lower()`),

виведіть: «Текст стосується лінгвістики». Якщо в реченні є слово «комп'ютер» або «програма», виведіть: «Текст стосується інформаційних технологій». Якщо немає жодного з цих слів, виведіть: «Тематика тексту не визначена».

3. Напишіть програму, яка запитує у користувача слово. Якщо довжина слова (функція `len()`) більша за 10 символів, виведіть: «Це довге слово». Якщо від 5 до 10 символів – «Це слово середньої довжини». Якщо менше 5 – «Це коротке слово».

4. Напишіть програму, яка перевіряє, чи є введене слово дієсловом у формі інфінітива (спрощено). Якщо слово закінчується на «ти» або «тися» (використовуйте `.endswith()`), виведіть: «Ймовірно, це дієслово». В іншому випадку виведіть: «Частина мови не визначено»

5. Користувач вводить слово. Програма має перевірити, чи починається воно з великої літери (метод `.isupper()` для першого символу або метод `.istitle()`). Якщо так – виведіть: «Це може бути власна назва (ім'я, місто тощо)». Якщо ні – «Це загальна назва».

6. Напишіть програму, яка аналізує введене речення на наявність сполучників «і», «та», «але». Якщо в реченні є хоча б один із них, виведіть: «У реченні вжито сполучники для зв'язку слів». Якщо немає – «Сполучників не виявлено».

7. Програма запитує користувача: «Як справи?». Якщо відповідь містить слово «good» або «fine», виведіть: «Ви відповіли англійською». Якщо «добре» або «чудово» – «Ви відповіли українською». В іншому випадку – «Мову відповіді не розпізнано».

8. Напишіть програму, яка аналізує введене користувачем слово. Якщо слово закінчується на «ість» (наприклад, *радість*, *сміливість*), виведіть: «Це абстрактний іменник, що означає якість або стан». Якщо слово закінчується на «ізм» або «їзм» (*гуманізм*, *архаїзм*), виведіть: «Це іменник, що означає ідеологію, напрям або систему». В іншому випадку виведіть: «Слово не належить до обраних категорій».

9. Напишіть програму для аналізу офіційного листа. Якщо речення починається зі слова «Шановний» або «Шановна», виведіть: «Це офіційне звернення». Якщо речення починається з «Привіт» – «Це неформальне спілкування».

10. Напишіть програму, яка перевіряє наявність заперечної частки «не» у реченні. Важливо: перетворіть текст у нижній регістр, щоб знайти «не» на початку речення. Якщо частка є – виведіть: «Речення містить заперечення». Якщо немає – «Речення стверджувальне».

11. Напишіть програму, яка перевіряє, чи закінчується введене слово на «-енко» або «-чук». Якщо так, виведіть: «Ймовірно, це українське прізвище». Якщо слово закінчується на «-ський» або «-цька» – виведіть: «Це прізвище або прикметник». В іншому випадку – «Морфологічний тип не визначений».

12. Користувач вводить слово. Програма має перевірити, чи складається воно повністю з великих літер (метод `.isupper()`). Якщо так, виведіть: «Слово може бути аббревіатурою (наприклад, ООН, НАНУ)». Якщо ні – виведіть: «Це звичайне слово або власна назва».

13. Напишіть програму, яка шукає в реченні слова, що виражають невпевненість: «мабуть», «можливо», «здається». Якщо в реченні є хоча б одне з них, виведіть: «Текст має модальність невпевненості». Якщо знайдено слово «безумовно» або «точно» – «Текст виражає впевненість». Якщо слів не знайдено – «Модальних маркерів не виявлено».

14. Напишіть програму, яка запитує іменник у називному відмінку. Якщо слово закінчується на «-а» або «-я» (крім винятків), виведіть: «Ймовірно, це іменник жіночого роду». Якщо на «-о» або «-е» – «Ймовірно, це іменник середнього роду». В іншому випадку – «Ймовірно, це іменник чоловічого роду (або слово з нульовим закінченням)».

15. Напишіть програму, яка перевіряє, чи містить речення поширені прийменники «у», «в», «на», «під», «через». Якщо знайдено хоча б один (використовуйте `in` та перевірку з пробілами, наприклад « в »), виведіть: «У реченні вжито службові частини мови для вказівки на місце або обставину». Якщо ні – «Прийменників не знайдено».

Контрольні запитання

1. Яку лінгвістичну задачу можна розв'язати виключно за допомогою послідовних операцій без розгалужень? Наведіть приклад.

2. Як використовуються логічні оператори `and`, `or`, `not` у розгалужених структурах?

3. Що таке вкладені оператори умови? Які існують правила відступів у Python для таких структур і як неправильний відступ може змінити логіку обробки текстових даних?

4. Як за допомогою оператора `in` перевірити, чи містить речення певний розділовий знак або чи є символ голосною літерою?

5. Як порядок виконання арифметичних та логічних операцій впливає на результат у лінійних програмах?

6. Чому важливо використовувати дужки при розрахунку статистичних показників тексту (наприклад, середньої кількості складів у слові), якщо формула містить кілька дій?

Корисні посилання

1. Математичні функції. URL : <https://docs.python.org/uk/3/library/math.html>

2. Ввід та вивід даних у Python. URL : <https://acode.com.ua/input-output-python/>

3. Математичні вирази та оператори Python. URL : <https://www.bestprog.net/uk/2019/10/14/python-mathematical-arithmetic-operators-examples-ua/>

4. Математичні обчислення. URL : <https://www.wolframalpha.com/>

5. Python If ... Else (з інтерактивними вправами) https://www.w3schools.com/python/python_conditions.asp

Лабораторне заняття № 3. Програмування циклічних алгоритмів. Обробка текстових та числових послідовностей

Мета: вивчити принципи побудови циклічних алгоритмів для автоматизованої обробки текстових даних та опанувати методи нормалізації тексту й обробки винятків.

План

1. Поняття циклічних алгоритмів.
2. Цикл `while`.
3. Оператори `break`, `continue`, `else`, `pass`.
4. Нескінченний цикл.
5. Цикл `for`
6. Оператор `range()`

3.1 Поняття циклічних алгоритмів

Цикл – це структура, що передбачає багаторазове виконання певної послідовності дій (*тіло циклу*). Використання циклів дозволяє зробити код компактним та ефективним. Змінні, значення яких змінюються під час кожного повторення, називаються *параметрами циклу*.

Якщо кількість повторень заздалегідь невідома, а завершення обчислень залежить від досягнення певної умови, такий процес називають *ітераційним*.

Основні правила побудови циклів:

1. Повторювані дії описуються в коді лише один раз.
2. Вхід у цикл здійснюється виключно через його початок.
3. Всі змінні циклу мають бути визначені до моменту входу в нього.
4. Алгоритм повинен обов'язково містити умову завершення, щоб уникнути нескінченного виконання («зациклення»).

3.2 Цикл `while`

Оператор `while` використовується, коли кількість повторень заздалегідь невідома, але визначена умова їх припинення.

Ключові особливості:

- перевірка умови здійснюється перед кожною ітерацією. Цикл виконується доти, доки умова повертає `True`;
- структура циклу: заголовок з умовою, тіло циклу (з відступами) та необов'язковий блок `else` (див. рис. 3.1);
- блок `else`: виконується лише у випадку «природного» завершення циклу (коли умова стала хибною) і ігнорується, якщо вихід відбувся через інструкцію `break`.

```
while <test>:
    <st1>
else:
    <st2>
```

Рисунок 3.1 – Загальний синтаксис оператора while

Оператор while дозволяє багаторазово виконувати певні дії в залежності від деякої <умови> (виразу логічного типу, який приймає тільки значення True або False). Цикл виконується поки <Умова>=«істина». Як тільки <Умова> порушується, виконання циклу завершується (див. рис. 3.2).

```
1 # Надрукувати числа від 1 до 10
2 count = 1
3 while count <= 10: # поки значення змінної count менше або дорівнює 10
4     print(count)   # друкуємо поточне значення змінної count
5     count = count + 1 # збільшуємо поточне значення змінної count 1
6
7 pep8(_ih)
8
```

Рисунок 3.2 – Друкування чисел від 1 до 10, цикл while

Нижче продемонстровано фрагмент програми, який демонструє використання циклу while у типовій лінгвістичній задачі – зборі текстових даних до появи певного маркера (стоп-слова) (див. рис. 3.3). У лінгвістичних дослідженнях часто виникає потреба динамічно формувати масиви даних (корпуси), коли кількість елементів заздалегідь невідома.

```
1 corpus = []
2 stop_marker = "стоп"
3
4 print(f"Введіть слова для аналізу (для завершення введіть '{stop_marker}'):")
5
6 while True:
7     word = input("Слово: ").strip().lower()
8     if word == stop_marker:
9         print("Процес збору даних завершено.")
10        break
11    corpus.append(word)
12
13 print(f"Кількість зібраних лексем: {len(corpus)}")
14 print(f"Ваш масив даних: {corpus}")
```

Рисунок 3.3 – Автоматизований збір лексем до появи стоп-слова

Проаналізуємо наведену програму. По-перше, заздалегідь невідомо, скільки слів введе користувач, а тому використовуємо while True (забезпечує «нескінченне» повторення) з оператором break (здійснює передачу керування на наступний оператор після циклу). Методи .strip().lower() гарантують, що програма розпізнає слово «Стоп » (з пробілом та великою літерою) як сигнал до зупинки. Таким чином, список corpus поступово наповнюється даними під час кожної ітерації циклу. Зазначимо, що процес нормалізації даних

приводить різні варіанти написання стоп-слова «стоп» до одного стандарту, щоб програма могла їх коректно порівняти.

3.3 Оператори `break`, `continue`, `else`, `pass`

Розглянемо дві прості інструкції, які можна використати тільки усередині циклів – інструкції `break` і `continue`. У Python:

`break` виконує перехід за межі циклу (усієї інструкції циклу);

`continue` виконує перехід на початок циклу (в рядок заголовка);

`pass` нічого не робить: це порожня інструкція.

Блок `else` виконується, тільки в разі планового завершення циклу (без використання інструкції `break`).

З урахуванням інструкцій `break` і `continue` цикл `while` має такий загальний вигляд:

```
while <test1>:
    <st1># Тіло циклу
if<test2>:
    break # Вийти з циклу пропустивши частину else
if<test3>:
    continue # Перейти на початок циклу, до test1
else:
    <st2># Виконується, якщо не була використана інструкція break
```

Рисунок 3.4 – Загальний синтаксис оператора `while`

Команди `break` і `continue` дозволяють змінювати хід виконання циклів `while` та `for`. Як правило, вони працюють у зв'язці з оператором `if`, активуючись лише при виконанні певної логічної умови.

Інструкція `pass` – це порожній оператор, який не виконує жодних дій. Його використовують як синтаксичний заповнювач у випадках, де структура мови вимагає наявності коду, але фактичні операції в цій точці не передбачені (див. рис 3.5).



```
1 while True:
2     pass
3
1 while True:
2     ...
3
```

Рисунок 3.5 – Інструкції `pass` та `...` відповідно

Інструкція `continue` виконує перехід на початок циклу. Вона іноді дозволяє уникнути використання вкладених інструкцій. Інструкція `break` ініціює миттєвий вихід із циклу `while` або `for`. Весь наступний код у поточному блоці циклу при цьому не виконується. На рисунку 3.6 наведено приклад обробки запитів: цикл працює ітеративно, доки користувач не введе слово «quit», що активує оператор `break` і завершує виконання.

```

1 while 1 :
2     name = input("Напишіть своє ім'я ... ")
3     if name == 'quit' :
4         break
5     age = input('Напишіть свій вік ... ')
6     print('Привіт', name, ' => ', age, 'років')
7
8

```

Напишіть своє ім'я ... Джон
 Напишіть свій вік ... 18
 Привіт Джон => 18 років
 Напишіть своє ім'я ... quit

Рисунок 3.6 – Нескінченний цикл з передбаченим виходом

3.4 Нескінченний цикл

Нескінченний цикл – це циклічна ділянка в програмі, в якій не передбачені засоби виходу з циклу при досягненні деякої умови (див. рис. 3.7).

```

1 while 1 :
2     xValue = input("напишіть якесь слово ...")
3     print("\n", xValue)

```

напишіть якесь слово ...hello

 hello
 напишіть якесь слово ...hello

Рисунок 3.7 – Приклад організації нескінченного циклу

Умова циклу завжди істинна, тому введення даних триватиме нескінченно. Зупинити виконання можна лише вручну: через Ctrl+Break у терміналі або натисканням «Stop» у середовищі Colab.

На рисунку 3.8 наведено модифіковану програму з рисунку 3.7 Вихід з циклу відбувається при натисненні символу «q», керування передається оператору break, після чого відбувається вихід з циклу.

```

1 while 1 :
2     xValue = input("напишіть якесь слово ...")
3     print(xValue)
4     if xValue == 'q':
5         break

```

напишіть якесь слово ...Python
 Python
 напишіть якесь слово ...q
 q

Рисунок 3.8 – Приклад організації нескінченного циклу

3.5 Цикл for

Інструкція `for` реалізує обхід елементів послідовності (див. рис. 3.9). На кожній ітерації змінна циклу приймає значення наступного елемента, доки об'єкт не буде вичерпано. Логіка роботи керуючих інструкцій (`break`, `continue`, `else`) ідентична циклу `while`.

```
for <target> in <object> :  
    # Пов'язує елементи об'єкта зі змінною циклу  
    <Statements>  
    # тіло циклу: використовує змінну циклу  
else:  
    <Statements>  
    # Якщо не потрапили на інструкцію break
```

Рисунок 3.9 – Загальний синтаксис оператора `for`

Цикл `for` – головний інструмент лінгвіста. Запис `for char in "text"`: Python сприймає слово як чергу з літер і дозволяє «опитати» кожну по черзі. Це значно простіше, ніж вручну рахувати індекси. На рисунку 3.10 наведено приклад використання циклу `for` для того, щоб надрукувати всі слова із заданого тексту:

```
1 import nltk  
2 nltk.download('punkt')  
3 text1 = "One of the most popular summer festivals in the UK is the Edinburgh Festival. "\  
4         "The Edinburgh Festival has happened in August every year since 1947, except for 2020. "\  
5         "The idea for a festival came from Rudolf Bing, an Austrian who loved and sponsored the arts. "\  
6         "Bing was the general manager of the famous Opera House at Glyndebourne in the South of England. He wanted to organise"\  
7         "an international festival of music and opera to help fund the Opera House. Different"\  
8         "cities were considered, including Oxford, but finally, Edinburgh, which already had a "\  
9         "long history of festivals, was chosen. Two important events took place during the first"\  
10        "International Festival. Firstly, a week-long film festival was organised by the"\  
11        "Edinburgh Film Guild. Secondly, eight theatre groups arrived uninvited. They were"\  
12        "not allowed to perform in the International Festival, so they set up their"\  
13        "shows in other venues in the city. That became the 'Edinburgh Festival Fringe.'"  
14  
15 rez = nltk.word_tokenize(text1)  
16 for word in rez :  
17     print(word, end = ' ')
```

One/of/the/most/popular/summer/festivals/in/the/UK/is/the/Edinburgh/Festival./The/Edinburgh/Festival/has/happened/in/August/every/year,
Festival dates: Edinburgh Festival Fringe, Edinburgh Festival Fringe, Edinburgh Festival Fringe, Edinburgh Festival Fringe, Edinburgh Festival Fringe,
Festival dates: Edinburgh Festival Fringe, Edinburgh Festival Fringe, Edinburgh Festival Fringe, Edinburgh Festival Fringe, Edinburgh Festival Fringe,

Рисунок 3.10 - Оператор `for` для розбиття тексту на слова

3.6 Оператор `range()`

Для виконання дій із заздалегідь відомою кількістю повторень використовують цикл `for` у поєднанні з функцією `range()`, яка генерує послідовність цілих чисел (див. рис. 3.11).

```
range(start, stop, step)
```

Рисунок 3.11 – Синтаксис функції `range()`

`start`: початок діапазону (типово – 0);

stop: кінцеве значення (важливо: воно не входить у послідовність!);
step: крок зміни значень (типово – 1).

Запис `range(1, 10, 2)` створить послідовність: 1, 3, 5, 7, 9.

Функцію `range()` часто використовують для перебору слів у реченні за їхніми індексами (номерама). Наприклад, якщо потрібно обробити кожне друге слово в тексті, то крок дорівнюватиме 2 (див. рис. 3.12).

```
1 for i in range(2, 20, 2) :  
2     print(i, end = ' ')
```

2 4 6 8 10 12 14 16 18

Рисунок 3.12 – Застосування функції `range()` з кроком 2

На рисунку 3.13 наведено приклад програми для пошуку заданих літер в тексті:

```
1 myText = input('Надрукуйте текст для пошуку букв...')  
2 symbs = ['a', 'o', 'e']  
3 print("Шукаємо такі букви ", symbs)  
4 for s in symbs :  
5     if s in myText :  
6         print("В тексті є буква ", s)  
7         continue  
8     else :  
9         print("В тексті немає букви ", s)  
10 print("Роботу програми заврешено")
```

Рисунок 3.13 – Пошук букв в тексті

У змінну `myText` записуємо текст, в якому здійснюватиметься пошук букв. В змінну `symbs` записано букви, які потрібно знайти в тексті. В рядку №4 здійснюється перебір букв з послідовності `symbs`. Потім, якщо знайдено поточну букву, буде виведено відповідне повідомлення (рядок №6). Далі керування передається оператору `continue`, внаслідок чого відбувається повернення до рядку №4 і в «розробку» беремо наступну букву. Якщо ж поточну букву з набору `symbs` не знайдено, то буде виведено відповідне повідомлення (рядок №9). Результат роботи програми наведено на рисунку нижче:

```
Надрукуйте текст для пошуку букв...Text for text  
В тексті немає букви  a  
В тексті є буква  o  
В тексті є буква  e  
Роботу програми заврешено
```

Рисунок 3.14 – Результат роботи програми пошуку букв в тексті

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №3

В середовищі програмування Google Colab виконайте чотири завдання відповідно варіанта.

- Для кожного завдання створюйте окрему кодову комірку. Формулювання завдань наводити в текстових комірках.

- Відповіді на контрольні запитання оформіть наприкінці блокноту в текстовому полі.

- Налаштуйте доступ для коментування.

- За результатами роботи оформіть текстовий звіт (шаблон міститься на сторінці курсу та додайте до цього документа посилання на ваш блокнот).

- Захистіть роботу.

Варіанти завдань до лабораторної роботи №3

Завдання №1. Модифікуйте програму із завдання №2 (лабораторна робота №2) для реалізації багаторазового виведення даних. Налаштуйте цикл `while` так, щоб фраза друкувалася N разів (де N – число, введене користувачем). Використовуйте конструкцію `try-except` для валідації вводу та запобігання помилкам при введенні нечислових значень.

Завдання №2. Задано текст (користувач вводить з клавіатури або безпосередньо в програмі) та список з декількох слів (див. варіанти нижче). Знайдіть слова з переліку у введеному користувачем тексті. Якщо поточне слово в тексті знайдено або якщо не знайдено виведіть на екран відповідне повідомлення. Використовуйте цикл `for` та скористайтесь прикладом (див. рис. 3.21). Приготуйте три тексти для тестування програми (всі слова зі списку наявні в тексті; тільки декілька слів з переліку наявні в тексті; жодного слова з переліку в тексті немає.)

Варіанти переліків слів:

1. Сад, яблуко, персик, село, сонце.
2. Аномалія, багато, вагон, два, радіація.
3. Айстра, меценат, вантаж, депозит, радощі.
4. Адвокат, бадмінтон, велосипед, дворянин, регіон.
5. Апельсин, баклажан, вершина, довжина, розкіш.
6. Астрономія, байкер, винагорода, дисплей, садівник.
7. Аромат, банан, виріб, дуга, ступінь.
8. Атракціон, бджола, вітамін, двірник, москіт.
9. Азалія, бензин, вода, добриво, приятель.
10. Акваланг, брукета, ворона, дуло, морфема.
11. Авокадо, боротьба, вугілля, дифузор, флот.
12. Антрекот, боротьба, ворона, гніздо, хмара.
13. Республіка, дирижабль, дорога, будинок, платіж.
14. Засіб, кіно, вистава, музика, стадіон.
15. Спорт, товар, ринок, птах, фрукт, прожектор.

Завдання №3. Відповідно до завдання напишіть програму, яка використовує цикл `for` для ітерації по рядку або списку слів.

1. Напишіть програму, яка зчитує слово та виводить кожен його символ у новому рядку.

2. Напишіть програму, яка перебирає рядок і друкує лише голосні літери (а, о, у, е, и, і).

3. Створіть цикл, який проходить по введеному реченню та виводить лише приголосні літери.

4. Напишіть програму, яка запитує у користувача текст і рахує, скільки разів у ньому зустрічається літера «а».

5. Дано список лінгвістичних термінів (наприклад, [«фонема», «морфема», «лексема»]). Виведіть кожен термін та кількість літер у ньому.

6. Пройдіть циклом по реченню і виведіть лише ті літери, які записані у верхньому регістрі (великі літери).

7. Дано список імен. Виведіть лише ті, що починаються на літеру «О».

8. Дано список слів. Виведіть лише ті слова, які закінчуються на «-ння» (використовуйте `if` всередині циклу).

9. Напишіть програму, яка замінює всі пробіли в реченні на символ підкреслення `_` під час виводу (символ за символом).

10. Виведіть кожен символ слова разом із його порядковим номером (індексом) у рядку.

11. Дано список слів (наприклад, [«Об'єднані», «Нації»]). Виведіть лише першу літеру кожного слова.

12. Пройдіть циклом по тексту і виведіть лише літери, ігноруючи знаки пунктуації (крапки, коми, знаки оклику).

13. Напишіть програму, яка перевіряє рядок і виводить усі цифри, які в ньому зустрічаються.

14. Напишіть програму, яка зчитує слово і виводить кожен його символ двічі (наприклад, «мова» -> «ммоовваа»).

15. Дано список слів. Виведіть лише ті слова, довжина яких перевищує 7 символів.

Завдання №4. Обчисліть значення y відповідно до свого варіанта. Використовуйте цикл `while` (завдання а) та цикл `for` (завдання б, ітерувати за змінною n). На кожному кроці обчислення друкувати поточне значення x , y

Позначення: Δx – крок зміни змінної x , n – кількість повторень.

Варіант	Функція	Завдання а)	Завдання б)
1	$y = \frac{\cos^2 x}{x^2 + 1}$	$3,8 \leq x \leq 7,6 ; \Delta x = 0,6$	$0,5 \leq x ; \Delta x = 0,1 , n = 9$
2	$y = \frac{tg^2 0,5x}{x^3 + 7,5}$	$0,1 \leq x \leq 1,2; \Delta x = 0,1$	$0,5 \leq x ; \Delta x = 0,05 , n = 8$
3	$y = \frac{e^{2x} - 8}{x + 3}$	$-1 \leq x \leq 2,3 ; \Delta x = 0,7$	$1,5 \leq x ; \Delta x = 0,3 , n = 6$

Варіант	Функція	Завдання а)	Завдання б)
4	$y = \frac{x + \cos^2 2x}{3x}$	$2,3 \leq x \leq 5,4 ; \Delta x = 0,8$	$x \geq 1,2 ; \Delta x = 0,2 , n = 7$
5	$y = \frac{x + \cos^2 2x}{x + 2}$	$0,2 \leq x \leq 10 ; \Delta x = 0,8$	$x \geq 0,6 ; \Delta x = 1,5 , n = 6$
6	$y = \frac{\cos^3 x^2}{1,5x + 2}$	$2,3 \leq x \leq 7,2 ; \Delta x = 0,8$	$x \geq 0 ; \Delta x = 0,3 , n = 5$
7	$y = \frac{x^3 + 2x}{3\cos\sqrt{x} + 1}$	$0 \leq x \leq 2 ; \Delta x = 0,4$	$x \geq 0,3 ; \Delta x = 0,8 , n = 7$
8	$y = \frac{x + \sin 2x}{t^2 - 3}$	$2,4 \leq x \leq 6,9 ; \Delta x = 0,4$	$x \geq 3,1 ; \Delta x = 0,8 , n = 6$
9	$y = \frac{x^3 - 2}{3\ln(x)}$	$4,5 \leq x \leq 16,4 ; \Delta x = 2,2$	$x \geq 2 ; \Delta x = 1,5 , n = 5$
10	$y = \frac{2\sin^3 x}{3 x + 1}$	$-1 \leq x \leq 1 ; \Delta x = 0,25$	$x \geq -2,5 ; \Delta x = 0,15 , n = 6$
11	$y = \frac{2,3x + 8}{ 2\cos x + 1}$	$0 \leq x \leq 6,5 ; \Delta x = 1,1$	$x \geq 0,4 ; \Delta x = 0,9 , n = 7$
12	$y = \frac{5tg(x + 7)}{(x + 3)^2}$	$1,2 \leq x \leq 6,3 ; \Delta x = 0,2$	$x \geq 0,2 ; \Delta x = 0,1 , n = 5$
13	$y = \frac{1,5x - \ln 2x}{3x + 1}$	$2,5 \leq x \leq 9 ; \Delta x = 0,8$	$x \geq 0,8 ; \Delta x = 1,2 , n = 6$
14	$y = \frac{2tg^3 x}{3x - 1}$	$-1 \leq x \leq 1 ; \Delta x = 0,25$	$x \geq -2,5 ; \Delta x = 0,15 , n = 6$
15	$y = \frac{x + \sin 2x}{t^2 - 3}$	$2,4 \leq x \leq 6,9 ; \Delta x = 0,4$	$x \geq 3,1 ; \Delta x = 0,8 , n = 6$

Контрольні запитання

1. У чому полягає принципова різниця між циклами for та while? Наведіть приклад лінгвістичної задачі, де використання while є більш доцільним, ніж for.

2. Що саме стає «змінною циклу» під час виконання конструкції for char in text:?

3. Чому при обробці текстових послідовностей у циклі важливо використовувати методи .lower() та .strip()? До яких логічних помилок у підрахунках може призвести ігнорування цих методів?

4. Яку роль відіграє функція range() при роботі з текстовими масивами? Як за її допомогою звернутися до кожного другого символу в слові?

5. Опишіть алгоритм використання циклу разом із умовним оператором `if` для відбору слів із тексту, що мають специфічне закінчення (наприклад, суфікс *-уват-* або *-nn-*).

6. Навіщо при введенні кількості ітерацій користувачем (наприклад, скільки разів повторити фразу) варто використовувати конструкцію `try-except`? Яку помилку вона допомагає попередити?

Корисні посилання

1. W3Schools: Інтерактивний довідник методів рядків. URL : https://w3schoolsua.github.io/python/python_strings_methods_en.html

2. Офіційний туторіал Python: Керування потоком (Control Flow). URL : <https://docs.python.org/3/tutorial/controlflow.html>

3. Google for Developers: Python Strings. URL : <https://developers.google.com/edu/python/strings>

4. GeeksforGeeks: Вступ до NLP (обробка природної мови). URL : <https://www.geeksforgeeks.org/nlp/natural-language-processing-nlp-tutorial/>

5. Real Python: Рядки та текстові дані (з інтерактивними вправами) <https://realpython.com/python-strings/>

Лабораторне заняття № 4 Програмний аналіз текстових даних: робота з рядковими та списковими структурами

Мета: навчитися використовувати рядки та списки як базові структури для зберігання лінгвістичної інформації, опанувати алгоритми сегментації тексту (токенізації) та автоматизованого пошуку елементів у текстових масивах.

План

1. Рядки
2. Зрізи рядків
3. Типові літерали рядків та операції над ними
4. Списки
5. Приклади роботи зі списками та рядками
6. Генератори списків
7. Методи списків
8. Способи ітерування послідовностей

4.1 Рядки

Рядки (str) – це впорядковані послідовності символів, що використовуються для зберігання та обробки тексту.

Ключові особливості:

- Незмінність. Рядки належать до категорії незмінних об'єктів. Це означає, що окремі символи всередині рядка неможливо замінити – будь-яка операція модифікації створює новий рядок.

- Відсутність типу «символ». На відміну від багатьох інших мов, у Python немає окремого типу для одного символу. Символ – це рядок довжиною 1.

- Приналежність до послідовностей. Рядки мають спільні методи роботи (індексація, зрізи) з іншими послідовностями, такими як списки та кортежі.

Рядки записуються в одинарних ('...') або подвійних ("...") лапках. Ці способи позначення рядків є рівнозначними. Символ зворотного слешу (\) дозволяє вставляти спеціальні знаки:

Таблиця 4.1 Деякі Escape-послідовності Python

Позначення	Дія
\n	перехід на новий рядок
\t	горизонтальна табуляція
\\	виведення самого символу зворотного слешу
\'	виведення символу одинарних лапок
\"	виведення символу подвійних лапок

Числа можуть бути перетворені в рядки за допомогою функції `str()`. Наприклад, `str(123)` дасть рядок `'123'`. Якщо рядок є послідовністю знаків-

цифр, то він може бути перетворений в ціле число за допомогою функції `int()`, дробне – за допомогою функції `float()`.

З рядками не можна проводити ті ж операції, що і над числами, зокрема, множення рядків або ділення.

Але оператор «+» працює з рядками, хоча позначає іншу операцію: **конкатенацію** рядків (див. рис. 4.1). Оператор `*` теж можна використовувати до рядків, але за умови, що одним з операндів буде ціле число.

```
1 "hello" + "world"*3
```

```
'helloworldworldworld'
```

Рисунок 4.1 – Конкатенація та множення рядків на ціле число

Для отримання окремого символу з рядка використовують **оператор індексування**. Кожен символ у послідовності має свій порядковий номер – **індекс**.

РЯДОК[ІНДЕКС]

У Python індексація починається з 0 (наприклад, `word[0]` поверне перший символ). Загальна довжина рядка визначається за допомогою вбудованої функції `len()`. Пробіли є повноцінними символами, вони мають свій індекс і враховуються функцією `len()`.

Останній символ рядка завжди має індекс `len(рядок) - 1`. Також у Python можна використовувати від'ємні індекси: наприклад, `[-1]` завжди вказує на останній символ, що дуже зручно для швидкої перевірки закінчень слів (див. рис. 4.2).

```
1 str1[len(str1) - 1]
```

```
'o'
```

```
1 str1[-1]
```

```
'o'
```

Рисунок 4.2 – Звичайна та від'ємна індексація

На рисунку 4.3 наведено приклад програми, в якій обчислюється довжина введеного користувачем рядка, а також перший, п'ятий і останній символи. У випадку, коли довжина рядка складатиме менше п'яти символів – буде виведено відповідне повідомлення.

```
1 string1 = input("Enter string ")
2 lenght = len(string1)
3 if lenght < 5:
4     print("Lenght of the string is less then 5")
5 else :
6     print(string1[0], string1[4], string1[-1])
```

Рисунок 4.3 – Індєксація рядка

4.2 Зрізи рядків

Якщо індексація дозволяє отримати один символ, то **зрізи** дають змогу виділити цілу частину рядка (підрядок). Це один із інструментів у Python для лінгвістичного аналізу (виділення префіксів, коренів чи суфіксів).

РЯДОК[start: end: step]

Рисунок 4.4 – Синтаксис зрізу

start – індекс, з якого починається зріз (включається).

stop – індекс, на якому зріз закінчується (не включається).

step – крок, з яким вибираються символи (необов'язковий параметр).

Оператор [start: end] дозволяє взяти зріз послідовності з точки, на яку вказує зміщення start, до точки, на яку вказує зміщення end-1 (див. рис. 4.5).

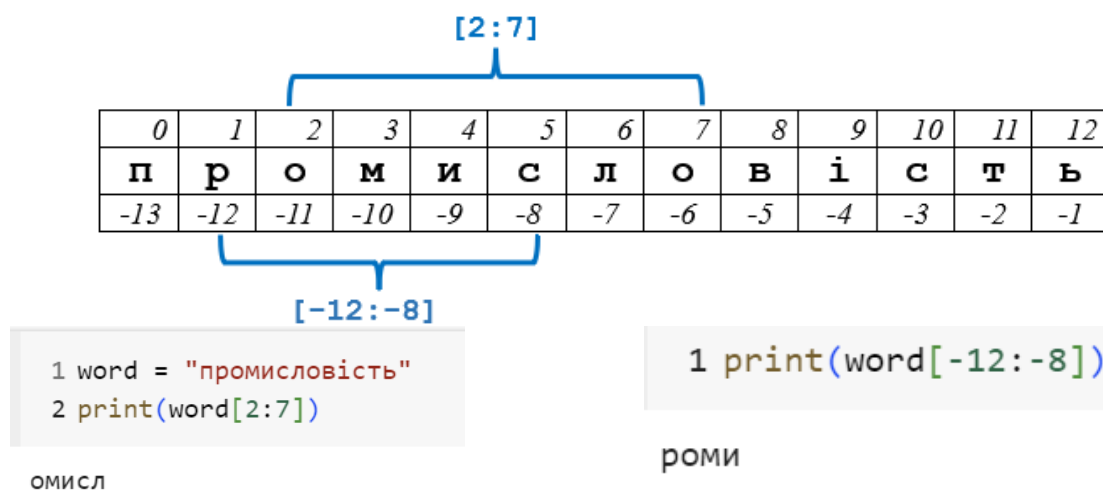


Рисунок 4.5 – Від'ємна індексація, зріз рядка

Оператор [:] дозволяє взяти зріз всієї послідовності від початку до кінця. Оператор [start:] дозволяє взяти зріз послідовності з точки, на яку вказує зміщення start, до кінця. Оператор [:end] дозволяє взяти зріз послідовності від початку до точки, на яку вказує зміщення end-1 (див. рис. 4.6)

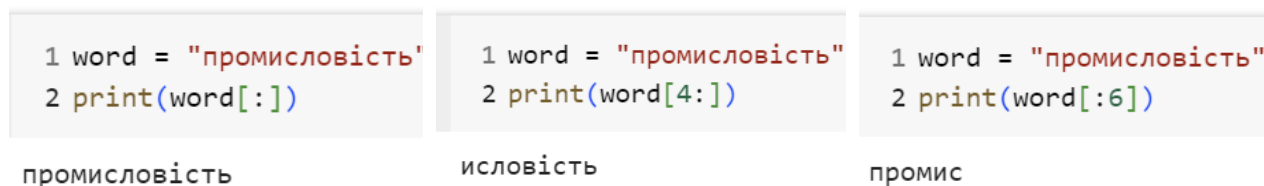


Рисунок 4.6 – Зрізи рядка

Рядок в мові Python неможливо змінити – в цьому випадку говорять, що це `immutable` тип. Спроба змінити символ в певній позиції або підрядку викличе помилку:

```
[87] 1 word = "промисловість"
      2 word[5]

      'с'
```

```
1 word[5] = 'ш'
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-88-07386005705c> in <cell line: 1>()
----> 1 word[5] = 'ш'

TypeError: 'str' object does not support item assignment
```

Рисунок 4.7 – Помилка при спробі напряду змінити елемент рядку

Речення або текст також можна представити як рядок і працювати з ним, як було показано вище

4.3 Типові літерали рядків та операції над ними

Порожній рядок має вигляд пари лапок (або апострофів), між якими нічого немає. Для роботи з рядками підтримуються операції над виразами, такі як конкатенація (об'єднання рядків), виділення підрядка, вибірка символів за індексами (за зсувом від початку рядка) тощо. Python пропонує ряд методів, які реалізують різні завдання роботи з рядками.

Таблиця 4.2 Операції з рядками

Операція	Інтерпретація
<code>S.find('pa')</code>	Виклик методу рядків: пошук
<code>S.count('a')</code>	Обчислення кількості входжень символу в рядок
<code>S.rstrip()</code>	Видалення провідних й кінцевих символів пробілу
<code>S.replace('pa', 'xx')</code>	Заміна «ра» на «xx»
<code>S.split(',')</code>	Розбиття за символом, який є роздільником
<code>S.isdigit()</code>	Перевірка вмісту
<code>S.lower()</code>	Перетворення регістра символів
<code>S.endswith('spam')</code>	Перевірка закінчення рядка
<code>'spam'.join(strlist)</code>	Формування рядка зі списку
<code>for x in S: print(x)</code>	Обхід елементів рядків в циклі
<code>'spam' in S</code>	Виклик методу рядків: пошук

Приклад 4.1 Обчислити та надрукувати довжину кожного слова в рядку.

Для обробки кожного слова в тексті спочатку необхідно виконати його сегментацію (розбиття) за допомогою методу `.split()`. Алгоритм роботи програми (див. рис. 4.8):

1. Токенізація. Метод `.split(' ')` розділяє рядок за пробілами та автоматично формує список окремих слів.
2. В циклі поелементно перебираємо отриманий список.
3. Для кожного слова застосовується функція `len()`, яка визначає кількість символів у ньому.
4. Програма друкує слово разом із показником його довжини.

```
1 sent = "colorless green ideas sleep furiously"
2 sent1 = sent.split(' ')
3 for word in sent1:
4     print(len(word), word)

9 colorless
5 green
5 ideas
5 sleep
9 furiously
```

Рисунок 4.8 – Аналіз окремих слів у реченні

4.4 Списки

Список – це впорядкована колекція об’єктів, яка використовується для зберігання послідовностей значень у певному порядку. У Python списки записуються як перелік елементів через кому у квадратних дужках `[]`.

Основні властивості списків:

- **Гетерогенність.** На відміну від багатьох інших мов, список у Python може одночасно містити об’єкти різних типів (числа, рядки, інші списки чи кортежі).

```
1 ['one', 10, 2.3, [10, 20]]

['one', 10, 2.3, [10, 20]]
```

Рисунок 4.9 - Список, що складається з елементів різних типів

- **Змінність (Mutability).** Це головна відмінність від рядків. Можна змінювати вміст списку: додавати нові елементи, видаляти існуючі або перезаписувати їх без створення копії всього об’єкта.

```
1 phrase1 = ['colorless', 'green', 'ideas']
2 phrase1[0] = 'colorful'
3 phrase1

['colorful', 'green', 'ideas']
```

Рисунок 4.10 – Заміна елемента списку

- **Індексація.** Кожен елемент має свій порядковий номер (індекс). Система індексації ідентична тій, що використовується в рядках (починається з 0).

```
[36] 1 len(phrase1)
```

```
3
```

```
[37] 1 phrase1[0]
```

```
'colorless'
```

```
[38] 1 phrase1[-1]
```

```
'ideas'
```

Рисунок 4.11 – Індексція списків

- **Дублювання.** Один і той самий об'єкт може зустрічатися в списку необмежену кількість разів.

Якщо рядок – це послідовність літер, то список – це зазвичай послідовність слів (токенів). Саме у списках найзручніше зберігати результати морфологічного аналізу, словники синонімів або масиви речень текстового корпусу.

Списки аналогічно до рядків можуть поєднуватись.

```
[43] 1 phrase2 = phrase1 + ['sleep', 'furiously']
```

```
2 phrase2
```

```
['colorless', 'green', 'ideas', 'sleep', 'furiously']
```



```
1 phrase1
```

```
['colorless', 'green', 'ideas']
```

Рисунок 4.12 – Об'єднання списків

4.5 Приклади роботи зі списками та рядками

Приклад 4.2. Дано одновимірний масив А числових значень, що нараховує N елементів. Знайти середнє арифметичне елементів масиву.

Алгоритм (див. рис. 4.13) передбачає введення користувачем кількості значень, після чого запускається цикл для збору та сумування дійсних чисел. На основі отриманої суми розраховується та відображається середнє арифметичне значення вибірки.

```
1 N = int(input("Кількість елементів : "))
2 Suma = 0
3 for i in range (N, 0, -1) :
4     element = float(input("Введіть число : "))
5     Suma = Suma + element
6 Average = Suma / N
7 print("Середнє арифметичне дорівнює : ", Average)
```

```
Кількість елементів : 5
Введіть число : 1
Введіть число : 2
Введіть число : 3.6
Введіть число : 7.8
Введіть число : 2.34
Середнє арифметичне дорівнює : 3.348
```

Рисунок 4.13 – Обчислення середнього арифметичного

У цій версії програми ми переходимо від простого накопичення суми в змінній до формування динамічного списку. Це дозволяє не лише отримати фінальний результат, а й зберегти всі введені дані для подальшої обробки.

```

1 N = int(input("Кількість елементів : "))
2 lst = []
3 for i in range(N, 0, -1) :
4     element = float(input("Введіть число : "))
5     lst.append(element)
6 Average = sum(lst) / N
7 print(lst)
8 print("Середнє арифметичне дорівнює : ", Average)

```

```

Кількість елементів : 5
Введіть число : 1
Введіть число : 2
Введіть число : 3.6
Введіть число : 7.8
Введіть число : 2.34
[1.0, 2.0, 3.6, 7.8, 2.34]
Середнє арифметичне дорівнює : 3.348

```

Рисунок 4.14 – Формування списку із введених елементів

Приклад 4.3 Розглянемо методи, які не створюють нові списки а модифікують початкові: `.sort()` (сортування) і `.reverse()` (обернення).

```

1 phrase2 = ['colorful', 'green', 'ideas', 'sleep', 'furiously']
2 phrase2.sort()
3 phrase2

```

```
['colorful', 'furiously', 'green', 'ideas', 'sleep']
```

```

1 phrase2.reverse()
2 phrase2

```

```
['sleep', 'ideas', 'green', 'furiously', 'colorful']
```

Рисунок 4.15 – Сортування списку

Приклад 4.4 Далі використаємо методи `append` (додати) і `index` (індексувати).

```

1 phrase2.append('said')
2 phrase2.append('Rossum')
3 phrase2

```

```
['sleep', 'ideas', 'green', 'furiously', 'colorful', 'said', 'Rossum']
```

```
1 phrase2.index('green')
```

```
2
```

Рисунок 4.16 – Додавання та індексування елементів списку

Приклад 4.5 Вивести на екран всі елементи списку та обчислити їхню довжину.

```

1 phrase2 = ['colorful', 'green', 'ideas', 'sleep', 'furiously']
2 for word in phrase2 :
3     print(len(word), word)

```

```

8 colorful
5 green
5 ideas
5 sleep
9 furiously

```

Рисунок 4.17 – Обробка елементів списку

Для програмної реалізації використано цикл по кожному елементу списку.

Приклад 4.6 Розробіть програму для автоматизованого розрахунку середньої довжини токена. Програма повинна приймати список слів, вилучених із тексту, обчислювати їхню сумарну довжину та ділити її на загальну кількість слів у списку.

```
1 phrase2 = ['colorful', 'green', 'ideas', 'sleep', 'furiously']
2 total = 0
3 for word in phrase2 :
4     total += len(word)
5 total = total / len(phrase2)
6 print(total)
```

6.4

Рисунок 4.18 – Розрахунок середньої довжини тексту

Для прикладного лінгвіста такі обчислення є базовими для багатьох задач: оцінка складності тексту, стилometрія (середня довжина слова або речення є індивідуальною ознакою автора), машинний переклад (ці дані допомагають оцінити обсяг обчислювальних ресурсів, необхідних для обробки конкретного корпусу текстів).

Приклад 4.7 Вивести на екран слова з речення, які розташовано на парних індексах.

Перед виведенням результатів текст необхідно сегментувати на окремі лексеми за допомогою методу `.split()`. Отриманий список слів опрацьовується в циклі, де за допомогою умовних операторів виконується вибіркова візуалізація елементів.

```
1 text = 'Agatha Christie was born in Devon, England in 1890 as the daughter of\
2 a British army captain. During the First World War she worked in a\
3 hospital as a nurse. Later on she got a job in a pharmacy.\
4 This influenced many of her crime stories because some of her victims\
5 were poisoned.'
6 text1 = text.split(' ')
7 print(text1)
```

['Agatha', 'Christie', 'was', 'born', 'in', 'Devon,', 'England', 'in', '1890', 'as

```
1 for word in text1 :
2     if (text1.index(word) % 2 != 0) :
3         print(word)
```

Christie
born
Devon,
as
daughter
a
army

Рисунок 4.19 – Токенізація та відбір елементів тексту

У прикладній лінгвістиці метод `.split()` є базовим етапом **токенізації** – розбиття неструктурованого тексту на одиниці, з якими комп'ютер може працювати окремо.

Приклад 4.8 У прикладній лінгвістиці часто виникає потреба перетворення текстового рядка на список токенів та зворотна операція – збірка речення зі списку слів. Наведений приклад ілюструє ітераційний підхід: за допомогою циклу ми формуємо цілісний рядок, додаючи пробіл перед кожною лексемою під час об’єднання елементів списку.

```
1 phrase2 = phrase1 + ['sleep', 'furiously']
2 phrase2

['colorful', 'green', 'ideas', 'sleep', 'furiously']

1 str1 = ''
2 for word in phrase2 :
3     str1 += ' ' + word
4 str1

' colorful green ideas sleep furiously'
```

Рисунок 4.20 – Перетворення списку на рядок

У мові Python існує більш короткий спосіб зробити це однією командою – метод `.join()`.

```
1 #імпортуємо модуль string
2 import string
3 #список, який необхідно конвертувати в рядок
4 phrase2 = ['colorful', 'green', 'ideas', 'sleep', 'furiously']
5 #створюємо рядок з пробілом, щоб елементи списку були розділені пробілом
6 phrase3 = ' '
7 #через метод join доєднуємо до рядка по одному слову зі списку
8 print(phrase3.join(phrase2))
9 # метод працює так: один елемент зі списку, інший - з рядка, а тому пробіл

colorful green ideas sleep furiously
```

Рисунок 4.21 – Конвертація списку в рядок

4.6 Генератори списків

У прикладній лінгвістиці, де часто доводиться працювати з великими масивами токенів, у Python є **генератори списків**. Це компактний спосіб створення нових списків шляхом застосування певних операцій до кожного елемента вже існуючої послідовності.

Генератор подібний до циклу `for`, але дозволяє записати процес трансформації даних в один рядок, що робить код більш швидким. Загальний вигляд генератора такий:

[вираз `for` змінна `in` послідовність]

змінна – ідентифікатор деякої змінної;

послідовність – послідовність значень, які приймає дана змінна (це може бути список, рядок або об’єкт отриманий за допомогою функції `range`).

вираз – деякий вираз, що як правило, залежить від використаної в генераторі змінної, яким будуть заповнені елементи списку.

```
1 words = ["мова", "програмування", "Python"]
2 lengths = [len(w) for w in words]
3 print(lengths)
```

... [4, 13, 6]

Рисунок 4.22 – Генерування списку з елементів списку words

Переваги використання генераторів:

- Лаконічність коду.
- Наочність модифікацій, які відбуваються з даними.
- Генератори працюють швидше за звичайні цикли при обробці великих текстових файлів.

Можлива і більш складна конструкція генератора списків. В програмі, яку наведено нижче, список формується з усіх можливих пар елементів рядків 'list' та 'spam', окрім елементів 'i' та 'a'. Пари формуються таким чином: перший елемент з рядка 'list', другий – з рядка 'spam' (див. рис. 4.23).

```
1 c = [ c + d for c in 'list' if c != 'i' for d in 'spam' if d != 'a' ]
2 c
```

['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']

Рисунок 4.23 – Генерування списку з елементів двох рядків

Але в складних випадках краще користуватися звичайним циклом for для генерації списків.

4.7 Методи списків

Python надає потужний набір вбудованих методів для гнучкого керування списками як динамічними структурами даних. Опанування цих інструментів дозволяє прикладним лінгвістам ефективно автоматизувати процеси редагування текстових масивів: від додавання нових tokenів до швидкого пошуку та впорядкування елементів корпусу.

Таблиця 4.3 Методи списків

Метод	Дія, що реалізується
List.append(x)	Додає елемент в кінець списку
List.extend(L)	Розширює список list, додає в кінець всі елементи списку L
List.insert(i, x)	Вставляє на i-й елемент значення x
List.remove(x)	Видаляє перший елемент у списку, що має значення x. ValueError, якщо такого елемента не існує

Метод	Дія, що реалізується
List.pop([i])	Видаляє i-й елемент та повертає його. Якщо індекс не вказано, видаляється останній елемент
List.index(x, [start[, end]])	Повертає положення першого елемента зі значенням x (пошук ведеться від start до end)
List.index(x)	Повертає індекс елемента x в списку List
List.count(x)	Повертає кількість елементів зі значенням x
List.sort([key=функція])	Сортує список на основі функції
List.reverse()	Перевертає список
List.copy()	Поверхнева копія списку
List.clear()	Очищує список

Далі розглянемо низку прикладів застосування наведених методів.

1. Додавання до списку елемента на зазначену позицію

```
1 phrase = ['red', 'green', 'blue', 'yellow']
2 phrase.insert(1, 'black')
3 phrase

['red', 'black', 'green', 'blue', 'yellow']
```

Рисунок 4.24 – Метод .insert()

2. Додавання нового елемента в кінець списку

```
1 phrase = ['red', 'black', 'green', 'blue', 'yellow']
2 phrase.append('purple')
3 phrase

['red', 'black', 'green', 'blue', 'yellow', 'purple']
```

Рисунок 4.25 – Метод .append()

3. Підрахунок частоти вживання обраної мовної одиниці (лексеми або символу).

```
1 phrase = ['red', 'black', 'green', 'blue', 'yellow', 'purple']
2 phrase.count('black')
```

1

Рисунок 4.26 – Метод .count()

4. Видалення елемента зі списку

```
1 phrase = ['red', 'black', 'green', 'blue', 'yellow', 'purple']
2 phrase.remove('black')
3 phrase

['red', 'green', 'blue', 'yellow', 'purple']
```

Рисунок 4.27 – Метод `.remove()`

5. Видалення елемента із заданим індексом

```
1 phrase = ['red', 'green', 'blue', 'yellow', 'purple', 10, 20, 30, 40]
2 phrase.remove(0)

'red'

1 phrase

['green', 'blue', 'yellow', 'purple', 10, 20, 30, 40]
```

Рисунок 4.28 – Метод `.pop()`

6. Видалення елементів списку

```
1 phrase = ['red', 'green', 'blue', 'yellow', 'purple']
2 del(phrase[2])
3 phrase

['red', 'green', 'yellow', 'purple']

1 del(phrase[:2])
2 phrase

['yellow', 'purple']

1 del(phrase)
```

Рисунок 4.29 – Функція `del()`

Для представлення складних текстових даних (наприклад, таблиць частотності або матриць «слово-документ») у Python застосовують вкладені списки. Виведення такої інформації у зручному для читання табличному вигляді потребує координації двох циклів: зовнішнього – для переходу між горизонтальними блоками даних, та внутрішнього – для послідовного відображення елементів у кожному блоці через пробіл (див. рис. 4.30).

```
1 a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
2 for row in a :
3     for elem in row :
4         print(elem, end = ' ')
5     print()

1 2 3 4
5 6
7 8 9
```

Рисунок 4.30 – Вкладений (двовимірний) числовий список

Для виведення одного рядка можна скористатися методом `join()`:

```
1 a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
2 for row in a :
3     print(' '.join([str(elem) for elem in row]))
```

1 2 3 4
5 6
7 8 9

Рисунок 4.31 – Друкування вкладених списків

4.8 Способи ітерування послідовностей

Ітерування є фундаментальним інструментом лінгвістичного аналізу, що дозволяє програмі послідовно «читати» та опрацьовувати кожен елемент тексту – від окремих графем до цілих речень. Python пропонує різні підходи до перебору послідовностей, кожен із яких обирається залежно від задачі: від прямого проходження за значеннями до гнучких циклів із використанням індексів та лічильників.

Таблиця 4.4 Ітерування послідовностей

Вираз в 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 елменти s

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №4

В середовищі програмування Google Colab виконайте завдання відповідно варіанта.

- Для кожного завдання створюйте окрему кодову комірку. Формулювання завдань наводити в текстових комірках.
- Відповіді на контрольні запитання оформіть наприкінці блокноту в текстовому полі.
- Налаштуйте доступ для коментування.
- За результатами роботи оформіть текстовий звіт (шаблон міститься на сторінці курсу та додайте до цього документа посилання на ваш блокнот).
- Захистіть роботу.

Варіанти завдань до лабораторної роботи №4

Завдання 1. Виконайте завдання відповідно до варіанта

1. Дано рядок, що містить літери «о» та «а». Визначити, яка з них зустрічається частіше.
2. Перевірити, чи містить задане речення коми. Вивести відповідне інформаційне повідомлення.
3. У рядку, де гарантовано є літери «н» та «к», визначити, яка з них з'являється раніше при читанні зліва направо.
4. Визначити, чого в заданому рядку більше: цифр чи ком (за умови, що обидва типи символів присутні).
5. Користувач вводить літеру з клавіатури. Програма має обчислити загальну кількість її входжень у заданий рядок.
6. Для введенної користувачем літери знайти та вивести індекси її першого та останнього входження в тексті.
7. Користувач вводить речення та два окремих слова – «цільове» та «замінне». Програма повинна автоматично замінити всі входження цільового слова на замінне та вивести оновлений текст.
8. Реалізувати перевірку слова на паліндром (слово, що читається однаково з обох боків). Алгоритм має бути нечутливим до регістру (наприклад, слово «Зараз» має розпізнаватися як паліндром).
9. Дано речення, у якому слова розділені одним пробілом. Програма має знайти найдовше слово у тексті та вивести його разом із кількістю символів у ньому.
10. Дано повне ім'я файлу (наприклад, C:\Windows\System32\calc.exe). Виконати декомпозицію шляху за символом «\» та вивести кожен складову в окремому рядку.
11. Дано речення (мінімум 7 слів). Використовуючи інструкції циклу, виділити та вивести перші 6 слів.
12. У реченні (мінімум 3 слова) знайти та вивести два останні слова.
13. Знайти та вивести індекси всіх цифр, що містяться у заданому змішаному рядку (літери та цифри).

14. Підрахувати та вивести окремо кількість великих та малих літер у заданому рядку.

15. Визначити точну кількість входжень слова «або» у заданому тексті.

Завдання 2. Виконайте завдання відповідно до варіанта

1. Дано дієслово «перезавантажити». Використовуючи зріз, витягніть перші чотири літери, щоб отримати префікс «пере».

2. Дано прикметник у називному відмінку (наприклад, «лінгвістична»). Отримайте зріз останніх двох символів, щоб виділити флексію «ая».

3. Дано рядок тексту. Виведіть кожен другий символ цього рядка (крок зрізу), щоб перевірити, чи не приховано в ньому «шифровку».

4. Для терміна «корпусна лінгвістика» створіть зріз, який містить лише перші три літери першого слова для формування короткого словникового індексу («кор»).

5. Дано слово. Використовуючи зріз із від'ємним кроком, розгорніть слово у зворотному порядку та перевірте, чи збігається воно з оригіналом (наприклад, «ротатор»).

6. Текст закінчується знаком оклику: «Увага!». Отримайте рядок без останнього символу, використовуючи від'ємний індекс у зрізі.

7. Дано складне слово «навчально-науковий». Виділіть другу частину слова («науковий»), розрахувавши індекс після дефіса.

8. Дано лексему «мова». Виведіть усі можливі пари сусідніх символів («мо», «ов», «ва»), використовуючи зрізи в циклі.

9. Лінгвістична цитата подана у форматі «Текст». Отримайте вміст цитати, видаливши перший та останній символи за допомогою зрізу [1:-1].

10. Залиште першу літеру, а всі інші символи замініть на зірочки (використовуйте зріз першого символу та операцію множення рядка *).

11. Припустимо, що в списку слів «мова», «мовознавство», «мовний» корінь займає перші три позиції. Витягніть ці символи з кожного слова.

12. Реалізуйте власну логіку: візьміть перший символ рядка, переведіть його у верхній регістр, а до нього додайте зріз решти рядка у нижньому регістрі.

13. Файл корпусу названо за шаблоном 2026-04-08_text.txt. Використовуючи зріз, отримайте лише фрагмент із датою (перші 10 символів).

14. У слові «лісостеп» виділіть єднальний голосний «о», який знаходиться між 3-м та 5-м індексами (використовуйте зріз [3:4]).

15. Якщо речення довше за 20 символів, залиште лише перші 17 символів і додайте в кінці три крапки (...), використовуючи зрізи та конкатенацію.

Завдання 3. Виконайте завдання відповідно до варіанта

1. Дано список довжин речень у тексті. Перевірте, чи є в тексті речення, що складається рівно з 1 слова (пошук числа 1 у списку).

2. Дано список частот вживання слів. Знайдіть та виведіть усі значення, які перевищують «порог значущості» (наприклад, більше 50).

3. Знайдіть максимальне число у списку частот. Це число відповідатиме найбільш вживаному слову в корпусі.

4. Знайдіть індекс (позицію) найбільшого елемента в списку довжин слів. Це допоможе ідентифікувати «найважче» для аналізу речення.
5. Знайдіть мінімальне значення в списку (гапакс – слова, що зустрілися лише один раз) та визначте його індекс.
6. Користувач вводить число. Програма має підрахувати, скільки разів це число зустрічається в списку (наприклад, скільки в тексті речень довжиною рівно в 10 слів).
7. Виведіть усі індекси елементів, значення яких лежать у межах від 5 до 15 (типова довжина фрази для певного стилю).
8. Для заданого числа знайти індекс його першого та останнього входження в статистичний масив.
9. Перевірте, чи містить список від’ємні числа. У лінгвістичній статистиці (довжини, частоти) їх бути не може, тому їх наявність свідчить про помилку в розрахунках.
10. Виведіть усі парні числа зі списку довжин слів. Це може бути корисно для специфічних метрик ритміки тексту.
11. Обчисліть середнє арифметичне списку та знайдіть усі числа, які більші за це середнє значення.
12. Визначте, чи є в списку хоча б два однакових числа (наприклад, чи є два речення з абсолютно однаковою кількістю символів).
13. Створіть новий список, що містить усі індекси, за якими в оригінальному списку стоїть число 0 (пошук порожніх значень або пропусків).
14. Знайдіть у списку першу пару сусідніх чисел, які є однаковими (наприклад, два послідовних речення однакової довжини).
15. Знайдіть друге за величиною число у списку (елемент, що йде відразу за абсолютним максимумом). Це важливо для порівняння частотності двох найпопулярніших слів.

Завдання 4. Виконайте завдання відповідно до варіанта

1. Задано М слів. У кожному слові порахувати кількість входжень літери «о». Визначити слово, де ця буква зустрічається найчастіше. Перемістити його на останнє місце та вивести список.
2. Задано М слів. Порахувати кількість символів у кожному слові. Знайти найдовше слово, перемістити його на перше місце та вивести оновлений список на екран.
3. Задано М слів. Підрахувати кількість глухих приголосних (наприклад, «п», «к», «т», «с»). Визначити слово з найбільшою їх кількістю, перемістити його на другу позицію в списку та вивести результат.
4. Задано М слів. Порахувати, скільки разів у кожному слові зустрічаються літери «я», «ю», «є», «ї». Визначити слово з мінімальною (але не нульовою) кількістю цих літер, перемістити його на перше місце.
5. Задано М слів. Визначити, чи починається слово з великої літери. Порахувати такі слова. Всі слова, що починаються з великої літери, перемістити на початок списку та вивести його.

6. Задано М слів. Порахувати кількість слів, що закінчуються суфіксом «-ння». Визначити перше таке слово в послідовності, перемістити його в самий кінець і вивести список.

7. Задано М слів. У кожному слові визначити кількість унікальних літер (тих, що не повторюються). Знайти слово з максимальним показником унікальності, поставити його на перше місце.

8. Задано М слів. Порахувати сумарну кількість літер «ж», «ч», «ш», «щ» у кожному слові. Слово з найбільшим показником перемістити на останнє місце.

9. Задано М слів. Визначити кількість голосних літер у кожному слові (кількість складів). Знайти слово з найменшою кількістю складів, перемістити його на перше місце.

10. Задано М слів. Визначити, чи є слово паліндромом (читається однаково в обох напрямках). Знайти найдовший паліндром серед заданих слів, перемістити його на перше місце.

11. Задано М слів. Знайти слова, у яких зустрічаються дві однакові літери поспіль (наприклад, «нн», «тт»). Порахувати такі пари. Слово з найбільшою кількістю подвоєнь перемістити на передостаннє місце.

12. Задано М слів. Порахувати кількість літер «ф» та «г» (часто характерні для запозичень). Визначити слово, де ці літери зустрічаються найбільше, перемістити його на початок списку.

13. Задано М слів. Знайти всі слова, що закінчуються на «-ий» або «-ій» (ймовірні прикметники). Порахувати їх. Останнє таке слово в списку перемістити на першу позицію.

14. Задано М рядків, що можуть містити цифри (наприклад, дати або коди). Порахувати кількість цифр у кожному рядку. Рядок з найбільшою кількістю цифр перемістити на останнє місце.

15. Задано М слів. Визначити різницю між кількістю голосних і приголосних літер у кожному слові. Слово, де ця різниця мінімальна (баланс найближчий до нуля), перемістити на перше місце.

Контрольні запитання

1. У чому полягає фундаментальна різниця між рядком та списком з погляду їхньої змінності? Як ця властивість впливає на методи опрацювання тексту?

2. Як отримати останні три символи слова, використовуючи від'ємну індексацію?

3. Чому для великих масивів даних об'єднання списку слів через `.join()` є продуктивнішим за конкатенацію (додавання) рядків у циклі?

4. Як взаємодіють зовнішній та внутрішній цикли під час обходу такої двовимярного списку?

5. Сформулюйте приклад генератора, який би відфільтровував із тексту всі слова, довжина яких менша за три символи.

Корисні посилання

1. Python Tutorial: Data Structures (Official Documentation). URL: <https://docs.python.org/3/tutorial/datastructures.html>
2. Natural Language Processing with Python (NLTK Book). URL: <https://www.nltk.org/book/>
3. Real Python: Python "for" Loops (Definitive Guide). URL: <https://realpython.com/python-for-loop/>
4. Python Course for Linguists (by Folgert Karsdorp). URL: <https://www.google.com/search?q=https://www.karsdorp.dot-q.nl/python-course/>
5. W3Schools: Python String & List Methods Reference. URL: https://www.w3schools.com/python/python_ref_list.asp

Лабораторне заняття № 5. Структури даних Python. Кортежі, множини, словники

Мета: навчитися застосовувати базові структури даних Python для ефективної обробки та пошуку текстової інформації.

План

1. Множина.
2. Кортеж.
3. Словник.
4. Приклади роботи зі словниками

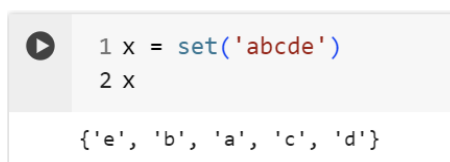
5.1 Множина

Множина – це неупорядкована колекція унікальних хешованих об’єктів. Її **ключовою властивістю** є автоматичне **усунення дублікатів**: кожен елемент зберігається в єдиному екземплярі, незалежно від частоти його додавання. Структура реалізує стандартні операції математичної теорії множин, такі як об’єднання, перетин та різниця. Оскільки множина є набором інших об’єктів, їй притаманні деякі властивості списків і словників:

- підтримання ітерацій;
- динамічна розмірність;
- можуть містити елементи різних типів.

Для прикладного лінгвіста **множина** – це передусім інструмент для отримання списку унікальних лексем тексту. Це набір елементів без фіксованого порядку, де кожне слово чи символ представлені лише один раз. Спроба додати вже існуючий елемент не змінює склад множини, що сприяє швидкій фільтрації повторів та порівняння різних словників.

Для **створення множини** використовується вбудована функція `set()` (див. рис. 5.1). Їй необхідно передати будь-який ітерований об’єкт (список, рядок, кортеж тощо). Результатом буде новий об’єкт-множина, що складається з усіх елементів переданої послідовності.



```
1 x = set('abcde')
2 x
{'e', 'b', 'a', 'c', 'd'}
```

Рисунок 5.1. – Задання множини

Множини підтримують стандартні математичні операції через вбудовані оператори. Об’єднання, перетин та різниця повертають нову множину, тоді як перевірка входження та порівняння (підмножина/надмножина) дають логічний результат (`True` або `False`).

Як ітеровані контейнери, множини підтримують функцію `len()`, цикли `for` та генератори (див. рис. 5.2). Проте через відсутність внутрішнього порядку

для них неможливі операції індексування та зрізів. У прикладній лінгвістиці це означає, що можна перебрати всі унікальні слова тексту в циклі, але неможливо звернутися до «першого» чи «п'ятого» слова в множині – для цього її спочатку доведеться перетворити на список за допомогою функції `list()`.

```
1 for item in set('abc'):
2     print(item * 3)
```

bbb
aaa
ccc

Рисунок 5.2. – Обробка елементів множини в циклі

Варто пам'ятати, що порожні фігурні дужки `{}` зарезервовані для створення **словника**. Щоб ініціалізувати **порожню множину**, необхідно використовувати функцію `set()`. При виведенні результату на екран порожня множина також відображається як `set()`, що дозволяє відрізнити її від словника (див. рис. 5.3).

```
1 w = set() # Пуста множина
2 print(type(w)) # <class 'set'>
3 s = {} # Літерал {} позначає порожній словник
4 print(type(s)) # <class 'dict'>
```

<class 'set'>
<class 'dict'>

Рисунок 5.3. – Оголошення порожньої множини та порожнього словника

Елементами множини можуть бути лише **незмінні** типи даних. Оскільки списки та словники є динамічними структурами, їхнє додавання до множини заборонено. Натомість для зберігання комплексних значень слід використовувати **кортежі**. У межах множини кортежі порівнюються за всім набором елементів, що до них входять.

5.2 Кортеж

Списки, рядки та кортежі належать до вбудованих типів послідовностей у Python, що дозволяє застосовувати до них універсальні операції індексування та отримання зрізів. **Кортеж (tuple)** формується як набір значень, розділених комами, і хоча в коді круглі дужки навколо них часто можна опускати, інтерпретатор завжди використовує їх при виведенні для коректного відображення вкладених структур (див. рис. 5.4).

```

1 t = 12345, 54321, 'hello!'
2 print(t[0]) # 12345
3 print(t) # (12345, 54321, 'hello!')
4 u = t, (1, 2, 3, 4, 5)
5 print(u) # ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
6 print(type(t), type(u)) # <class 'tuple'> <class 'tuple'>

```

Рисунок 5.4 – Оголошення кортежу

Головною особливістю кортежів є їхня **незмінність**, подібна до властивостей рядків, що виключає можливість прямого переприсвоєння значень окремим елементам, проте дозволяє імітувати зміни шляхом комбінування зрізів у нові об'єкти (див. рис 5.4.).

```

1 t = 1, ['foo', 'bar']
2 print(t) # (1, ['foo', 'bar'])
3 t[1] = []
4 #TypeError: 'tuple' object does not support item assignment
5 t[1].append('baz')
6 print(t) # (1, ['foo', 'bar', 'baz'])

```

Рисунок 5.5 – Зміна елементів кортежу

5.3 Словник

Словник у Python – це тип даних, де дані зберігаються у вигляді пар «ключ:значення». У словниках для пошуку інформації використовується унікальний ключ. Це робить словники зручним інструментом для лінгвістичного моделювання, наприклад, при створенні частотних словників, де ключем виступає слово, а значенням – кількість його вживань у тексті.

Ключами можуть бути числа, рядки або кортежі (якщо вони самі складаються з незмінних об'єктів). Використання списків як ключів заборонено, оскільки їхній вміст можна змінювати, що порушуватиме цілісність структури словника. Натомість значення, що відповідають ключам, можуть бути будь-якого типу, включно зі списками чи іншими словниками.

Синтаксично словник визначається за допомогою фігурних дужок {}, а для його ініціалізації з даними всередині дужок через кому перелічуються пари, розділені двокрапкою. (див. рис. 5.6).

```

1 price = {'apple': 40980, 'dell': 34139}
2 price['hp'] = 34127 # Доповнення словника
3 print(price) # {'apple': 40980, 'dell': 34139, 'hp': 34127}
4 print(price['apple']) # 40980
5 del price['dell'] # Видалення пари зі словника
6 price['asus'] = 4127
7 print(price) # {'apple': 40980, 'hp': 34127, 'asus': 4127}
8 print(price.keys()) # dict_keys(['apple', 'hp', 'asus'])

```

```

{'apple': 40980, 'dell': 34139, 'hp': 34127}
40980
{'apple': 40980, 'hp': 34127, 'asus': 4127}
dict_keys(['apple', 'hp', 'asus'])

```

Рисунок 5.6 – Оголошення та робота зі словником

Основні операції зі словниками:

- запис значення за ключем,
- зчитування запису та видалення за допомогою інструкції `del`.

! Важливо. Якщо записати значення за ключем, який уже існує в словнику, старе значення буде замінено новим без можливості відтворення старого. Звернення до відсутнього ключа призведе до виникнення виняткової ситуації `KeyError`.

Оскільки словник є неупорядкованою колекцією, порядок ключів при виведенні може бути довільним. Для отримання повного переліку всіх ключів використовується метод `keys()`. Отриманий результат можна перетворити на список або відсортувати за допомогою методу `sort()`, якщо лінгвістичне завдання вимагає алфавітного порядку (наприклад, при формуванні глосарія).

Таблиця 5.1. Методи роботи зі словниками

Метод	Дія
<code>dict.clear()</code>	Очищення вмісту словника
<code>dict.copy()</code>	Повертає копію словника
<code>dict.fromkeys (seq [, value])</code>	Створює словник з ключами з <code>seq</code> і значенням <code>value</code> (за замовчуванням <code>None</code>)
<code>dict.items()</code>	Повертає пари (ключ, значення) - кортежі
<code>dict.keys()</code>	Повертає ключі в словнику
<code>dict.pop(key [, default])</code>	Видаляє ключ і повертає значення. Якщо ключа немає, повертає <code>default</code> (за замовчуванням - виняток)
<code>dict.update([other])</code>	Оновлює словник, додаючи пари (ключ, значення) з <code>other</code> . Існуючі ключі перезаписуються. Повертає <code>None</code> (не новий словник!)
<code>dict.values()</code>	Повертає значення в словнику
<code>dict[key]</code>	Отримання значення з ключем <code>key</code> . Якщо ключ не існує, відображення реалізує спеціальний метод <code>__missing__</code> (<code>self, key</code>): якщо ключ не існує і метод <code>__missing__</code> не визначений, видається виняток <code>KeyError</code>
<code>dict[key] = value</code>	Змінити значення або створити пару ключ-значення, якщо ключ не існує
<code>key in d, key not in d</code>	Перевірка наявності ключа в словнику

5.4 Приклади роботи зі словниками

Приклад 5.1. Створити словник через функцію `dict()`

```
1 size = dict(small = "S", medium = "M", large = "L")
2 size
{'small': 'S', 'medium': 'M', 'large': 'L'}
```

Рисунок 5.7 – Створення словника через функцію `dict()`

! Важливо. Доступ до елементів словника здійснюється за ключем. Вказується ім'я словника, в квадратних дужках – ключ, доступ буде до відповідного значення `dictionary[key]` (див. рис. 5.8)

```
1 dictionary = {'drink': 'juice', 'food': 'spaghetti', 'cutlery': 'fork'}
2 print(dictionary['cutlery'])
```

fork

Рисунок 5.8 – Доступ до елементів словника

Приклад 5.2. Видалити запис зі словника.

Для цієї операції використовують ключ та функцію `del`

```
1 del dictionary['cutlery']
2 print(dictionary)
```

{'drink': 'juice', 'food': 'spaghetti'}

Рисунок 5.9 – Видалення запису зі словника

Приклад 5.3. Для зміни значення у словнику необхідно використати ключ

```
1 dictionary['cutlery'] = 'spoon'
2 print(dictionary)
```

{'drink': 'juice', 'food': 'spaghetti', 'cutlery': 'spoon'}

Рисунок 5.10 – Зміна значення у словнику

Приклад 5.4. Функція `zip()` надає можливість створити словник шляхом об'єднання списків ключів та значень

```
1 # Створення словника з списків ключів та значень
2 Numbers = dict(zip([1, 2, 3], ['One', 'Two', 'Three']))
3 print(Numbers)
```

{1: 'One', 2: 'Two', 3: 'Three'}

Рисунок 5.11 – Створення словника через функцію `zip()`

Приклад 5.5. Функцію `dict()` можна використовувати для перетворення двозначних послідовностей у словник. Наприклад, в результаті роботи автоматичного морфологічного аналізатора було отримано дані у вигляді списку

списків. Кожен внутрішній елемент містить два рядки: слово та назву частини мови, до якої вона відноситься. Цю послідовність необхідно перетворити на словник. Програмну реалізацію завдання наведено на рисунку 5.12

```
1 # дані отримані в ході морфологічного аналізу
2 tagged_data = [
3     ["мова", "NOUN"],
4     ["програмувати", "VERB"],
5     ["швидко", "ADV"],
6     ["цікавий", "ADJ"],
7     ["лінгвістика", "NOUN"]
8 ]
9
10 # перетворюємо список списків у словник за допомогою функції dict()
11 pos_dictionary = dict(tagged_data)
12
13 # виводимо отриманий словник
14 print("Сформований морфологічний словник:")
15 print(pos_dictionary)
16
17 # перевірка частини мови
18 word_to_check = "мова"
19 if word_to_check in pos_dictionary:
20     print(f"\nСлово '{word_to_check}' має тег: {pos_dictionary[word_to_check]}")
21
22 # унікальні лема, що потрапили в словник
23 print(f"Кількість слів у глосарії: {len(pos_dictionary)}")
```

Рисунок 5.12 – Перетворення списку у словник

Наведемо декілька пояснень до програми наведеної на рисунку 5.12. По-перше, через вбудовану функцію `dict()` здійснено перетворення вихідного списку `tagged_data` на словник `pos_dictionary`, де ключами є лексеми, а значеннями – їхні теги (рядок №11). Отриманий словник виводиться на рядках №14-15. Далі реалізовано перевірку наявності слова «мова» у словнику. Якщо слово є в словнику (рядок №18-20), програма виводить відповідне повідомлення та частину мови, до якої належить задане слово.

Приклад 5.6. Додавання або зміна елемента за допомогою `[key]`

Для додавання елемента до словника потрібно звернутись до елемента за його ключем і призначити значення. Якщо ключ вже був у словнику, існуюче значення замінюється новим. Якщо ключ новий, його значення додається до словника (див. рис. 5.13).

```
1 price = {'apple': 40980, 'dell': 34139, 'hp': 34127, 'asus': 4127}
2 price['lenovo'] = 41495 #додавання до словника
3 price['apple'] = 45500 #зміна значення
4 price

{'apple': 45500, 'dell': 34139, 'hp': 34127, 'asus': 4127, 'lenovo': 41495}
```

Рисунок 5.13 – Додавання до словника нового елемента

Приклад 5.7. Отримання всіх ключів за допомогою `keys()`

```
1 price = {'apple': 40980, 'dell': 34139, 'hp': 34127, 'asus': 4127}
2 price.keys()

dict_keys(['apple', 'dell', 'hp', 'asus'])
```

Рисунок 5.14 – Послідовність ключів словника

! Важливо. Метод `keys()` повертає об'єкт типу `dict_keys`, який є ітерабельним «виглядом» усіх ключів словника. Цей об'єкт не підтримує індексування, а тому, коли вам потрібен саме список (наприклад, для сортування чи вибору елемента за номером), необхідно виконати явне перетворення за допомогою функції `list()` (див. рис. 5.15).

```
1 list(price.keys())

['apple', 'dell', 'hp', 'asus']
```

Рисунок 5.15 – Перетворення у список послідовності ключів словника

Приклад 5.8. Отримання всіх значень за допомогою `values()`

```
1 list(price.values())

[40980, 34139, 34127, 4127]
```

Рисунок 5.16 - Перетворення у список послідовності значень словника

Приклад 5.9 Отримання всіх пар ключ-значення за допомогою `items()`

Для отримання всіх пар ключ-значення зі словника, використовуємо функцію `items()`:

```
1 list(price.items())

[('apple', 40980), ('dell', 34139), ('hp', 34127), ('asus', 4127)]
```

Рисунок 5.17 – Перетворення у список послідовності кортежей з ключами і значеннями словника

Приклад 5.10 Генератор словника:

```
{key_expression : value_expression for expression in iterable}
```

```
1 word = 'letters'
2 letter_counts = {letter: word.count(letter) for letter in word}
3 letter_counts

{'l': 1, 'e': 2, 't': 2, 'r': 1, 's': 1}
```

Рисунок 5.18 – Генератор словника

За допомогою генераторів можна конструювати і списки. Такі визначення списків записуються зі застосуванням у квадратних дужках виразу та наступних за ним блоків `for` та `if`.

```
1 vec = [2, 4, 6]
2 vec_first = [3 * x for x in vec]
3 print(vec_first) # [6, 12, 18]
4 #створюємо новий список списків (з двох елментів)
5 vec_second = [[x, x ** 2] for x in vec]
6 print(vec_second) # [[2, 4], [4, 16], [6, 36]]
7
```

[6, 12, 18]
[[2, 4], [4, 16], [6, 36]]

Рисунок 5.19 – Генератор списків

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №5

В середовищі програмування Google Colab виконайте завдання відповідно варіанта.

- Для кожного завдання створюйте окрему кодову комірку. Формулювання завдань наводити в текстових комірках.
- Відповіді на контрольні запитання оформіть наприкінці блокноту в текстовому полі.
- Налаштуйте доступ для коментування.
- За результатами роботи оформіть текстовий звіт (шаблон міститься на сторінці курсу та додайте до цього документа посилання на ваш блокнот).
- Захистіть роботу.

Варіанти завдань до лабораторної роботи №5

Завдання 1. У прикладній лінгвістиці генерація випадкових фраз часто використовується для тестування синтаксичних аналізаторів або створення тренувальних корпусів. Розробіть програму, яка автоматично конструює речення, обираючи по одній лексемі з трьох тематичних списків (словників) – див. таблицю нижче.

Методичні рекомендації до виконання завдання:

- Для роботи з випадковими значеннями імпортуйте модуль `random`.
- Використовуючи функцію `random.choice()`, виберіть по одному випадковому слову з кожного списку.
- Поєднайте вибрані слова у цілісне речення. Результат має виводитися з великої літери та закінчуватися крапкою (використовуйте методи рядків та f-рядки).
- Програма повинна згенерувати та вивести на екран 3 різні випадкові фрази.

№	Підмети	Дієслова	Додатки
1	лінгвіст, студент, дослідник	аналізує, вивчає, описує	морфему, діалект, неологізм
2	алгоритм, парсер, скрипт	обробляє, сортує, індексує	конкорданс, словосполучення, метадані
3	нейромережа, система, модель	перекладає, адаптує, відтворює	контекст, ідіому, термінологію
4	синтаксист, аналізатор, програма	розбирає, будує, виокремлює	речення, словосполучення, предикат
5	семантик, тезаурус, словник	визначає, класифікує, пов'язує	значення, синонім, поняття
6	фонетист, мікрофон, процесор	реєструє, фіксує, розпізнає	звук, алофон, інтонацію

№	Підмети	Дієслова	Додатки
7	лексикограф, редактор, бот	укладає, редагує, додає	дефініцію, статтю, етимологію
8	автор, критик, рецензент	створює, цитує, інтерпретує	метафору, гіпертекст, наратив
9	морфолог, модуль, лемматизатор	виділяє, деривує, відмінює	корінь, суфікс, парадигму
10	аналітик, фільтр, код	вилучає, очищує, перевіряє	помилку, дублікат, стоп-слово
11	соціолінгвіст, носій, респондент	вживає, обирає, змінює	жаргон, соціолект, мовну норму
12	мозок, когнітивіст, пам'ять	сприймає, зберігає, розшифровує	стимул, асоціацію, мовленнєвий акт
13	чат-бот, трансформер, інтерфейс	генерує, прогнозує, ідентифікує	відповідь, намір користувача, сутність
14	користувач, сервіс, плагін	шукає, знаходить, відображає	інформацію, транскрипцію, переклад
15	графема, символ, шрифт	передає, відображає, кодує	звук, значення, повідомлення

Завдання 2. Візьміть текст, що містить фрагмент художнього твору. Розробіть алгоритм для обчислення обсягу тексту на рівні знаків:

1. Визначте загальну кількість символів у тексті з пробілами та пунктуацією та без цих символів (чистий обсяг значущих одиниць).

2. Визначте загальну кількість слів у тексті, загальну кількість різних слів (для цього використайте тип даних `set`) та кількість унікальних слів, що зустрічаються лише один раз (важливий показник індивідуального стилю автора).

3. Обчисліть відношення кількості різних слів до загальної кількості слів. Чим вищий цей показник, тим різноманітніша лексика автора.

Методичні рекомендації для виконання завдання:

- перед підрахунком слів приведіть текст до нижнього регістру (`.lower()`) та видаліть знаки пунктуації, щоб слова «Книга» та «книга» не сприймалися як різні лексеми.

- для розділення тексту на слова скористайтеся методом `.split()`.

- для підрахунку частотності кожного слова (щоб знайти ті, що зустрічаються один раз) використовуйте словник (`dict`), де ключем буде слово, а значенням – кількість його вживань.

Завдання 3. Виконайте наступне завдання відповідно до свого варіанта

1. Складіть список коренів слів, що характеризують колір; визначте кількість слів у тексті, що характеризують колір, а також те, який колір серед них переважає. Таких списків має бути декілька, відповідно до обраних кольорів.

2. Визначте процент води у тексті (це кількість «стоп-слів» поділена на загальну кількість слів). Стоп-слова – це слова, які ігноруються при індексації

сторінок пошуковими системами, не несуть смислового навантаження, замінюються маркерами і негативно впливають на якість текстів, знижуючи їх корисність. Списки стоп-слів див. в Інтернеті.

3. Складіть списки слів, що характеризують різні емоції – не менше 3 емоцій та не менше 10 слів у кожному списку. Підрахуйте частоту появи у тексті слів з кожного списку. Зробіть висновок про те, які емоції з досліджуваних переважають у тексті.

4. Складіть список слів, що характеризують романтичні почуття (не менше 30 слів у списку) та визначте кількість появи даних слів у тексті. Надрукуйте речення, в якому таких слів більше трьох.

5. Знайдіть у тексті найдовшу послідовність слів, що повторюється більше одного разу.

6. Порівняйте два тексти на наявність однакових послідовностей, що містять не менше 5 слів. (Знадобиться ще один текст, напр., того ж автора).

7. Знайдіть у тексті послідовності, що містять не менше 3 слів та повторюються не менше 5 раз, створіть список таких послідовностей та підрахуйте кількість їх повторів.

8. Підрахуйте загальну кількість речень у тексті, кількість окличних речень, кількість питальних речень, кількість речень, що закінчуються трикрапкою.

9. Складіть списки слів, що характеризують різні емоції – не менше 3 емоцій та не менше 10 слів у кожному списку. Підрахуйте частоту появи у тексті слів з кожного списку. Зробіть висновок про те, які емоції з досліджуваних переважають у тексті.

10. Визначте максимальну, мінімальну та середню довжину слів, речень у тексті.

11. Визначте частоту появи питальних речень у тексті та частоту появи слів в цих реченнях.

12. Визначте, які слова найчастіше зустрічаються поряд (перед або після) з вказаним користувачем словом.

13. Визначте, 10 найчастіших послідовностей з N слів у тексті. N вказується користувачем.

14. Визначте, з якого слова найчастіше починаються речення у тексті, а також яким найчастіше закінчуються.

15. Визначте частоту появи слів з різною довжиною.

Завдання 4. Підготуйте текст (мінімум 50 слів). Створіть два списки: перший – містить слова, які зустрічаються в тексті менше 5 разів, другий – кількість повторів даних слів. Створіть словник з цих двох списків. Використайте функцію **zip()**

Контрольні запитання

1. Поясніть переваги використання словника у порівнянні зі списком списків при розв'язанні задачі пошуку морфологічного тега для конкретної

лексеми. Чому доступ за ключем вважається ефективнішим для великих корпусів?

2. Як тип даних `set` допомагає автоматично обчислити обсяг унікального словникового запасу автора?

3. У чому полягає ключова відмінність між кортежем та списком в контексті цілісності даних?

4. Поясніть, чому конструкція `empty = {}` не створить порожню множину. Яку вбудовану функцію слід використати в цьому випадку?

5. Що таке гапакси?

6. Який об'єкт повертає метод `.keys()`? Чому для подальшого сортування слів за алфавітом цей об'єкт необхідно перетворювати у список за допомогою функції `list()`?

Корисні посилання

1. Офіційна документація Python: Структури даних. URL: <https://docs.python.org/3/tutorial/datastructures.html>

2. Natural Language Processing with Python (NLTK Book). URL: <https://www.nltk.org/book/>

3. Real Python: Словники в Python (Python Dicts). URL: <https://realpython.com/python-dicts/>

4. Programming Historian: Робота з текстом та словниками. URL: <https://programminghistorian.org/en/lessons/counting-frequencies>

5. Вступ до структур даних у Python з прикладами. URL: <https://ua.linkedin.com/pulse/introduction-data-structures-python-examples-shriyansh-tiwari-sivif?tl=uk>

Лабораторне заняття № 6. Автоматизація обробки корпусів: функції та файли

Мета: оволодіти навичками написання функцій та роботи з текстовими файлами в Python для вирішення типових задач прикладної лінгвістики; навчитися розробляти власні програмні модулі.

План

1. Функції.
2. Передавання у функцію змінної кількості параметрів.
3. Використання lambda-функцій для швидкої фільтрації та сортування текстів.
4. Опрацювання корпусних даних: файлове введення-виведення та розробка прикладних модулів.
5. Створення власного модуля.

6.1 Функції

Використання функцій у Python дозволяє автоматизувати типові задачі аналізу тексту. Створивши власну функцію один раз (наприклад, для розрахунку лексичного багатства), її можна викликати для будь-якої кількості нових корпусів. Це позбавляє необхідності дублювати код і робить лінгвістичні інструменти універсальними та придатними для спільного використання.

Функція – це програмний модуль, визначений через інструкцію `def`, що приймає вхідні дані (аргументи) та повертає результат їхньої обробки. У лінгвістичних дослідженнях функції дозволяють інкапсулювати складні операції – наприклад, алгоритми токенизації чи очищення тексту – забезпечуючи можливість багаторазового використання коду в різних проєктах та обміну інструментами між дослідниками.

Інструкція `def` створює об'єкт функції та зв'язує його з ім'ям. У загальному вигляді інструкція має такий формат:

```
def <name>(arg1, arg2, ..., argN) :  
    <statements>  
    return <value>
```

Рисунок 6.1 – Загальний синтаксис оголошення функції

Визначення функції за допомогою інструкції `def` складається з двох частин:

- Заголовок: містить **назву** функції та перелік **аргументів** (параметрів) у круглих дужках, що завершується двокрапкою.
- Тіло: блок коду з обов'язковими відступами, який реалізує алгоритм обробки даних. Цей код виконується лише в момент виклику функції.

- Функція може бути будь-якої складності і повертати будь-які об'єкти (списки, кортежі, і навіть функції).

```
1 def add(x, y):  
2     return x + y
```

Рисунок 6.2 – Оголошення функції додавання двох чисел

Щоб викликати цю функцію, необхідно ввести ім'я та дужки (див. рис. 6.3). Інструкція `return` каже, що потрібно повернути значення. У нашому випадку функція повертає суму `x` і `y`. Тепер ми її можемо викликати:

```
1 add (1, 10)  
  
11  
  
1 add ('abc', 'def')  
  
'abcdef'
```

Рисунок 6.3 – Виклик функції для обчислення суми двох чисел

На рисунку 6.4 наведено програмну реалізацію функції, яка обчислює коефіцієнт лексичного розмаїття TTR (Type-Token Ratio – відношення кількості унікальних слів до загальної кількості слів у тексті, показує, наскільки багатим є словниковий запас автора у конкретному уривку або наскільки часто в тексті повторюються одні й ті самі слова).

```
1 # оголошення функції  
2 def calculate_ttr(text):  
3     """  
4     Функція приймає рядок тексту і повертає коефіцієнт  
5     лексичного розмаїття (від 0 до 1).  
6     """  
7 # переводимо в нижній регістр і розбиваємо на слова  
8     tokens = text.lower().split()  
9  
10 # обчислюємо загальну кількість слів  
11     total_words = len(tokens)  
12  
13 # обчислюємо кількість унікальних слів за допомогою функції set  
14     unique_words = len(set(tokens))  
15  
16 # обчислюємо коефіцієнт  
17     ttr = unique_words / total_words  
18  
19     return ttr  
20  
21 # застосування функції  
22 sample_text = "Мова це код а код це мова"  
23 diversity_score = calculate_ttr(sample_text)  
24  
25 print(f"Текст для аналізу: '{sample_text}'")  
26 print(f"Коефіцієнт лексичного розмаїття: {diversity_score:.2f}")
```

Рисунок 6.4 – Програмна реалізація функції обчислення коефіцієнта лексичного розмаїття

В цій програмі ми даємо функції зрозумілу назву та вказуємо, що вона очікує на вході один параметр – `text`. Всередині функції відбувається очищення даних, використовуємо властивості функції `set()` для пошуку унікальних слів та проводимо математичне обчислення. Далі функція повертає результат програмі. Результат роботи функції зберігається у змінну `diversity_score` і використовувати його далі (наприклад, для порівняння двох авторів). Виклик функції здійснюється за ім'ям, передаючи їй реальний текст у дужках.

Вдалим стилем є документування кожної функції. Для цього в наступному рядку відразу після заголовка додано короткий опис функції, укладений у потрібні `"""` лапки (див. рис. 6.4).

Атрибут `__doc__` містить документацію функції.

Наприклад, увесь вміст усередині потрібних лапок виводиться інструкцією `print (ім'я_функції.__doc__)`. Такий спосіб дозволяє легко зрозуміти призначення функції, якщо прочитати початковий текст або скористатись спеціальним сервером документації Python (див. рис. 6.5).

\

```
1 print (calculate_ttr.__doc__)
```

Функція приймає рядок тексту і повертає коефіцієнт лексичного розмаїття (від 0 до 1).

Рисунок 6.6 – Результат документування функції

! Важливо. У прикладній лінгвістиці такий підхід дозволяє створити «бібліотеку» власних інструментів. Один раз написавши функцію для видалення стоп-слів або підрахунку середньої довжини речення, можна застосовувати її до багатьох різних текстових файлів.

6.2 Передавання у функцію змінної кількості параметрів

Високий ступінь адаптивності механізмів передачі аргументів є визначальною характеристикою мови Python, що забезпечує ефективне проектування уніфікованого програмного інструментарію для вирішення широкого спектру лінгвістичних задач

Типи аргументів у Python:

- **Позиційні аргументи:** значення копіюються у параметри функції згідно з їхньою позицією. Наприклад, у функції `analyze(text, lang)` першим завжди має йти текст (`text`), а другим – мова (`lang`).
- **Змінна кількість аргументів (*args):** якщо кількість вхідних даних наперед невідома (наприклад, при передаванні списку слів або декілька речень для аналізу), перед ім'ям параметра ставиться зірочка `*`. Це дозволяє функції приймати будь-яку кількість значень, які всередині неї об'єднуються в кортеж.

- Обов'язкові параметри: часто вказуються на початку списку аргументів. Програма не дозволить викликати функцію, поки не буде передано для них конкретні значення.

На рисунку 6.7 наведено програмну реалізацію функції, яка приймає обов'язковий тег (наприклад, «NOUN») і довільну кількість слів для перевірки:

```
1 def check_category(category, *words):
2     print(f"Категорія для перевірки: {category}")
3     print(f"Аналізуємо слова: {words}")
4
5 # виклик з різною кількістю слів
6 check_category("NOUN", "мова", "код", "лінгвістика")
7
8 check_category("NOUN", "наука", "математика")
```

```
Категорія для перевірки: NOUN
Аналізуємо слова: ('мова', 'код', 'лінгвістика')
Категорія для перевірки: NOUN
Аналізуємо слова: ('наука', 'математика')
```

Рисунок 6.7 – Функція зі змінної кількості параметрів

Використання `*args` дозволяє створювати гнучкі функції для обробки корпусів: можна передавати в одну і ту саму функцію як одне слово, так і цілий абзац, розбитий на токени, не змінюючи структуру самої функції.

6.3 Використання лямбда-функцій для швидкої фільтрації та сортування текстів

Анонімні функції – це компактні однорядкові структури, що складаються з ключового слова `lambda`, переліку аргументів та єдиного обчислювального виразу після двокрапки. На відміну від стандартних функцій (`def`), вони дозволяють лаконічно описувати операції трансформації лінгвістичних одиниць (наприклад, нормалізацію токенів чи фільтрацію стоп-слів) безпосередньо у місці їхнього виклику. Такий підхід забезпечує синтаксичну стислість коду та підвищує ефективність розробки інструментів для швидкої обробки текстових масивів.

Загальний синтаксис `lambda`-виразу:

`lambda` аргументи: вираз

Результатом `lambda`-виразу є такі ж об'єкти функцій, які утворюють інструкції `def`, але тут є кілька відмінностей:

- `lambda` – це вираз, а не інструкція;

- тіло `lambda` – це не блок інструкцій, а один вираз. Наприклад, функцію з рисунка 6.2 можна описати за допомогою `lambda`-виразу, явно присвоївши результат імені, яке пізніше буде використано для виклику функції:

```
1 sum = lambda x, y : x + y
2 sum (1, 10)
```

11

Рисунок 6.8 – `lambda`-функція

Таким чином, замість написання окремої функції, наприклад, для сортування слів за їхньою довжиною, можна використати лямбда-вираз: `sorted(words, key=lambda w: len(w))` – це дозволяє обробити корпус текстів одним рядком коду.

6.4 Опрацювання корпусних даних: файловий ввід-вивід та розробка прикладних модулів

Файл – це іменована структура для довготривалого зберігання інформації на зовнішніх носіях. У прикладній лінгвістиці текстові файли є базовим форматом для збереження корпусів та словників, оскільки вони підтримують послідовності символів довільної довжини.

Взаємодія з даними в Python реалізується через функцію `open()`, яка ініціалізує файловий об'єкт. Цей об'єкт виступає програмним інтерфейсом між скриптом та файловою системою комп'ютера. На відміну від стандартних структур даних (списків чи словників), файловий об'єкт не завантажує весь вміст у пам'ять автоматично, а надає спеціалізовані методи для керування потоками введення-виведення (зчитування, запису та позиціювання).

Загальний синтаксис функції `open()`:

```
my_file = open(ім'я_файлу [, режим_доступу] [, буферизация])
```

Таблиця 6.1 Режими відкриття файлів

Операція	Інтерпретація
<code>r</code>	Відкрити файл для читання (використовується за замовчуванням).
<code>w</code>	Відкрити файл для запису. Створює новий файл, якщо він не існує, або видаляє вміст файлу, якщо існує.
<code>x</code>	Відкрити файл для ексклюзивного створення. Якщо файл вже існує, операція завершується невдало.
<code>a</code>	Відкрити файл для додавання даних у кінець файлу без видалення поточного вмісту. Створює новий файл, якщо він не існує.
<code>t</code>	Відкрити файл в текстовому режимі (використовується за замовчуванням).
<code>b</code>	Відкрити файл у двійковому режимі.
<code>+</code>	Відкрити файл для оновлення (читання та запис).

Таблиця 6.2 Методи для роботи з файлами

Метод	Опис
<code>close()</code>	Закриває відкритий файл. Нічого не відбувається, якщо файл вже закритий.
<code>detach()</code>	Відокремлює базовий двійковий буфер від <code>TextIOBase</code> та повертає його.
<code>fileno()</code>	Повертає ціле число файлу (файловий дескриптор).

Метод	Опис
<code>flush()</code>	Очищає буфер запису у файловому потоці.
<code>isatty()</code>	Повертає <code>True</code> , якщо потік файлів є інтерактивним.
<code>read(n)</code>	Зчитує не більше <code>n</code> символів з файлу. Виконує читання до кінця файлу, якщо <code>n</code> є від'ємним або <code>None</code> .
<code>readable()</code>	Повертає <code>True</code> , якщо з файлового потоку можна зчитати дані.
<code>readline(n=-1)</code>	Зчитує та повертає один рядок з файлу. Зчитує не більше <code>n</code> байтів, якщо вказано.
<code>readlines(n=-1)</code>	Зчитує та повертає список рядків з файлу. Зчитує не більше <code>n</code> байтів/символів, якщо вказано.
<code>seekable()</code>	Повертає <code>True</code> , якщо файловий потік підтримує довільний доступ.
<code>tell()</code>	Повертає ціле число, що становить поточну позицію файлового об'єкта.
<code>truncate(size=None)</code>	Змінює розмір файлового потоку до <code>size</code> байтів. Якщо розмір не вказано, розмір змінюється щодо поточного розташування.
<code>writable()</code>	Повертає <code>True</code> , якщо у файловий потік можна здійснити запис.
<code>write(s)</code>	Записує рядок (<code>s</code>) у файл та повертає кількість записаних символів.
<code>writelines(lines)</code>	Записує список рядків (<code>lines</code>) у файл.

Для опрацювання файлів в середовищі Google Colab необхідно здійснити такі кроки:

1. Приєднати Google-диск до середовища Colab. Це необхідно зробити для того, щоб була можливість створити, змінити або зчитати вміст файлу.

```
from google.colab import drive
drive.mount('/content/drive')
```

Цей процес може зайняти певний час оскільки вимагатимуться додаткові користувацькі дозволи. Після того, як у рядку для виведення з'явиться «Mounted at /content/drive» – можна створювати, відкривати та зберігати файли.

2. Для того, щоб мати доступ до вже створеного файлу, наприклад, через функцію `open()`, в параметрах даної функції необхідно вказати повний шлях до файлу. Отримати інформацію про повний шлях до файлу можна так: на бічній панелі натиснути кнопку із позначкою тека (див. рис. 6.9).

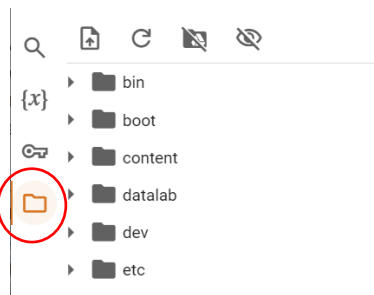


Рисунок 6.9 – Робоча директорія середовища програмування

Потім, відкрити теку `/content/drive/MyDrive`. Далі шукаємо потрібний файл. Потім викликаємо контекстне меню на шуканому файлі та обираємо пункт меню «Скопіювати шлях». Отриманий рядок необхідно буде скопіювати як параметр у функцію `open()`

3. Для відкриття файлу та виведення його вмісту на екран:

```
file_1 = open("/content/drive/MyDrive/Colab Notebooks/sample_file.txt", "r+")
read_content = file_1.read()
print(read_content)
```

4. Для закриття файлу:

```
file_1.close()
```

5. Для відкриття файлу для дозапису:

```
file_1 = open("/content/drive/MyDrive/Colab Notebooks/sample_file.txt", "a+")
file_1.write("\nThird record")
file_1.close()
```

6. Створення текстового файлу з назвою `sample1_file.txt`

```
! touch "/content/drive/MyDrive/Colab Notebooks/sample1_file.txt"
```

Зверніть увагу, що знак оклику – обов'язковий елемент синтаксису

6.5 Створення власного модуля

Для створення та наступної інтеграції модуля з власними функціями до середовища розробки Google Colab необхідно виконати такий алгоритм:

1. Створіть текстовий файл, наприклад в Notepad.
2. Скопіюйте в цей файл розроблені вами функції (бажано до кожної функції додати анотацію)

3. Збережіть цей файл з розширенням `.py`

4. Створіть на своєму Google-диску окрему теку, для власного модуля. Завантажте туди файл отриманий в пункті 3.

5. В цю ж саму папку додайте порожній файл з назвою `__init__.py` (Python до 3.3 має потребу в ньому, щоб розглядати каталог, що містить його, як пакет).

6. В рядку для введення команд надрукуйте:

```
import sys
sys.path.append('/content/drive/MyDrive/Colab Notebooks/python_module')
```

Це необхідно для того, щоб «повідомити» середовищу програмування, де саме знаходиться власний модуль із власними функціями. В даному прикладі `python_module` – це тека на диску, де міститься файл з власними функціями

7. Імпортуйте модуль із власними функціями:

```
import my_module
```

8. Можна викликати функції з даного модуля, аналогічно з вбудованими модулями, наприклад, `random`, `nltk` тощо.

```
1 import my_module
```

```
1 my_module.hello()
```

```
Hello from my module
```

Рисунок 6.10 – Імпорт власного модуля та подальший виклик функції

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №6

В середовищі програмування Google Colab виконайте завдання відповідно варіанта.

- Для кожного завдання створюйте окрему кодову комірку. Формулювання завдань наводити в текстових комірках.
- Відповіді на контрольні запитання оформіть наприкінці блокноту в текстовому полі.
- Налаштуйте доступ для коментування.
- За результатами роботи оформіть текстовий звіт (шаблон міститься на сторінці курсу та додайте до цього документа посилання на ваш блокнот).
- Захистіть роботу.

Варіанти завдань до лабораторної роботи №6

Завдання 1. Розробити функцію відповідно варіанта завдання. Розроблена функція повинна задовольняти таким умовам:

- Назва функції має бути змістовною (англійською мовою), наприклад `calculate_ttr` або `find_longest_word`.
- Всередині функції обов'язково додайте `docstring` з описом того, які параметри функція приймає та що вона робить.
- Перед обробкою функція повинна приводити текст до нижнього регістру та (за потреби) видаляти зайві пробіли.
- Після оголошення функції наведіть приклад її виклику з двома різними варіантами вхідних даних.

1. Напишіть функцію, яка обчислює середню кількість символів у словах заданого тексту (без урахування пробілів).

2. Створіть функцію для розрахунку Type-Token Ratio (коефіцієнт лексичного розмаїття TTR – відношення кількості унікальних слів до загальної кількості слів у тексті, показує, наскільки багатим є словниковий запас автора у конкретному уривку або наскільки часто в тексті повторюються одні й ті самі слова.).

3. Розробіть функцію, яка обчислює відсоток розділових знаків відносно загальної кількості символів у тексті.

4. Напишіть функцію, яка приймає текст і повертає кількість «гапаксів» – слів, які зустрічаються в цьому фрагменті лише один раз.

5. Створіть функцію, яка підраховує кількість слів, що складаються лише з великих літер (довжиною більше 1 символу).

6. Напишіть функцію, яка підраховує, скільки слів у списку починаються на заданий префікс (наприклад, «не-», «пре-»).

7. Аналіз суфіксів: Розробіть функцію, яка знаходить та повертає список усіх слів, що закінчуються на заданий суфікс (наприклад, «-сть», «-ий»).

8. Вокалізація тексту: Створіть функцію, яка обчислює «вокальність» тексту – відношення кількості голосних літер до загальної кількості літер.

9. Фільтр стоп-слів: Напишіть функцію, яка приймає рядок і список стоп-слів, повертаючи текст без цих слів.

10. Довжина речень: Розробіть функцію, яка приймає текст і повертає середню кількість слів у реченнях (розділювачі: ., !, ?).

11. Найдовша лексема: Створіть функцію, яка знаходить найдовше слово у тексті та повертає його довжину.

12. Частота власних назв: Напишіть функцію, яка рахує кількість слів, що починаються з великої літери (ігноруючи перше слово в реченні).

13. Символьна щільність: Розробіть функцію, яка обчислює, який відсоток у тексті займає конкретний символ (наприклад, літера «о» або пробіл).

14. Короткі слова: Створіть функцію, яка вилучає з тексту всі слова, довжина яких менша за N символів (N – параметр функції).

15. Пошук повторів: Напишіть функцію, яка перевіряє, чи є в реченні слова, що повторюються підряд (лексичний повтор), і повертає їх кількість.

Завдання 2. Створіть модуль з розробленою вами функцією. Реалізуйте перегляд вмісту модуля, імпортування модуля з використанням псевдоніма, виведення анотації до функції.

Завдання 3. Створіть файл, в якому міститься невеликий фрагмент художнього твору. Виконайте з ним такі завдання:

- Програмно організуйте відкриття, читання, та виведення на екран вмісту даного файлу.

- Створіть список слів з даного файлу та запишіть його в інший файл. В отриманому файлі не має бути жодних повторів слів (однаковими вважаються слова з повним співпадінням всіх символів).

Контрольні запитання

1. У яких лінгвістичних задачах доцільніше використовувати анонімні функції `lambda` замість стандартного оголошення через `def`? Наведіть приклад.

2. Поясніть роль інструкції `return` у функціях. Чим відрізняється повернення результату від його виведення на екран через `print()`?

3. Для чого у функціях обробки текстів використовується оператор `*args`?

4. Чому при роботі з великими текстовими корпусами стандартним підходом є використання менеджера контексту `with open(...)`? Які ризики виникають при використанні звичайного методу `.close()`?

5. Поясніть різницю між режимами `'w'` (запис) та `'a'` (дозапис) при формуванні результатів аналізу. У якому випадку використання режиму `'w'` може призвести до втрати лінгвістичних даних?

6. Які переваги надає винесення власних лінгвістичних функцій в окремий файл-модуль? Як це сприяє повторному використанню коду в інших дослідженнях?

Корисні посилання

1. Офіційна документація Python: Визначення функцій. URL: <https://docs.python.org/3/tutorial/controlflow.html#defining-functions>
2. NLTK Book: Модульне програмування та обробка текстів. URL: <https://www.nltk.org/book/ch04.html>
3. Real Python: Читання та запис файлів у Python. URL: <https://realpython.com/read-write-files-python/>
4. GitHub: Python for Linguists (Dirk Hovy). URL: https://github.com/dirkhovy/python_for_linguists/tree/master

Лабораторне заняття № 7. Попередня обробка тексту

Мета: опанувати повний цикл препроцесингу тексту для підготовки даних до автоматизованого лінгвістичного аналізу.

План

1. Перетворення до нижнього регістру.
2. Видалення пунктуації.
3. Видалення стоп-слів.
4. Заміна скорочень.
5. Виправлення потенційних помилок.
6. Токенізація.
7. Стемінг або лематизація.
8. Дослідження статистичних характеристик даних

Процес попереднього опрацювання (препроцесингу) лінгвістичних даних у задачах автоматичного аналізу природної мови (NLP) передбачає реалізацію таких базових етапів:

- Перетворення до нижнього регістру.
- Видалення пунктуації.
- Видалення стоп слів.
- Заміна скорочень.
- Виправлення потенційних помилок.
- Токенізація.
- Стемінг або лематизація.
- Дослідження статистичних характеристик даних.

7.1 Перетворення до нижнього регістру

Перетворення тексту до нижнього регістру – це фундаментальний етап препроцесингу, який полягає в уніфікації написання всіх символів корпусу. У прикладній лінгвістиці цей процес є критично важливим для зменшення розрідженості даних та коректного підрахунку частотності лексем.

Без нормалізації регістру комп'ютер сприйматиме слова «Мова», «мова» та «МОВА» як три різні лексичні одиниці. Приведення до нижнього регістру дозволяє об'єднати всі словоформи в один токен, що забезпечує точність статистичного аналізу (наприклад, при розрахунку індексу TTR (Type-Token Ratio, коефіцієнт лексичного розмаїття або побудові частотних словників).

У Python цей етап реалізується за допомогою методу рядків `.lower()`. У межах функцій обробки тексту цей метод зазвичай застосовується на самому початку, ще до етапу токенізації (див. рис. 7.1).

```

1 text = "Прикладна лінгвістика забезпечує Автоматизацію."
2 normalized_text = text.lower()
3 print(normalized_text)

```

прикладна лінгвістика забезпечує автоматизацію.

Рисунок 7.1 – Нормалізація регістру

Хоча нормалізація регістру є стандартом, слід враховувати, що в окремих задачах (наприклад, при розпізнаванні власних назв або аналізі тональності) велика літера може нести додаткове семантичне навантаження, яке буде втрачено після перетворення.

7.2 Видалення пунктуації

Видалення пунктуації – це процес очищення корпусу від знаків припинення, дужок, тире та інших неалфавітних символів. У контексті попередньої обробки тексту цей етап класифікується як «усунення шуму», оскільки пунктуація зазвичай не несе самостійного лексичного значення, але значно ускладнює автоматизований аналіз.

Головна мета цього етапу – забезпечити точність токенізації. Більшість алгоритмів розбивають текст за пробілами, тому без попереднього видалення пунктуації слово в кінці речення (наприклад, «код.») буде розпізнане як окремий токен, відмінний від слова всередині речення («код»). Очищення дозволяє звести всі вживання лексеми до єдиної форми, що є критичним для розрахунку частотності та побудови n-грам.

Найбільш ефективним способом у Python є використання вбудованої бібліотеки `string` та методу `.translate()`, який дозволяє видалити всі стандартні знаки пунктуації за один прохід (див. рис. 7.2). Також часто застосовуються генератори списків для фільтрації символів.

```

1 import string
2
3 # вхідний текст
4 raw_text = "Прикладна лінгвістика: це Автоматизація, чи не так?"
5
6 # перетворення до нижнього регістру
7 # використовуємо вбудований метод .lower()
8 lower_text = raw_text.lower()
9
10 # видалення пунктуації
11 # створюємо таблицю перекладу, де кожен знак із string.punctuation замінюється на None
12 # string.punctuation містить рядок: !"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
13 clean_text = lower_text.translate(str.maketrans('', '', string.punctuation))
14
15 print(f"Оригінальний текст: {raw_text}")
16 print(f"Очищений текст: {clean_text}")

```

Оригінальний текст: Прикладна лінгвістика: це Автоматизація, чи не так?
Очищений текст: прикладна лінгвістика це автоматизація чи не так

Рисунок 7.2 – Видалення пунктуації з тексту

В цій програмній реалізації використано стандартний метод рядків `.lower()`, який перетворює всі великі літери на малі. Готовий набір символів `string.punctuation`, які вважаються пунктуацією в Python і до нього входять не лише крапки та коми, а й технічні символи, як-от: `#` чи `$`. Також використано метод `str.maketrans('', '', ...)` для того, щоб знайти всі символи з цього списку і видалити їх. Це працює значно швидше, ніж видалення кожного знака окремим циклом.

У специфічних лінгвістичних задачах, таких як аналіз синтаксису, визначення авторства або аналіз емоційного забарвлення, пунктуація може бути важливою ознакою. У таких випадках цей етап попередньої обробки пропускають або модифікують.

7.3 Видалення стоп-слів

Видалення стоп-слів – це етап очищення тексту від найбільш частотних слів, які зазвичай виконують службову роль і не несуть самостійного семантичного навантаження. До цієї категорії традиційно належать сполучники, прийменники, частки та деякі займенники.

У задачах автоматичної обробки мови (наприклад, при тематичному моделюванні або пошуку ключових слів) такі слова, як «і», «в», «на», «що», створюють «інформаційний шум». Оскільки вони зустрічаються майже в кожному реченні, їхня присутність заважає алгоритмам ідентифікувати специфічні терміни та значущі поняття, які визначають зміст конкретного тексту. Видалення стоп-слів дозволяє зосередити аналіз на повнозначних частинах мови.

Для реалізації цього етапу в Python зазвичай використовують заздалегідь підготовлені списки (словники) стоп-слів. Процес полягає у порівнянні кожного токена тексту зі списком заборонених слів та його вилученні у разі збігу (див. рис 7.3). Найбільш ефективним інструментом для цього є генератори списків.

```
1 # Початковий набір токенів (вже нормалізований)
2 tokens = ["прикладну", "лінгвістику", "вивчають", "у", "університеті", "та", "на", "курсах"]
3
4 # Базовий список стоп-слів для української мови
5 stop_words = ["у", "та", "на", "і", "а", "з", "до"]
6
7 # Фільтрація: залишаємо лише ті слова, яких немає в списку stop_words
8 filtered_tokens = [word for word in tokens if word not in stop_words]
9
10 print(f"До фільтрації: {tokens}")
11 print(f"Після фільтрації: {filtered_tokens}")
```

До фільтрації: ['прикладну', 'лінгвістику', 'вивчають', 'у', 'університеті', 'та', 'на', 'курсах']
Після фільтрації: ['прикладну', 'лінгвістику', 'вивчають', 'університеті', 'курсах']

Рисунок 7.3 – Видалення стоп-слів

У реальних проєктах списки стоп-слів не створюють вручну, а імпортують із професійних лінгвістичних бібліотек, таких як NLTK, spaCy або Stanza. Зазначимо, що немає універсального списку стоп-слів. Наприклад, слово «що» може бути стоп-словом для класифікації новин, але критично важливим для аналізу синтаксичних зв'язків. Також і текст, і список стоп-слів мають бути

зведені до однакового регістру (зазвичай нижнього), інакше фільтрація не спрацює (слово «У» не буде видалене списком, де є тільки «у»).

Для прикладної лінгвістики в Україні найкращим стандартом зараз є бібліотека spaCy, яка має повноцінну модель для української мови з розширеним списком стоп-слів та кращою обробкою граматики (див. рис. 7.4). Цей інструмент реально використовується в сучасній ІТ-індустрії для обробки української мови (наприклад, у чат-ботах чи системах аналізу відгуків).

```
1 !pip install spacy
2 !python -m spacy download uk_core_news_sm

1 import spacy
2
3 # завантажуюмо українську модель
4 nlp = spacy.load("uk_core_news_sm")
5
6 # отримуємо список стоп-слів безпосередньо з моделі
7 ua_stop_words = nlp.Defaults.stop_words
8
9 text = "Прикладна лінгвістика допомагає нам аналізувати тексти."
10
11 # токенизація за допомогою spaCy
12 doc = nlp(text.lower())
13
14 # фільтрація: залишаємо лише ті слова, які не є стоп-словами та не є пунктуацією
15 filtered_tokens = [token.text for token in doc if not token.is_stop and not token.is_punct]
16
17 print(f"Кількість стоп-слів у spaCy: {len(ua_stop_words)}")
18 print(f"Результат: {filtered_tokens}")
```

Кількість стоп-слів у spaCy: 467
Результат: ['прикладна', 'лінгвістика', 'допомагає', 'аналізувати', 'тексти']

Рисунок 7.5 – Видалення стоп-слів засобами бібліотеки spaCy

7.4 Заміна скорочень

Заміна скорочень – це етап препроцесингу, що полягає у приведенні скорочених форм слів та сталих виразів до їхнього повного лексичного вигляду. У прикладній лінгвістиці цей процес є необхідним для усунення неоднозначності та забезпечення цілісності лінгвістичного аналізу.

Використання скорочень (наприклад, «і т.д.», «т.зв.», «ім.») створює перешкоди для автоматизованих систем аналізу. По-перше, крапки у скороченнях можуть помилково сприйматися алгоритмами як маркери кінця речення. По-друге, розширення скорочень дозволяє токенизаторам правильно ідентифікувати слова, що підвищує якість роботи векторних моделей та систем машинного перекладу.

Найчастіше заміна реалізується за допомогою словника відповідностей та бібліотеки регулярних виразів `re`. Це дозволяє знаходити скорочення навіть за наявності специфічних символів або різного написання (див. рис. 7.6).

```

1 import re
2
3 # Текст із скороченнями
4 text = "Прикладна лінгвістика вивчає мову, код і т.д. в т.ч. автоматизацію."
5
6 # Словник скорочень: ключ – регулярний вираз, значення – повна форма
7 abbreviations = {
8     r"і т\.\.д\.\.": "і так далі",
9     r"в т\.\.ч\.\.": "в тому числі",
10    r"т\.\.зв\.\.": "так званий"
11 }
12
13 # Процес заміни
14 normalized_text = text
15 for pattern, replacement in abbreviations.items():
16     normalized_text = re.sub(pattern, replacement, normalized_text)
17
18 print(f"Оригінал: {text}")
19 print(f"Результат: {normalized_text}")

```

Оригінал: Прикладна лінгвістика вивчає мову, код і т.д. в т.ч. автоматизацію.

Результат: Прикладна лінгвістика вивчає мову, код і так далі в тому числі автоматизацію.

Рисунок 7.6 – Заміна скорочень в тексті

В регулярних виразах використання символу \ перед крапкою в шаблоні (\.) є обов'язковим, оскільки в регулярних виразах крапка означає «будь-який символ». Списки скорочень мають адаптуватися під конкретний тип тексту. Наприклад, у юридичних текстах скорочення будуть відрізнятися від тих, що вживаються в наукових статтях з філології. Слід бути обережними з омонімічними скороченнями, де одна і та сама скорочена форма може мати різні значення залежно від контексту.

7.5 Виправлення потенційних помилок

Виправлення помилок – це етап препроцесингу, спрямований на ідентифікацію та корекцію друкарських помилок (одруківок), орфографічних дефектів або артефактів, що виникли під час оптичного розпізнавання тексту. У прикладній лінгвістиці цей крок є критичним для забезпечення чистоти статистичних даних.

Наявність помилок у корпусі призводить до штучного збільшення кількості унікальних токенів. Наприклад, слово «лінгвістика», написане з помилкою як «лінгвістіка», буде розпізнане програмою як окрема лексема. Це спотворює результати частотного аналізу, розрахунку лексичної щільності та заважає коректній лематизації.

Для автоматичного виправлення помилок ми скористаємося інструментом, що базується на правилах та словниках української мови (див. рис. 7.7).

```
1 !pip install language-tool-python

1 import language_tool_python
2
3 # ініціалізація інструменту для української мови ('uk-UA')
4 tool = language_tool_python.LanguageTool('uk-UA')
5
6 # текст із типовими помилками: "лінгвістика" (орфографія), "програмі" (узгодження)
7 text = "Прикладна лінгвістика допомагає створювати корисні програмі."
8
9 # пошук помилок
10 matches = tool.check(text)
11
12 # виведемо інформацію про знайдені проблеми
13 for match in matches:
14     print(f"Знайдено: '{match.sentence}'")
15     print(f"Помилка: {match.message}")
16     print(f"Варіанти виправлення: {match.replacements}")
17     print("-" * 10)
18
19 # автоматичне застосування найкращих пропозицій
20 corrected_text = tool.correct(text)
21
22 print(f"Оригінальний текст: {text}")
23 print(f"Виправлений текст: {corrected_text}")
```

Downloading LanguageTool 6.8-SNAPSHOT: 100%|██████████| 259M/259M [00:11<00:00, 22.8MB/s]
Знайдено: 'Прикладна лінгвістика допомагає створювати корисні програмі.'
Помилка: Знайдено потенційну орфографічну помилку.
Варіанти виправлення: ['лінгвістика', 'лінгвістка']

Оригінальний текст: Прикладна лінгвістика допомагає створювати корисні програмі.
Виправлений текст: Прикладна лінгвістика допомагає створювати корисні програмі.

Рисунок 7.7 – Автоматичне виправлення помилок

На відміну від простих списків слів, цей інструмент розрізняє зв'язки між словами. Він може підказати, що в реченні «корисні програмі» закінчення не узгоджується за числом чи відмінком. Використання `uk-UA` гарантує, що перевірка йде згідно з чинним правописом.

7.6 Токенізація

Токенізація – це процес розбиття безперервного потоку тексту на окремі змістовні одиниці, які називаються токенами. Залежно від рівня аналізу, токенами можуть бути речення, слова, морфеми або навіть окремі символи.

Токенізація є початковим етапом до будь-якого комп'ютерного аналізу мови. Перш ніж обчислювати статистику чи визначати частини мови, програма повинна чітко розрізняти межі кожної лексеми. Головна складність цього етапу полягає в неоднозначності розділювачів: наприклад, крапка може означати кінець речення, бути частиною скорочення (і т.д.) або входити до складу числа.

Найпростіший спосіб токенизації в Python – метод `.split()`, який ділить рядок за пробілами. Також часто використовують бібліотеку `spaCy`, оскільки вона вміє відокремлювати знаки пунктуації від слів (наприклад, перетворювати «мова!» на два токени: «мова» та «!»), що критично для подальшої обробки (див. рис. 7.8). В лабораторній роботі №3 (див. рис. 3.10) розглядався приклад токенизації засобами бібліотеки `nlTK`

```

1 import spacy
2
3 # завантажуюмо модель
4 nlp = spacy.load("uk_core_news_sm")
5
6 text = "Токенізація – це перший крок. Хіба ні?"
7
8 # створення об'єкта Doc (автоматична токенизація)
9 doc = nlp(text)
10
11 # виведення токенів - використано генератор списку
12 tokens = [token.text for token in doc]
13
14 print(f"Оригінал: {text}")
15 print(f"Список токенів: {tokens}")

```

Оригінал: Токенізація – це перший крок. Хіба ні?
Список токенів: ['Токенізація', '-', 'це', 'перший', 'крок', '.', 'Хіба', 'ні', '?']

Рисунок 7.8 – Токенізація тексту

7.7 Стемінг та лематизація

Для того, щоб комп'ютер міг ефективно аналізувати текст, різні словоформи одного й того самого слова (наприклад, «мова», «мови», «мовою») мають бути зведені до спільного знаменника. Це завдання вирішується двома способами: стемінгом або лематизацією.

Стемінг – це процес відсікання афіксів (закінчень та суфіксів) для отримання основи слова (стема). Це грубий евристичний процес, який не враховує контекст і граматичні правила. Результат часто не є реальним словом (наприклад, слово «читати» після стемінгу може перетворитися на «чит»). Використовується в пошукових системах, де швидкість обробки важливіша за лінгвістичну точність.

Лематизація – це приведення слова до його канонічної, словникової форми, яка називається лемою (наприклад, для іменників – називний відмінок однини, для дієслів – інфінітив). На відміну від стемінгу, лематизація використовує морфологічний аналіз та словники, що дозволяє точно визначити основу слова з урахуванням його частини мови.

Для української мови, стемінг працює погано через багатство форм. Тому в NLP-проектах для української мови майже завжди використовують лематизацію (див. рис. 7.9).

```

1 import spacy
2
3 # завантаження моделі
4 nlp = spacy.load("uk_core_news_sm")
5
6 text = "Студенти вивчають прикладну лінгвістику в університетах."
7 doc = nlp(text.lower())
8
9 # отримання лем для кожного токена
10 lemmas = [token.lemma_ for token in doc if not token.is_punct]
11
12 print(f"Оригінал: {text}")
13 print(f"Лематизований текст: {lemmas}")

```

Оригінал: Студенти вивчають прикладну лінгвістику в університетах.

Лематизований текст: ['студент', 'вивчати', 'прикладний', 'лінгвістика', 'в', 'університет']

Рисунок 7.9 – Лематизація тексту засобами бібліотеки SpaCy

7.8 Дослідження статистичних характеристик

Дослідження статистичних характеристик – це етап кількісного аналізу тексту, на якому дослідник отримує числові показники структури корпусу. Після очищення, токенізації та лематизації дані стають придатними для розрахунку метрик, що дозволяють об'єктивно порівнювати різні тексти або автори.

Основні показники в прикладній лінгвістиці:

1. **Частотний розподіл** – підрахунок кількості вживань кожного слова. Це дозволяє виділити найбільш значущі (ключові) слова тексту та побудувати частотний словник.

2. **Коефіцієнт лексичного розмаїття TTR** (Type-Token Ratio – відношення кількості унікальних слів до загальної кількості слів у тексті). Показує, наскільки багатим є словниковий запас автора у конкретному уривку або наскільки часто в тексті повторюються одні й ті самі слова. Чим вищий цей показник, тим багатша лексика автора.

3. **Середня довжина слова та речення** – метрики, що допомагають визначити складність тексту або ідентифікувати авторський стиль (стилометрія).

Для базового статистичного аналізу в Python скористаємось вбудованим класом Counter з модуля collections.

```

1 from collections import Counter
2
3 # припустимо, ми вже маємо список лематизованих токенів
4 tokens = ["мова", "код", "лінгвістика", "мова", "автоматизація", "код", "мова"]
5
6 # підрахунок частотності
7 frequency = Counter(tokens)
8
9 # розрахунок лексичної різноманітності тексту TTR
10 unique_words = len(frequency.keys())
11 total_words = len(tokens)
12 ttr = unique_words / total_words
13
14 print(f"Частотність: {frequency.most_common(3)}")
15 print(f"Загальна кількість токенів: {total_words}")
16 print(f"Кількість унікальних слів: {unique_words}")
17 print(f"Коефіцієнт лексичного розмаїття (TTR): {ttr:.2f}")

```

Частотність: [('мова', 3), ('код', 2), ('лінгвістика', 1)]
 Загальна кількість токенів: 7
 Кількість унікальних слів: 4
 Коефіцієнт лексичного розмаїття (TTR): 0.57

Рисунок 7.10 – Обчислення статистичних характеристик тексту

Статистичні характеристики дозволяють перейти від суб'єктивного сприйняття тексту до доказового аналізу. Наприклад, за допомогою цих метрик можна:

- Автоматично визначити жанр тексту (наприклад, наукові тексти мають інший TTR, ніж художні).
- Виявити плагіат або встановити авторство анонімного документа.
- Оцінити складність навчальних матеріалів для студентів відповідного рівня.

7.9 Візуалізація результатів: хмара слів

Хмара слів – це візуальне представлення частотності лексем у тексті, де розмір кожного слова прямо пропорційний числу його вживань. Це дозволяє досліднику миттєво ідентифікувати головні теми та ключові поняття корпусу без детального вивчення таблиць із цифрами.

Для лінгвіста хмара слів є інструментом швидкої діагностики контенту. Вона допомагає порівняти дискурси різних авторів, виявити домінантні концепти в політичних промовах або проаналізувати медійні наративи.

Для створення хмари слів у Python використовується бібліотека `wordcloud` у поєднанні з бібліотекою для побудови графіків `matplotlib` (див. рис. 7.11).

! Важливо. Перед побудовою хмари обов'язково потрібно пройти етапи видалення стоп-слів та лематизації, інакше центр хмари займуть службові слова (сполучники та прийменники), які не несуть семантичного навантаження.

```
1 !pip install wordcloud matplotlib
```

```
1 from wordcloud import WordCloud
2 import matplotlib.pyplot as plt
3
4 # Приклад обробленого (лематизованого) тексту без стоп-слів
5 processed_text = "мова код лінгвістика програмування автоматизація мова "\
6                 "алгоритм код мова аналіз мова"
7
8 # створення об'єкта WordCloud
9 wordcloud = WordCloud(
10     width = 800,
11     height = 400,
12     background_color = 'white',
13     colormap = 'viridis', # Колірна схема
14     min_font_size = 10
15 ).generate(processed_text)
16
17 # візуалізація через matplotlib
18 plt.figure(figsize = (10, 5), facecolor = None)
19 plt.imshow(wordcloud)
20 plt.axis("off") # вимикаємо осі координат
21 plt.tight_layout(pad = 0)
22
23 plt.show()
```



Рисунок 7.11 – Візуалізація хмари слів

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №7

В середовищі програмування Google Colab виконайте завдання відповідно варіанта.

- Для кожного завдання створюйте окрему кодову комірку. Формулювання завдань наводити в текстових комірках.
- Відповіді на контрольні запитання оформіть наприкінці блокноту в текстовому полі.
- Налаштуйте доступ для коментування.
- За результатами роботи оформіть текстовий звіт (шаблон міститься на сторінці курсу та додайте до цього документа посилання на ваш блокнот).
- Захистіть роботу.

Варіанти завдань до лабораторної роботи №7

Завдання 1. Загальна інструкція для всіх варіантів. Для кожного завдання необхідно:

- Завантажити відповідний текст (корпус).
- Виконати нормалізацію (нижній регістр).
- Очистити текст від пунктуації та специфічних символів.
- Розширити скорочення, притаманні обраному стилю.
- Провести токенізацію та видалити стоп-слова.
- Виконати лематизацію (або стемінг).
- Розрахувати статистику (TTR, частотний словник, середня довжина речення).

1. Обробити тексти передвиборчих програм. Звернути увагу на розширення аббревіатур політичних партій та видалення загальномовних стоп-слів. Порівняти лексичне багатство (TTR) різних програм.

2. Обробити корпус анотацій до статей з комп'ютерної лінгвістики. Замінити латинські скорочення (e.g., i.e., et al.) та виділити ключові терміни через частотний аналіз лем.

3. Очистити корпус постів від хештегів, посилань та емодзі. Виконати виправлення типових одруків та нормалізувати скорочення (наприклад, «плз» → «будь ласка»).

4. Обробити тексти законів. Реалізувати розширення складних юридичних скорочень («ст.», «пп.», «ККУ»). Дослідити середню довжину речень у юридичному стилі.

5. На прикладі казок порівняти результати стемінгу та лематизації. Розрахувати індекс лексичної різноманітності для текстів різних вікових категорій.

6. Обробити файли README програмних проєктів. Видалити фрагменти коду (як шум) та залишити лише текстовий опис. Провести лематизацію термінології.

7. Обробити стрічку новин за добу. Виконати токенизацію та визначити найчастотніші власні назви (топоніми та антропоніми) після видалення стоп-слів.

8. Обробити деідентифіковані виписки. Виконати заміну медичних аббревіатур та виправлення потенційних помилок у термінах за допомогою відстані Левенштейна.

9. Обробити збірку прислів'їв та приказок. Видалити застарілі частки (як стоп-слова) та знайти «гапакси» (слова, що зустрічаються один раз) через частотний словник.

10. Обробити корпус відгуків інтернет-магазину. Виконати детекцію та виправлення помилок, нормалізувати реєстр та побудувати хмару слів для позитивних і негативних відгуків окремо.

11. Обробити текст кіносценарію. Відфільтрувати імена героїв та ремарки (в дужках). Проаналізувати словниковий запас головних персонажів через TTR.

12. Обробити логи запитів користувачів. Виконати токенизацію на рівні символьних n -грам для обробки сленгу та скорочень. Видалити питальні частки як стоп-слова.

13. Обробити базу рецептів. Замінити скорочення одиниць вимірювання («кг», «ст. л.», «шт.»). Виділити список дієслів-інструкцій через лематизацію та POS-тегування.

14. Обробити архівні статті початку XX століття. Виконати нормалізацію правопису (заміна застарілих літер/форм на сучасні) та порівняти статистику з сучасними текстами.

15. Обробити тексти вакансій для розробників. Виділити назви технологій (як ключові токени), видалити шаблонні фрази (стоп-фрази) та розрахувати щільність термінології.

Контрольні запитання

1 Як перетворення тексту до нижнього реєстру впливає на кількість «типів» та «токенів» у корпусі?

2 У чому полягає принципова різниця між стемінгом та лематизацією?

3 Що таке «інформаційний шум» у контексті NLP?

4 Які складнощі виникають при токенизації текстів, що містять скорочення (наприклад, «т.зв.», «ім.», «вул.»)? Як попередня заміна скорочень допомагає уникнути помилок при сегментації речень?

5 Чому порядок етапів попередньої обробки є критично важливим? Обґрунтуйте, чому видалення пунктуації та стоп-слів зазвичай передують етапу лематизації, а не навпаки.

Корисні посилання

1. Офіційна документація spaCy: Моделі для української мови. URL: <https://spacy.io/models/uk>

2. NLTK Book: Processing Raw Text (Розділ 3). URL :
<https://www.nltk.org/book/ch03.html>
3. Real Python: Natural Language Processing With spaCy URL:
<https://realpython.com/natural-language-processing-spacy-python/>
4. Репозиторій Stopwords ISO: Українська мова. URL:
<https://github.com/stopwords-iso/stopwords-uk>
5. Документація модуля Python 're' (Регулярні вирази). URL:
<https://docs.python.org/3/library/re.html>

Рекомендована література

1. Васильєв О. М, Пилипенко А. І.. Основи програмування на Python : підручник. Київ : Ліра К, 2025. 194 с.
2. Висоцька В. А., Оборська О. В. Python: алгоритмізація та програмування. Київ : Новий світ 2000, 2026. 516
3. Зеленьак О. П. Програмування мовою Python. Алгоритмічні структури і стратегії. Теорія та практика. Київ : Ліра К, 2025. 338 с.
4. Когут І. В. Ментинський С. М. Базовий курс програмування на Python. Львів : Магнолія 2006, 2026. 230 с.
5. Лутц М. Python. Довідник програміста. Київ : Науковий світ, 2024. 294 с.
6. Офіційна документація Python. URL : <https://docs.python.org/uk/3.10/library/functions.html#slice>
7. Підручник з Python. URL : <https://docs.python.org/uk/3/tutorial/index.html>
8. Початкові відомості з Colab. URL : https://colab.research.google.com/notebooks/welcome.ipynb?hl=uk#scrollTo=GJBs_flRovLc
9. Путівник мовою програмування Python. URL : <https://pythonguide.rozh2sch.org.ua/>
10. Стиль коду Python. URL : <https://peps.python.org/pep-0008/>
11. Філософія Python. URL : <https://peps.python.org/pep-0020/>
12. Documentation. Natural Language Toolkit. URL : <https://www.nltk.org/>
13. Industrial-Strength Natural Language Processing in Python. URL : <https://spacy.io/>
14. Python Tutorial. URL : <https://www.w3schools.com/python/default.asp>
15. Yuhang G., Mao Y., Gao Y. Text-to-Pipeline: Bridging Natural Language and Data Preparation Pipelines. 10.48550/arXiv.2505.15874., 2025.

Використана література

1. Васильєв О. М., Пилипенко А.І.. Основи програмування на Python : підручник. Київ : Ліра К, 2025. 194 с.
2. Висоцька В. А., Оборська О. В. Python: алгоритмізація та програмування. Київ : Новий світ 2000, 2026. 516
3. Зеленьак О. П. Програмування мовою Python. Алгоритмічні структури і стратегії. Теорія та практика. Київ : Ліра К, 2025. 338 с.
4. Когут І. В. Ментинський С. М. Базовий курс програмування на Python. Львів : Магнолія 2006, 2026. 230 с.
5. Лутц М. Python. Довідник програміста. Київ : Науковий світ, 2024. 294 с.
6. Підручник з Python. URL : <https://docs.python.org/uk/3/tutorial/index.html>
7. Путівник мовою програмування Python. URL : <https://pythonguide.rozh2sch.org.ua/>
8. Yuhang G., Mao Y., Gao Y. Text-to-Pipeline: Bridging Natural Language and Data Preparation Pipelines. 10.48550/arXiv.2505.15874., 2025.

Навчально-методичне видання
(українською мовою)

Кривохата Анастасія Григорівна
Кудін Олексій Володимирович

МОВА ПРОГРАМУВАННЯ PYTHON

Методичні рекомендації до лабораторних занять
для здобувачів ступеня вищої освіти бакалавра
спеціальності «Філологія»
спеціалізації «Прикладна лінгвістика»
освітньо-професійної програми «Прикладна лінгвістика»

Рецензент *О. С. Пшенична*
Відповідальний за випуск *А. О. Лісняк*
Коректор *А. Г. Кривохата*