



**Міністерство освіти і науки України
Одеський національний морський університет
Кафедра кібербезпеки та захисту інформації**

**МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни «Програмна обробка та аналіз даних»
другого (магістерський) рівня вищої освіти
за спеціальністю F5 Кібербезпека та захист інформації
галузі знань F Інформаційні технології
Розділ 1. «Базові структури та формати даних»**

Одеса, 2025



Міністерство освіти і науки України
Одеський національний морський університет
Кафедра кібербезпеки та захисту інформації

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни «Програмна обробка та аналіз даних»
другого (магістерський) рівня вищої освіти
за спеціальністю F5 Кібербезпека та захист інформації
галузі знань F Інформаційні технології
Розділ 1. «Базові структури та формати даних»

Затверджено
на засіданні кафедри КЗІ
Протокол № 2 від 06.10.2025 р.

Одеса, 2025

Методичні вказівки до лабораторних робіт з дисципліни «**Програмна обробка та аналіз даних**»
Розділ 1. «Базові структури та формати даних» для другого (магістерський) рівня вищої освіти за спеціальністю F5 Кібербезпека та захист інформації галузі знань F Інформаційні технології / Укл.: *Лебедєва О.Ю.* – Одеса, 2025. – 19 с.

Укладач: **Лебедєва О.Ю.**, к.т.н., доц.

ЗМІСТ

Змістовий модуль 1. «Базові структури та формати даних»	5
Лабораторна робота 1. Формати даних	5
1. Теоретичний матеріал	5
2. Завдання до лабораторної роботи 1	9
Лабораторна робота 2 Структури даних. Списки	11
1. Теоретичний матеріал	11
2. Завдання до лабораторної роботи 2	15
Лабораторна робота 3. Структури даних. Зв'язний список	16
1. Теоретичний матеріал	16
Завдання до лабораторної роботи 3	17
Література.....	19

Змістовий модуль 1. «Базові структури та формати даних»

Лабораторна робота 1. Формати даних

Мета роботи: ознайомитися з форматами даних, такими як текстові файли, CSV, JSON, XML. Навчитися працювати з форматами даних, такими як текстові файли, CSV, JSON, XML на мові Python.

1. Теоретичний матеріал

У Python операції з файлами виконуються так:

- відкрити файл;
- читання чи запис;
- закрити файл.

Натомість, щоб створити об'єкт файлу, необхідно викликати вбудовану функцію `open`, надаючи їй ім'я зовнішнього файлу та рядок режиму доступу до файлу.

Режимі відкриття файлів:

- `r` – відкрити файл для читання. (використовується за замовчуванням)
- `w` – відкрити файл для запису. Створює новий файл, якщо він не існує, або видаляє вміст, якщо він існує.
- `x` – відкрити файл для ексклюзивного створення. Якщо файл існує, операція завершиться невдало.
- `a` – відкрити файл для додавання даних до кінця файлу без видалення поточного вмісту. Створює новий файл, якщо він не існує.
- `t` – відкрити файл у текстовому режимі. (використовується за замовчуванням)
- `b` – відкрити файл у двійковому режимі.
- `+` – відкрити файл для оновлення (читання та запис).

Щоб прочитати те, що було записано у файл, його потрібно відкрити в режимі `r` (цей режим використовується за замовчуванням, якщо рядок режиму у виклику функції опущений).

Python для читання файлів використовується метод `read()`.

Наприклад:

```
1 file1 = open("intro.txt", "r")
2 read_content = file1.read()
3 print(read_content)
```

Розглянемо приклад, в якому зчитуємо файл по рядкам. Програма виводить перші чотири рядка з файлу:

```
1 file1 = open("intro.txt", "r")
2
3 count = 0
4 for line in file1:
5     print(line.strip())
6     if count == 3:
7         break
8     count += 1
9 file1.close()
```

Коли закінчили виконувати операції з файлом, нам потрібно його правильно закрити. Закриття файлу звільняє ресурси, задіяні для роботи з ним. Це робиться за допомогою методу `close()`.

У Python можна використовувати синтаксис `with...open` для автоматичного закриття

файлу.

Наприклад:

```
1 with open("intro.txt", "r") as file1:
2     count = 0
3     for line in file1:
4         print(line.strip())
5         if count == 3:
6             break
7         count += 1
```

При записі у файл необхідно пам'ятати дві речі:

- Якщо ми намагаємося відкрити неіснуючий файл, створюється новий файл.
- Якщо файл вже існує, його вміст видаляється, а до файлу додається новий вміст.

Для того, щоб записати дані у файл у Python, нам потрібно відкрити його в режимі запису, вказавши "w" у функції open() як другий аргумент.

Наприклад:

```
1 with open('test.txt', 'w') as file2:
2     file2.write('Programming is Fun. ')
3     file2.write('Python for beginners')
4
5 with open('test.txt') as file1:
6     print(file1.read())
```

```
D:\WorkProjects\Python\PythonProject\.venv\
```

```
Programming is Fun. Python for beginners
```

CSV (від англ. comma-separated values, "значення, розділені комою") – простий текстовий відкритий формат, призначений для представлення табличних даних.

Найпростіший CSV-файл виглядає так:

```
name,category,price
Samsung A5,Smartphones,15000
iPhone 5S,Smartphones,14000
HP Laser Jet,Printers,12000
iMac 21.5,MonoBlocks,25000
```

У цьому прикладі перший рядок містить заголовки стовпців, а наступні рядки є даними.

Файл у форматі CSV є звичайним текстовим документом, в якому дані організовані у вигляді рядків і стовпців. Кожен рядок відповідає одному запису, а значення всередині рядка поділяються спеціальними символами – роздільниками.

У Python для роботи з CSV-файлами найчастіше використовують стандартний модуль csv.

Модуль csv реалізує класи для читання та запису табличних даних у форматі CSV. Об'єкти reader і writer модуля csv читають і записують послідовності. Також можна читати та записувати дані у формі словника за допомогою класів DictReader і DictWriter.

Наприклад, маємо CSV-файл (output.csv) такої структури:

Name, Age, City
Alice, 30, New York
Bob, 25, Los Angeles
Charlie, 35, Chicago

Читання інформації з CSV-файлу:

```
1 import csv
2
3 # Open the CSV file
4 with open('output.csv', mode='r') as file:
5     reader = csv.reader(file)
6
7     # Iterate over each row in the CSV
8     for row in reader:
9         print(row)
```

```
D:\WorkProjects\Python\PythonP
['Name', 'Age', 'City']
['Alice', '30', 'New York']
['Bob', '25', 'Los Angeles']
['Charlie', '35', 'Chicago']
```

Читання CSV у вигляді словника (DictReader):

```
1 import csv
2
3 # Open the CSV file
4 with open('output.csv', mode='r') as file:
5     reader = csv.DictReader(file)
6     for row in reader:
7         print(row["Name"], "->", row["Age"], "->", row["City"])
```

```
D:\WorkProjects\Python\Pyth
Alice -> 30 -> New York
Bob -> 25 -> Los Angeles
Charlie -> 35 -> Chicago
```

JSON (JavaScript Object Notation) — це полегшений формат даних, який використовується для обміну даними між сервером і клієнтом. Він зазвичай використовується у веб-додатках для надсилання та отримання даних.

JSON — це текстовий формат, який легко читати та писати як для людей, так і для машин. Він складається з пар ключ-значення.

Наприклад:

```
{
  "name": "Alice",
  "age": 30,
  "city": "New York",
  "is_student": false,
  "skills": ["Python", "JavaScript", "SQL"]
}
```

Python має вбудований модуль `json`, який дозволяє легко працювати з JSON-даними.

Щоб прочитати дані JSON із файлу, використовуйте метод `json.load()`. Цей метод читає вміст файлу та перетворює його на словник Python.

Наприклад, маємо такий файл `example.json`:

```
{
  "students": [
    {
      "id": 1,
      "name": "Ivan",
      "age": 20
    },
    {
      "id": 2,
      "name": "Olena",
      "age": 19
    },
    {
      "id": 3,
      "name": "Petro",
      "age": 21
    },
    {
      "id": 4,
      "name": "Maria",
      "age": 18
    }
  ]
}
```

Відкриваємо файл і читаємо дані з нього:

```
1 import json
2
3 # Example of reading JSON from a file
4 with open('example.json', 'r', encoding='utf-8') as file:
5     data = json.load(file)
6
7 for student in data["students"]:
8     print(student["name"], student["age"])
```

```
D:\WorkProjec
Ivan 20
Olena 19
Petro 21
Maria 18
```

Extensible Markup Language(XML) є мовою розмітки так само як HTML, або SGML. XML рекомендується W3C(World Wide Web Consortium) і доступний в якості відкритого стандарту.

XML являє собою мову з відкритим вихідним кодом, що дозволяє програмістам розробляти програми, які можуть бути прочитані іншими додатками, незалежно, від ОС чи мови програмування.

XML є надзвичайно корисною для малих та середніх обсягів даних, без використання SQL.

У Python для роботи з файлами XML найчастіше використовують стандартні модулі `xml.etree.ElementTree` і `minidom`, а також зовнішню бібліотеку `lxml` (для більш складних

завдань).

Наприклад маємо файл students.xml наступної структури:

```
<students>
  <student id="1">
    <name>Ivan</name>
    <age>20</age>
    <grade>A</grade>
  </student>
  <student id="2">
    <name>Olena</name>
    <age>19</age>
    <grade>B</grade>
  </student>
</students>
```

Зчитаємо інформацію з файлу та виведемо на екран:

```
1 import xml.etree.ElementTree as ET
2
3 tree = ET.parse("students.xml")
4 root = tree.getroot()
5
6 for student in root.findall("student"):
7     name = student.find("name").text
8     age = student.find("age").text
9     grade = student.find("grade").text
10    print(f"{name} ({age} років) -> {grade}")
```

```
D:\WorkProjects\Python\F
Ivan (20 років) -> A
Olena (19 років) -> B
```

2. Завдання до лабораторної роботи 1

Написати програми на Python для виконання наступних завдань.

1. Створити текстовий файл, в якому буде зберігатися набір з десяти чисел. Кожне число записано з нового рядку. Вивести дані з файлу та розрахувати суму та добуток цих чисел.

2. Створити файл students.csv, який містить інформацію про студентів у форматі: Name, Age, Grade, Group. Додати в файл не менше десяти значень. Виконати наступні завдання:

- Зчитати дані з файлу students.csv.
- Вивести на екран імена всіх студентів.
- Знайти середній вік студентів.
- Вибрати всіх студентів, які мають оцінку A і вивести їх імена.

3. Створити файл library.json, який містить інформацію про книжки в бібліотеці з таким ключами: isbn, title, author, year. Додати в файл не менше десяти значень. Виконати наступні завдання:

- Зчитати дані з файлу library.json.
- Вивести на екран назви всіх книжок.
- Вивести назву та автора за певний рік. Рік за яким шукаємо вводиться користувач.
- Вивести книжки одного автора.

4. Створити файл store.xml, який містить інформацію про продукти в інтернет

магазині з таким тегам product який описується: name, price, category, in_stock. Додати в файл не менше десяти продуктів. Виконати наступні завдання:

- Зчитати дані з файлу store.xml.
- Вивести на екран назви всіх продуктів та їх ціну.
- Вивести на екран всі продукт для яких in_stock = true

Лабораторна робота 2 Структури даних. Списки

Мета роботи: ознайомитися з часто використовуваними структурами даних, такими як списки, словники. Навчитися працювати зі структурами даних, такими як списки, словники на мові Python.

1. Теоретичний матеріал

У Python є безліч комплексних типів даних, які називаються послідовностями.

Список – один із найчастіше використовуваних і універсальних типів даних. Це аналог масивів за іншими мовами програмування.

Списки в Python використовуються для одночасного зберігання багатьох даних. Список створюється шляхом розміщення елементів всередині квадратних дужок [], розділених комами.

Список може містити будь-яку кількість елементів, і вони можуть бути різних типів (int, float, string тощо).

Елементом списку може бути інший список. Такі списки називаються *вкладеними*.

В Python кожен елемент списку асоціюється з індексом. Можна отримати доступ до елементів масиву, використовуючи номер індексу (0, 1, 2,...). Наприклад:

```
1 my_list = ["mouse", [8, 4, 6], ['a'], -5]
2
3 print(my_list[1])           [8, 4, 6]
4 print(my_list[1][0])       8
5 print(my_list[3])          -5
```

Python дозволяє використовувати від'ємний індекс для своїх послідовностей. Індекс -1 відноситься до останнього елемента, -2 відноситься до передостаннього елемента і так далі.

В Python за допомогою оператора «:» можна отримати доступ відразу до групи елементів (а не тільки до одного). При виконанні зрізу в списках перший індекс є включаючим, а кінцевий – виключаючим.

Наприклад:

```
1 my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'e', 'r']
2
3 print(my_list[2:4])
4 print(my_list[5:])
5 print(my_list[:])
6 print(my_list[:-5])
7 print(my_list[-3:-1])
```

```
['o', 'g']
['a', 'm', 'm', 'e', 'r']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'e', 'r']
['p', 'r', 'o', 'g', 'r']
['m', 'e']
```

Ключове слово `in` використовується для перевірки того, чи існує елемент у списку, чи ні.

Методи:

– `list.append(x)` – додає елемент `x` в кінець списку `list`;

- `list.extend(L)` – розширює список `list`, додаючи у кінець всі елементи списку `L`;
- `list.insert(i, x)` – вставляє у список `list` на `i`-ий елемент значення `x`;
- `len(list)` – використовується для визначення кількості елементів у списку.
- `list.remove(x)` – видаляє перший елемент у списку `list`, який має значення `x`; `ValueError`, якщо такого елемента не існує;
- `list.pop(i)` – видаляє із списку `list` `i`-ий елемент і повертає його; якщо індекс не вказано, видаляється останній елемент;
- `list.count(x)` – повертає кількість елементів із списку `list` зі значенням `x`;
- `list.index(x, start, end)` – повертає положення у списку `list` першого елемента зі значенням `x` (при цьому пошук ведеться від `start` до `end`);
- `list.sort(reverse=False)` – сортує елементи списку `list`; якщо необов'язковий параметр `reverse` має значення `True`, список сортується в зворотному порядку;
- `list.reverse()` – розвертає список `list`;
- `list.copy()` – поверхнева копія списку `list`;
- `list.clear()` – очищає список `list`;
- оператор `del(list)` – для видалення одного або кількох елементів зі списку.

Подання списків – елегантний та лаконічний спосіб створити новий список із існуючого. Подання списків складається з виразу, за яким слідує оператор `for`, і ця конструкція полягає у квадратних дужках `[]`.

У поданнях списків може бути більше операторів `for` та `if`. Оператор `if` дозволяє фільтрувати елементи для нового списку. Він необов'язковий.

Наприклад:

```
1 pow1 = [x for x in range(10)]
2 print(pow1)
3
4 pow2 = [2 ** x for x in range(6)]
5 print(pow2)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 4, 8, 16, 32]
```

Наприклад:

```
1 odd = [x for x in range(20) if x % 2 == 1]
2 print(odd)
3
4 str1 = [y+x for x in ['Python ', 'C '] for y in ['Язык ', 'Программирование на ']]
5 print(str1)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
['Язык Python ', 'Программирование на Python ', 'Язык C ', 'Программирование на C ']
```

Словник у Python – це впорядкована послідовність елементів. Кожен елемент словника має пару «ключ: значення». Словники використовуються для отримання значень, за його ключем.

Словник – це колекція, яка є впорядкованою, змінною та не допускає дублікатів. Починаючи з версії Python 3.7, словники *впорядковані*. У Python 3.6 і раніших версіях словники *не впорядковані*.

Коли ми говоримо, що словники впорядковані, це означає, що елементи мають визначений порядок, і цей порядок не зміниться. Невпорядкований означає, що елементи не мають визначеного порядку, ви не можете посилатися на елемент за допомогою індексу.

Словники можна змінювати, тобто ми можемо змінювати, додавати або видаляти елементи після створення словника.

В якості ключів не можна використовувати змінювані типи об'єктів. Наприклад, списки, словники та інші змінювані типи не можуть використовуватись в якості ключів. Однак, в якості значень ці змінювані типи можна використовувати.

Значення в елементах словника можуть мати будь-який тип даних: string, int, boolean, list та інші.

Переваги застосування словників:

- з допомогою словників можна розробляти ефективні структури даних;
- для словників не потрібно писати вручну алгоритми пошуку даних, оскільки ці операції вже є реалізовані;
- словники можуть містити об'єднані дані у вигляді записів;
- словники ефективні при представленні розріджених структур даних.

У Python створення словника можна виконати кількома способами.

1. Використання фігурних дужок {}. Наприклад:

```
dict = {1: 'Яблуко', 2: 'Груша'}

dics = {'name': 'John', 1: [2, 3, 4, 5]}
```

2. Використання функції dict(). Наприклад:

```
my_dict = dict(name="Anna", age=25, city="Kyiv")

dict = dict({1: 'Яблуко', 2: 'Груша'})
```

3. Створення порожнього словника. Наприклад:

```
empty_dict = {}

empty_dict = dict()
```

У мові Python існує велика кількість операцій та методів для обробки словників:

- стандартний метод len() – визначає кількість елементів у списку;
- операція D[key] – доступ до елемента словника D за значенням ключа key;
- операція del – видалення елемента за ключем;
- операції in, not in – визначення наявності чи відсутності ключа у словнику;
- метод iter() – отримати ітератор за ключами словника;
- метод clear() – видалити всі елементи зі словника;
- метод copy() – вернути копію словника;
- метод dict.fromkeys() – створити новий словник з ключів та значень;
- метод get() – отримати значення за ключем;
- метод items() – вернути представлення елементів словника;
- метод keys() – отримати нове представлення ключів словника;
- метод pop() – видалення елементів зі словника з поверненням значення;
- метод popitem() – витягування зі словника довільної пари ключ:значення;
- метод setdefault() – встановити елемент за замовчуванням;
- метод update() – оновлення словника за заданим списком пар ключ:значення;
- метод values() – отримати список значень зі словника.

Для доступу до елементів словника можна скористатися квадратними дужками [], або

використовувати метод `get()`. Якщо ми скористаємося квадратними дужками, то якщо ключ не знайдений, ми отримаємо помилку `KeyError`. Якщо ми скористаємося `get()`, і за умови, що ключ не буде знайдений, функція поверне нам `None`.

```
dict = {'name': 'Тимур', 'age': 30}

print(dict['name'])
print(dict.get('age'))
```

За допомогою оператора присвоєння ми можемо додавати нові елементи, а також змінювати значення існуючих. Якщо ключ існує, то оновлюється його значення, якщо він відсутній, то до словника додається пара (ключ: значення).

```
dict = {'name': 'John', 'age': 30}

dict['age'] = 31
print(dict)

dict['adress'] = 'Odesa'
print(dict)
```

```
{'name': 'John', 'age': 31}
{'name': 'John', 'age': 31, 'adress': 'Odesa'}
```

За допомогою методу `pop()`, можемо видалити конкретний ключ, і він поверне значення цього ключа. Метод `popitem()` видаляє випадкову пару ключ: значення, і так само поверне випадкову пару ключ: значення. Весь словник можна очистити за допомогою `clear()`. Для видалення окремих елементів можна скористатися ключовим словом `del`.

```
diction = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

print(diction.pop(4))
print(diction)

print(diction.popitem())
print(diction)

del diction[1]
print(diction)
```

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{2: 4, 3: 9}
```

Перевіряємо наявність елемента за допомогою ключового слова `in`.

```
dictionary = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

print(1 in dictionary)
print(2 in dictionary)
print(7 not in dictionary)
```

```
True
False
False
```

Наприклад, вивести всі значення словника:

```
dictionary = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

for i in dictionary:
    print(dictionary[i])
```

```
1
9
25
49
81
```

2. Завдання до лабораторної роботи 2

Написати програму на Python для виконання наступного завдання.

1. Створити текстовий файл, в якому буде зберігатися набір з чисел от 0 до 100 включно. Зчитати дані з файлу у список та виконати завдання. Список із чисел (0 – 100) це список оцінок студентів. Створити словник, де ключ — оцінка ЄКТС (A, B, C, D, E, FX, F), значення — кількість студентів із цією оцінкою.

Оцінка ЄКТС	Бальна оцінка
A	90–100
B	82–89
C	75–81
D	67–74
E	60–66
FX	35–59
F	0–34

- Знайти оцінку з максимальною кількістю студентів;
- Знайти оцінку з мінімальною (не нульовою) кількістю студентів;

Для виконання завдань не використовувати модулі зі статистики та аналізу даних.

2. Створити файл csv, який містить інформацію про складання іспиту студентами у форматі: Student, Gender, Grade. Створити словник: ключ – стать, значення – список оцінок. Обчисли середній бал для кожної статі, максимальну та мінімальну оцінку.

Для виконання завдань не використовувати модулі зі статистики та аналізу даних.

Лабораторна робота 3. Структури даних. Зв'язний список

Мета роботи: ознайомитися зі структурою зв'язний список. Навчитися створювати та працювати зі зв'язним списком на мові Python.

1. Теоретичний матеріал

Зв'язний список являє собою структуру даних, яка вільна від обмежень у масивах.

Зв'язний список не зберігає свої елементи в послідовних комітках пам'яті, і користувач може як додати будь-яку кількість нових елементів до нього в довільний момент, так і видалити елементи, які стали непотрібними зі списку без необхідності виділення нових зв'язних областей в "купі" і перезапису списку на нове місце.

Проте, на відміну від масиву, зв'язний список не допускає прямого (випадкового) доступу до даних. Елементи в зв'язному списку можуть бути доступні тільки в послідовному порядку. Але на відміну від масиву, вставка і видалення елемента списку може бути виконана в будь-який момент в реальному часі.

Зв'язаний список, являє собою лінійну сукупність елементів даних, які називаються *вузлами* і так званий *дескриптор* (або покажчик на початок) списку.

Кожен з вузлів містить покажчик (адресу) наступного вузла в списку. Останній вузол у списку містить NULL-покажчик, який і сигналізує про кінець списку.

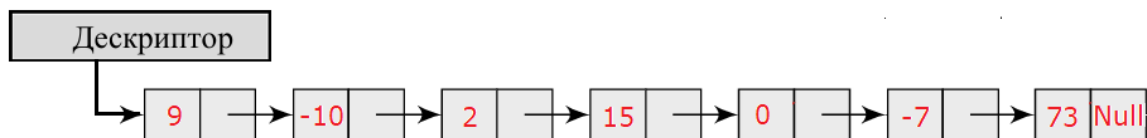


Рисунок 3.1 – Приклад зв'язного списку

Фізично список реалізується як сукупність дескриптора списку і елементів, розміщених в оперативній пам'яті або на зовнішніх запам'ятовуючих пристроях (ЗЗП) і зв'язаних один з одним у ланцюжок за допомогою покажчиків.

Дескриптор зазвичай реалізується у вигляді структури і може містити наступну інформацію про список:

- тип структури;
- назву списку;
- покажчик (адреса) початку списку;
- поточне число елементів;
- опис елемента (довжина, тип, тощо).

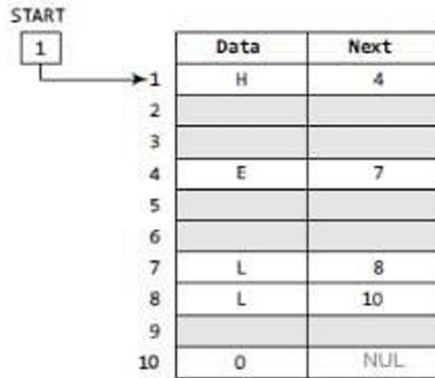
Зв'язаний список є абстрактною структурою даних, яка в свою чергу, часто використовується для реалізації інших, більш складних абстрактних структур даних. Таким чином, він виступає в якості будівельного блоку для реалізації структур даних, таких як стеки, черги і їх варіації.

Зв'язний список може сприйматися як поїзд або послідовність вузлів, в яких кожен вузол містить один або кілька полів даних і покажчик на наступний вузол. Ліва частина вузла, яка і містить дані, може включати в себе простий тип даних, масив або структуру. Права частина вузла містить покажчик на наступний вузол (або адресу наступного вузла в послідовності). Останній вузол не матиме за собою ніякого наступного вузла, так що він буде зберігати спеціальне значення, зване NULL.

Зв'язні списки містять дескриптор, який зберігає адресу першого вузла в списку. Ми можемо обійти весь список за допомогою дескриптора, який містить адресу першого вузла; адресна частина першого вузла в свою чергу, зберігає адресу наступного за ним вузла. Використовуючи цю техніку, окремі вузли списку утворюють ланцюжок вузлів

Якщо дескриптор в якості адреси першого вузла містить NULL, то зв'язний список порожній і не містить вузлів.

Наприклад:



Дескриптор START вказує перший елемент зв'язного списку в пам'яті.

В прикладі $START = 1$, так що перший вузол списку зберігається за адресою 1. Перший вузол містить дані $data = 'H'$, які є символ 'H'. Відповідний $next$ зберігає адресу наступного вузла, що дорівнює 4. Таким чином, ми будемо бачити адресу 4, щоб витягти наступний елемент даних, тощо. Ми повторюємо цю процедуру до тих пір, поки не досягнемо позиції, де наступний вузол в полі $next$ містити NULL, так як це означає кінець зв'язного списку.

Зверніть увагу, що в прикладі показується блок пам'яті, з адресами від 1 до 10. Заштрихована частина містить дані інших застосувань.

Вузли зв'язного списку не обов'язково повинні бути в зв'язаних комірках пам'яті. У нашому прикладі вузли для зв'язного списку, зберігаються за адресами 1, 4, 7, 8 і 10.

Односпрямований список є найпростішим типом зв'язного списку, в якому кожен вузол містить деякі дані і покажчик на наступний вузол того ж списку. Говорячи про те, що вузол містить покажчик на наступний, ми маємо на увазі, що вузол зберігає адресу наступного вузла в послідовності. Односпрямований список дозволяє виконувати обхід даних тільки в один бік.

Зв'язний список складається з вузлів, які розділені на дві частини: інформаційну і адресну.

Таким чином, пошук призначений для виявлення того, чи присутнє в інформаційній частині вузлів списку задане значення, чи ні. Якщо воно присутнє, то алгоритм повертає адресу вузла, який містить таке значення.

Завдання до лабораторної роботи 3

Написати програму на Python для роботи зі односпрямованим зв'язним списком. Передбачити в програмі наступний функціонал:

- Створити зв'язний список;
- Вивести зв'язний список;
- Додавання та видалення елементів зі списку;
- Пошук елемента у списку.

Література

1. Крєневич А.П. Алгоритми і структури даних. Підручник. – К.: ВПЦ «Київський університет», 2021. – 200 с. <https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf>
4. Barnes G. Data Structures and Algorithms: Annotated Reference with Examples [Електроний ресурс] Режим доступу: <https://mta.ca/~rrosebru/oldcourse/263114/Dsa.pdf>
5. Buki. (n.d.). Python у науці про дані: Як використовувати Python для аналізу даних та машинного навчання. Retrieved September 15, 2025, <https://buki.com.ua/blogs/python-u-nauci-pro-dani-iak-vikoristovuvati-python-dlia-analizu-danix-ta-masinnogo-navcannia>
6. Bethke Uli CSV vs JSON vs XML – The Best Comparison Guide 2026 [Електроний ресурс] Режим доступу: <https://sonra.io/csv-vs-json-vs-xml/>