Міністерство освіти та науки України
Національний університет водного господарства та природокористування
Кафедра комп'ютерних наук та прикладної математики

**04-01-60М**

**МЕТОДИЧНІ ВКАЗІВКИ**
до виконання лабораторних робіт
з навчальної дисципліни

# «Сучасні та спеціалізовані мови програмування»

для здобувачів вищої освіти першого (бакалаврського) рівня
за освітньо-професійною програмою «Прикладна математика»
спеціальності 113 «Прикладна математика»
денної та заочної форм навчання
Частина 2

Рекомендовано науково-методичною
радою з якості ННІ АКОТ
Протокол № 1 від 11.11.2021 р.

Рівне – 2021

Методичні вказівки до виконання лабораторних робіт з навчальної дисципліни «Сучасні та спеціалізовані мови програмування» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Прикладна математика» спеціальності 113 «Прикладна математика» денної та заочної форм навчання. Частина 2 [Електронне видання] / Грицюк І. М. – Рівне: НУВГП, 2021. – 38 с.

Укладач:
Грицюк І. М., асистент кафедри комп'ютерних наук та прикладної математики.

Відповідальний за випуск:
Турбал Ю. В., д.т.н., професор, завідувач кафедри комп'ютерних наук та прикладної математики

Керівник групи забезпечення спеціальності 113 «Прикладна математика»: О. В. Прищепа, к.ф.-м.н., доцент кафедри комп'ютерних наук та прикладної математики.

**Зміст**

Лабораторна робота №5

# Операції з рядками. Основні методи рядків. Форматування рядків. Регулярні вирази

## 5.1. Мета роботи

Вивчити теоретичні відомості, що до рядків. Познайомитися з роботою рядків та їх основними функціями. Навчитися використовувати рядки в роботі та форматування їх. Навчитися використовувати регулярні вирази.

## 5.2. Теоретичні відомості

### 5.2.1. Загальні відомості

*Рядки* – впорядковані послідовності символів, що використовуються для зберігання і представлення текстової інформації, тому за допомогою рядків можна працювати з усім, що може бути представлено в текстовій формі.

*Керуючі символи або екрановані послідовності* – послідовність символів, що починаються зі зворотного слеша і використовуються для подання інших символів. Екрановані послідовності найчастіше використовуються для включень в рядки спеціальних символів, що не мають стандартного односимвольного друкованого представлення.

Для розділення тексту на рядки можна використовувати поєднання трьох подвійних лапок, що дозволить записувати текст в одному операторі та переносити його в звичайному режимі за допомогою переносу. Розглянемо на прикладі.

*Наприклад:*

```
print("""Привіт,
мене
звати
Грицюк
Іванна
Михайлівна""")
```

*Результат:*

```
Привіт,
мене
звати
Грицюк
Іванна
Михайлівна
```

Але для більш швидкого та компактного вигляду коду використовуються керуючі символи.

Найчастіше вживаним є \n, що слугує для переходу на новий рядок. Також часто використовується табуляція для вирівнювання тексту за допомогою відступів. Для виділення тексту за допомогою одинарних або подвійних лапок використовується \. Для виведення в тексті зворотного слешу використовується його подвоєння.

| **Послідовність** | **Представлений символ** |
|---|---|
| \' | Одинарна лапка |
| \" | Подвійна лапка |
| \\ | Зворотний слеш |
| \a | Звуковий сигнал |
| \b | Backspace |
| \n | Новий рядок |
| \r | Повернення курсора (не одне й теж, що \n) |
| \t | Табуляція |
| \v | Вертикальна табуляція |

*Наприклад:*

```
print("Hello\nHello\nHello")
```

*Результат:*

```
Hello
Hello
Hello
```

*Наприклад:*

```
print("Hello\tHello\tHello")
```

*Результат:*

```
Hello   Hello   Hello
```

### 5.2.2. Додавання рядків.

Як і в інших мовах програмуваннях в мові Python є можливість додавання рядків один до одного для отримання нових рядків. Це можна зробити з допомогою символу «+».

*Наприклад:*

```
import re

first_name = "Мурчик"
last_name = "Пушок"
full_name = first_name + " " + last_name
print(full_name)
```

*Результат:*

```
Мурчик Пушок
```

### 5.2.3. Отримання фрагментів рядка

Отримання фрагменту рядка часто використовується на практиці, виділення потрібної інформації із рядків. В мові

програмування Python існує декілька основних способи отримати фрагмент рядка:

- за допомогою операції індексування (таким способом можна отримати один символ із рядка);
- операція зрізу (цей спосіб дає можливість отримати фрагмент рядка починаючи з будь якої позиції).

Розглянемо детальніше кожен спосіб отримання фрагментів рядка. Подібно до списків до елементів рядка також можна звертатись по індексу та отримувати символ який знаходиться на потрібній нам позиції.

*Наприклад:*

```
user_name = "first_user"
print(user_name[3])
```

*Результат:*

```
s
```

Також доступне індексування в зворотному порядку для цього потрібно використати знак «-» мінуса. Тобто якщо в якості індексу використати -1 тоді буде отримано останній символ із кінця.

*Наприклад:*

```
user_name = "first_user"
print(user_name[-2])
```

*Результат:*

```
e
```

Отримання фрагмента рядка з допомогою зрізів дає можливість отримати фрагмент рядка потрібної довжини починаючи із необхідного індексу. Це дуже зручний і розповсюджений метод для отримання фрагменту рядка.

Існує дві форми зрізів це зрізи виду [SATRT:END] та розширені [START:END:STEP] де START це індекс початкового символу фрагменту рядка включно, END індекс кінця рядка не включно та STEP крок за індексом. Значення SATRT, END та STEP є не обов’язковими значеннями, тобто запис команди

матиме вигляд [:] або [::], у випадку якщо їх не вказувати будуть використані значення по завмовчуванню, тобто в якості значення SATRT буде використано початок рядку, в якості END кінець рядку та STEP буде рівним 1, цей механізм можна використовувати для копіювання рядків у інші комірки пам'яті.

Нижче наведені приклади використання зрізів для отримання фрагментів рядку.

*Наприклад:*

```
user_name = "first_user"
print(user_name[2:5], end="\n\n")
print(user_name[:5], end="\n\n")
print(user_name[2:], end="\n\n")
print(user_name[2:5:2], end="\n\n")
print(user_name[::2], end="\n\n")
print(user_name[::], end="\n\n")
```

*Результат:*

```
rst

first

rst_user

rt

frtue

first_user
```

### 5.2.4. Функції та методи для роботи з рядками.

В мові програмування Python існує велика кількість функцій для роботи з рядками. Найчастіше використовуються наступні функції:

- *find(needle)* та *rfind(needle)* функції які повертають індекси першого входження шуканого фрагменту рядку. Відмінність цих методів полягає лише в тому з якого боку починається пошук уривка find шукає з початку рядка, а rfind з кінця.

*Наприклад:*

```
user_name = "first_user"
print(user_name.find('rst'))
```

*Результат:*

```
2
```

- *replace(old_part, new_part)* метод який заміняє частину рядка рівну *old_part* на *new_part.*

*Наприклад:*

```
user_name = "first_user"
print(user_name.replace('user', 'tester'))
```

*Результат:*

```
first_tester
```

- *join* метод якій об’єднує елементи списку в рядок.

*Наприклад:*

```
words = ['user', 'name']
print(" ".join(words))
```

*Результат:*

```
user name
```

- *split(delimeter)* метод, який дає можливість розділити рядок з допомогою роздільника *delimeter*.

*Наприклад:*

```
print("user name".split())
```

*Результат:*

```
['user', 'name']
```

- *strip()* метод який видаляє пробіли на початку та в кінці рядка.

*Наприклад:*

```
print("   user ".strip())
```

*Результат:*

```
user
```

- *len(string)* функція яка визначає довжину рядка.*string.*

*Наприклад:*

```
print(len("   user "))
```

*Результат:*

```
8
```

- *string.encode(encoding)* цей метод кодує рядок в задане кодування За замовчуванням використовується кодування utf-8. Можна використовувати й інші варіанти кодування, але при неможливості закодувати рядок виникне помилка.

*Наприклад:*

```
print('cat'.encode(encoding='ascii'))
print('котик'.encode(encoding='utf-8'))
print('котик'.encode(encoding='cp1251'))
```

*Результат:*

```
b'cat'
b'\xd0\xba\xd0\xbe\xd1\x82\xd0\xb8\xd0\xba'
b'\xea\xee\xf2\xe8\xea'
```

- *str.format()* метод призначений для форматування вказаного аргументом значення та заповнення їх у рядок. Приклад використання функції зображений нижче, в ньому розглядається заповнення значеннями змінних {name} та {age} в рядку, значеннями які передаються в якості іменованих аргументів у метод *format.*

*Приклад:*
```
message = "Мене звуть {name}, мені {age} роки."
print(message.format(name = "Петро", age = 34))
```
*Результат:*
```
Мене звуть Петро, мені 34 роки.
```

Ще можливе використання даного методу без іменованих аргументів методу, приклад такого використання зображено нижче.

*Приклад:*
```
message = "Мене звуть {1}, мені {0} роки."
print(message.format(34, "Петро"))

message = "Мене звуть {}, мені {} роки."
print(message.format("Петро", 34))
```
*Результат:*
```
Мене звуть Петро, мені 34 роки.
Мене звуть Петро, мені 34 роки.
```

### 5.2.5. Регулярні вирази.

*Регулярний вираз* – це рядок шаблон описаний з допомогою спеціальних синтаксичних правил для рядків. Регулярні вирази дуже часто використовуються для перевірки правильності структури рядку наприклад для перевірки коректності email адреси або номеру мобільного телефону, також для пошуку частин тексту які задовільняють описаним в регулярному виразі

правилам, або для заміни тексту за певними правилами, до прикладу потрібно відфільтрувати рядок від різних спец символів.

В мові програмування Python підтримка регулярних виразів забезпечується з допомогою пакету re. Підключити цей пакет можна наступим чином *import re*. Розглянемо детальніше найбільш розповсюджені функції даного модулю:

- *re.match(pattern, string)* метод шукає по заданому шаблону на початку рядка. Тобто якщо шуканий фрагмент знаходиться не на початку рядку його не буде знайдено. Знайдений результат можна отримати з допомогою методу *group*. Приклад використання цих методів наведений нижче.

*Наприклад:*

```
import re

result = re.match(r"[0-9]*", "123 test string")
print("Результат: ", result.group(0))

result = re.match(r"[0-9]*", "test 123 string")
print("Результат: ", result.group(0))
```

*Результат:*

```
Результат:  123
Результат:
```

- *re.search(pattern, string)* метод подібний за своєю роботою до *re.match* але шукає по всьому рядку та поверне лише перше знайдене співпадіння.

*Наприклад:*

```
import re

result = re.search(r"[0-9]+", "test 123 string")
print("Результат: ", result.group(0))

result1 = re.search(r"[0-9]+", "test 123 string 456 string")
print("Результат: ", result1.group(0))
```

*Результат:*

```
Результат:  123
Результат:  123
```

- *re.findall(pattern, string)* метод якій повертає список всіх знайдених частин рядка за правилом *pattern* та в рядку *string*.

*Наприклад:*

```
import re

result = re.findall(r"[0-9]+", "test 123 string")
print("Результат: ", result)

result1 = re.findall(r"[0-9]+", "test 123 string 456 string")
print("Результат: ", result1)
```

*Результат:*

```
Результат:  ['123']
Результат:  ['123', '456']
```

- *re.split(pattern, string, [maxsplit=0])* цей метод розділяє рядок *string* за правилом описаним в *pattern*, до прикладу цей метод можна використати для розділення рядку за спецсимволами.

*Наприклад:*

```
import re

result = re.split(r"\s", "test string 123")
print(result)
```

*Результат:*

```
['test', 'string', '123']
```

- *re.sub(pattern, repl, string)* метод заміняє частини рядку які підходять під правило *pattern* та заміняє їх в рядку *string* на рядок *repl*. До прикладу це може бути використано для заміни всіх спец символів на якійсь один дозволений символ.

*Наприклад:*

```
import re

result = re.sub(r"\s", "_", "test string 123")
print(result)
```

*Результат:*

```
test_string_123
```

**5.3. Програма роботи**

1. Ознайомитися з теоретичними відомостями, що до роботи із рядками та регулярними виразами.
2. Виконати ознайомчі завдання теоретичних відомостей.
3. Виконати завдання згідно вашого варіанту. Додати коментарі до коду.
4. Оформити звіт.

## 5.4. Порядок виконання роботи

*Завдання 1*

Написати програму, яка буде визначати топ 3 символів, що найчастіше зустрічаються у рядку. Додаткові вимоги до програми:

- Пробіли повинні ігноруватись, тобто не потрапляти до підрахунку.
- Програма повинна видавати помилку у разі, якщо кількість символів у рядку менше 3.
- Програма повинна запитувати рядок у користувача.

*Завдання 2*

Написати функцію для перевірки коректності формату email адреси введеної користувачам використовуючи регулярні вирази.

## 5.5. Контрольні питання

1. Що таке рядок?
2. Що таке керуючі символи? Як їх ще називають?
3. Які є типи виводу рядка?
4. Які існують керуючі символи? Навіщо вони?
5. Методи додавання рядка.
6. Способи отримати фрагмент рядка.
7. Опишіть дві форми зрізі рядка.
8. Функції для роботи з рядками.
9. Що таке регулярний вираз?
10. Забезпечення використання регулярних виразів.
11. Функції модулю re.

Лабораторна робота №6

# Робота з файлами в мові програмування Python

## 6.1. Мета роботи

Вивчити теоретичні відомості, що до роботи з файлами. Навчитися розрізняти та використовувати їх функції та особливості. Вивчити основні методи роботи з файлами. Навчитися здійснювати операції читання та запису для файлів у мові Python.

## 6.2. Теоретичні відомості

### 6.2.1. Загальні відомості

Взаємодія з файлами в мові програмування Python дає можливість зберігати інформацію, яка була оброблена програмою, в постійній пам'яті комп'ютера та мати можливість отримати доступ до цієї інформації в будь який час.

*Файл* – це впорядкована множина даних, що зберігається на диску та може бути використаним програми для збереження даних або ж і є самою програмою.

Існує два типи файлів: текстові та бінарні файли. Текстові файли – це файли в яких інформація представлена у текстовому вигляді, до прикладу – це файли з розширенням txt, csv, html та багато інших. *Бінарні файли* – це файли дані в яких зберігаються в бінарному вигляді.

### 6.2.2. Робота із текстовими файлами

В мові програмування Python існують вбудовані функції для роботи з файлами. Перед початком роботи з файлом його потрібно відкрити з допомогою функції **open(file_path, mode)**. У цій функції є два вхідних параметри **file_path** та **mode**, **file_path** це шлях до файлу, а **mode** це режим в якому буде відкритий файл, список всіх доступних режимів можна знайти в таблиці.

| Режим | Опис |
|---|---|
| r | Відкриття на читання |
| w | Відкриття на запис, вміст файлу видаляється, якщо файла не існує буде створено новий. |
| x | Відкриття файлу на запис, якщо файлу не існує то буде викинуте виключення. |
| a | Відкриття файлу на дозаписування файлу вміст буде дописано в кінець файлу. |
| b | Відкриває файл в бінарному вигляді. |
| t | Відкриття в текстовому режимі. |
| + | Відкриття на читання та запис. |

Режими можуть бути об'єднані, до прикладу 'rb' – читання файлу в двійковій формі. По завмовчуванню рівний 'rt' тобто читання в текстовому форматі. Приклад відкриття файл який знаходиться поряд із кодом програми.

*Приклад:*

```
f = open('text.txt')
```

Після відкриття файлу на читання будуть доступні декілька способів для зчитування даних з файлу.

Перший спосіб це використання методу *read* цей метод зчитує всі дані з файлу. Приклад використання методу нижче.

*Приклад:*

```
f = open('text.txt')
data = f.read()
print(data)
```

*Результат:*

```
Hello world.
Hello people.
```

Вивід – це вміст файлу.

Ще одним способом може бути метод *readlines* даний метод зчитує всі рядки з файлу та повертає у вигляді списку. Приклад використання нижче.

*Приклад:*

```
f = open('text.txt')
data = f.readlines()
print(data)
```

*Результат:*

```
['Hello world.\n', 'Hello people.']
```

Також для зчитування файлу можна використати цикл **for** передавши в якості даних об'єкт відкритого файлу. Таким чином файл буде зчитуватись по рядку.

*Приклад:*

```
f = open('text.txt')
for data in f:
    print(data)
```

*Результат:*

```
Hello world.

Hello people.
```

Після завершення роботи із файлом його потрібно закривати, щоб зробити доступним для інших процесів. Це робиться з допомогою методу **close.** Приклад роботи з даним методом зображений нижче.

*Приклад:*

```
f = open('text.txt')
for data in f:
    print(data)
f.close()
```

Для того щоб відкрити файл для запису використовується модифікатор 'w'. Після відкриття файл на запис буде доступний метод для запису у файл.

*Приклад:*

```
f = open('text.txt', 'w')
f.write("Hello students")
f.close()
```

*Результат:*

```
Hello students
```

Вміст файлу text.txt

### 6.2.3. Робота із бінарними файлами

Для роботи із бінарними файлами можна використати пакет *pickle* який доступний в стандартній бібліотеці Python.

Для запису у файл бінарних даних потрібно відкрити файл для цього використовуючи режим ‘wb’.

*Приклад:*

```
import pickle

data = "string for writing"
f = open('text.data', 'wb')
pickle.dump(data, f)
f.close()
```

*Результат:*

```
€EOT•SYNNULNULNULNULNULNULNULDC2string for writing”.
```

Вміст файлу text.data

В бінарні файли можна записувати не тільки рядки в бінарній формі, а й різні складні структури даних наприклад словники.

*Приклад:*

```
import pickle

data = {
    "one": 1,
    "two": 2,
}
f = open('text.data', 'wb')
pickle.dump(data, f)
f.close()
```

*Результат:*

```
€EOT•NAKNULNULNULNULNULNULNUL}”(ETXone”KSOHETXtwo”KSTXu.
```

Вміст файлу text.data

Для того щоб зчитувати бінарні дані із файлу його подібно відкрити в режимі читання бінарних даних та використати метод **load**.

*Приклад:*

```
import pickle

f = open('text.data', 'rb')
data = pickle.load(f)
f.close()

print(data)
```

*Результат:*

```
{'one': 1, 'two': 2}
```

### 6.3. Програма роботи

1. Ознайомитися з теоретичними відомостями, що до роботи із текстовими та бінарними файлами.
2. Виконати ознайомчі завдання із теоретичних відомостей.
3. Виконати завдання згідно вашого варіанту. Додати коментарі до коду.
4. Оформити звіт.

### 6.4. Порядок виконання роботи

*Завдання 1:*

Написати дві програми: перша запитуватиме в користувача особисті дані (прізвище, ім’я та по-батькові, дату народження, вік, місто проживання та номер телефону) та записуватиме їх у файл та друга, що буде зчитувати дані із файлу та виводити їх на екран сортуючи людей на віком.

*Завдання 2:*

Створити програму, яка буде запитувати в користувача бали з декількох предметів, зберігати їх в словнику та записувати його у файл, а друга програма зчитує дані із файлу та виводити їх на екран.

*Завдання 3:*

Створити текстовий файл в якому рядки чисел розділені пробілами (мінімум 10 рядків). Знайти суму чисел в кожному рядку та зберегти його в словник, де ключ – це номер рядка, а значення – сума чисел. Вивести даний словник на екран.

*Завдання 4:*

Написати програму, яка видаляє рядок із файлу (файл створений в завданні 3) за його номером, нумерація рядків починається з 1.

### 6.5. Контрольні питання

1. Що таке файл?
2. Які існують типи файлів?
3. Що таке текстові і бінарні файли?
4. Які існують вбудовані функції для роботи з файлами?
5. Назвіть режими файлів.
6. Охарактеризуйте пакет *pickle*.
7. Методи роботи з файлами.

Лабораторна робота №7

# Об'єктно-орієнтоване програмування та його принципи

## 7.1. Мета роботи

Вивчити теоретичні відомості, що до принципів об'єктно-орієнтованого програмування. Навчитися працювати з класами та об'єктами. Навчитись писати код в об'єктно-орієнтованому стилі.

## 7.2. Теоретичні відомості

### 7.2.1. Загальні відомості

*Об'єктно-орієнтоване програмування (ООП)* – це парадигма програмування в якій компоненти програми розроблені на основі зв'язані між собою об'єктів, де характеристики та взаємодія об'єктів описуються в класах.

### 7.2.2. Класи та об'єкти.

*Клас* – в об'єктно-орієнтованому програмування можна представити у вигляді креслення або карти об'єкту. Дивлячись на клас можна зрозуміти як побудований об'єкт, зрозуміти які характеристики має об'єкт, побачити його логіку роботи та зрозуміти як він взаємодіє із іншими об'єктами.

Розглянемо детальніше, що таке клас на прикладі автомобіля. З точки зору ООП автомобіль – це абстрактне поняття, яким можна описати велику кількість конкретних марок та моделей автомобілів, тобто об'єктів або їх ще називають екземплярами класу.

Таким чином *об'єкт* – це конкретний екземпляр певного класу, тобто у випадку з автомобілями – це може бути автомобіль Mazda 6 2021 року випуску.

В мові програмування Python класи можна створити з допомогою ключового слова **class**. Приклад створення класу зображений у прикладі:

*Приклад:*

```
class Car:
    pass
```

Використовуючи ключове слово **pass** можна пропустити оголошення класу.

Для оголошення характеристик класу в ООП існують атрибути або в англомовній термінології *properties*. Їх прийнято оголошувати в конструкторі класу.

*Конструктор класу* – це метод, який визивається під час створення екземпляру (про це детальніше згодом). В конструктор можна передати аргументи по аналогії із функціями в Python, та використати їх для оголошення характеристик класу. Перший аргумент **self** - це екземпляр описаного класу Car, який автоматично передається в якості першого аргументу у всі методи класу.

Таким чином можна отримати клас Car з трьома атрибутами які оголошуватимуть виробника, модель. Та рік випуску автомобіля.

*Приклад:*

```
class Car:
    def __init__(self, manufacturer = None, model = None, age = None):
        self.manufacturer = manufacturer
        self.model = model
        self.age = age
```

Для обробки даних або виконання якихось інших дій в класах існують методи. Методи аналоги функцій, але для класів. Також в них першим аргументом передається екземпляр об’єкту описаного класу, його прийнято іменувати **self.** Оголосимо методи для запуску, зупинки та руху автомобіля, в та реалізуємо їх. Для прикладу в якості їхньої реалізації виведемо дію яку вони реалізовуватимуть.

*Приклад:*

```
class Car:
    def __init__(self, manufacturer = None, model = None, age = None):
        self.manufacturer = manufacturer
        self.model = model
        self.age = age

    def start(self):
        print('Start {} car'.format(self.manufacturer))

    def stop(self):
        print('Stop {} car'.format(self.manufacturer))

    def move(self):
        print('Mode {} car'.format(self.manufacturer))
```

Таким чином було створено клас, який описує автомобіль.

Для створення екземпляра класу необхідно присвоїти його змінній, синтаксис подібний до виклику функції, у дужках можна передати аргументи, які приймає конструктор класу.

*Приклад:*

```
mazda_car = Car('Mazda', '6', '2021')
```

Для доступу до атрибутів об’єкту та методів використовується символ крапки.

*Приклад:*

```
mazda_car = Car('Mazda', '6', '2021')
print(mazda_car.manufacturer)
mazda car.start()
```

*Результат:*

```
Mazda
Start Mazda car
```

В класах мови програмування Python також існують так звані «чарівні» методи з допомогою них можна описати логіку роботи для роботи об’єкту із циклами, з математичними операторам або з функцією print та багато іншого, так як це реалізовано в інших типах даних в Python адже клас – це є також опис свого типу даних. Такі методи зазвичай починаються та закінчується двома нижніми підкресленнями «__». Розглянемо приклад який

перетворюватиме об’єкт класу Car в типу рядок. Для цього треба реалізувати метод **__str__**.

*Приклад:*

```
class Car:
    def __init__(self, manufacturer = None, model = None, age = None):
        self.manufacturer = manufacturer
        self.model = model
        self.age = age

    def __str__(self):
        return "Manufacturer {},\nModel: {},\nAge: {}".format(
            self.manufacturer, self.model, self.age)

mazda_car = Car('Mazda', '6', '2021')
print(mazda_car)
```

*Результат:*

```
Manufacturer Mazda,
Model: 6,
Age: 2021
```

### 7.3. Програма роботи

1. Ознайомитися з теоретичними відомостями, що об’єктно-орієнтованого програмування.
2. Виконати ознайомчі завдання із теоретичних відомостей.
3. Виконати завдання згідно вашого варіанту. Додати коментарі до коду.
4. Оформити звіт.

### 7.4. Порядок виконання роботи

*Завдання 1:*

Створити клас студента та заповнити поля цього класу із клавіатури, вивести дані на екран.

*Завдання 2:*

Написати клас, який буде зберігати в атрибуті та змінювати баланс користувача. В класі має бути два методи: для збільшення балансу та зменшення балансу.

### 7.5. Контрольні питання

1. Що таке об’єктно-орієнтоване програмування?
2. Що таке клас?
3. Які особливості можна зрозуміти з вигляду класу?
4. Що таке об’єкт?
5. Як створюються класи?
6. Наведіть приклад створення класу.
7. Як запустити оголошення класу?
8. Що таке атрибути?
9. Що таке конструктор класу?
10. За що відповідає self?
11. Як отримати доступ до атрибутів об’єкту та методів?
12. За що відповідає __str__?

Лабораторна робота №8

# Парадигми в об'єктно-орієнтованого програмуванні

## 8.1. Мета роботи

Ознайомитися з основними парадигмами об’єктно-орієнтованого програмування. Вивчити теоретичні відомості, що до парадигмів об’єктно-орієнтованого програмування. Навчитись писати код в об’єктно-орієнтованому стилі використовуючи парадигми та патерни.

## 8.2. Теоретичні відомості

### 8.2.1. Загальні відомості

Для написання якісно коду потрібно вміти структурувати цей код в проекті та вміти розділяти логіку на частини, для того щоб код можна було підтримувати та розвивати. Для цього в програмуванні існує велика кількість правил, принципів, парадигм та патернів.

### 8.2.2. Парадигми ООП.

В об’єктно-орієнтованому програмуванні є велика кількість правил, понять, які визначають стиль написання програм. До цих правил також можна віднести парадигми об’єктно-орієнтованого програмування.

*Виділяють 6 основних парадигм ООП*:

1. Інкапсуляція;
2. Наслідування;
3. Поліморфізм;
4. Абстракція;
5. Повторне використання;
6. Відправка повідомлень.

Розглянемо детальніше кожну із цих парадигм.

### 8.2.3. Інкапсуляція

*Інкапсуляція* – це парадигма, яка об’єднує код та дані, також захищає код і данні від зовнішнього впливу та від не правильного

використання коду. Разом це все створює об’єкт в якому можуть бути приватні дані, якими можуть маніпулювати лише методи об’єкту. Цей весь механізм створює ще одну корисну властивість – приховування реалізації програмних частин.

Для реалізації цього механізму в Python існують *модифікатори доступу*:

- *public* – публічні методи та атрибути, доступні із будь якого місця в програмі. Тобто цей модифікатор будує інтерфейс взаємодії роботи із класом.
- *protected* – захищені методи та атрибути доступні лише в класі у якому оголошений цей атрибут або в дочірньому класі (дочірні класи буде розглянуто в наступній парадигмі)
- *private* – приватні методи та атрибути, які доступні тільки у класі в якому вони оголошені.

Для того щоб оголосити приватний або захищений метод або атрибут необхідно перед назвою додати нижнє підкреслювання для захищених змінних або методів та два для приватних.

*Приклад:*

```
class Car:
    def __init__(self,
                 manufacturer = None,
                 model = None,
                 age = None,
                 enginсe_software_version = None):
        self.manufacturer = manufacturer # публічний атрибут
        self.model = model # публічний атрибут
        self.age = age # публічний атрибут

        self._enginсe_software_version = enginсe_software_version # захищений атрибут
        self.__circle_count = 0 # приватна атрибут
```

### 8.2.4. Наслідування

*Наслідування* в об’єктно-орієнтованому програмуванні схоже до наслідування в реальному житті, де дитина наслідує ті чи інші характеристики чи поведінку дорослих і доповнює їх своїми власними.

Повернемося до прикладу з автомобілями, таким чином автомобіль Mazda може бути унаслідуваний від класу автомобіль. Головна ідея наслідування в ООП – це те, що клас може наслідувати характеристики іншого класу. Його називають батьківський клас, а клас, який наслідує називається – дочірнім.

Розглянемо приклад наслідування в Python.

*Приклад:*

```
class Car:
    def __init__(self, model = None, age = None):
        self.model = model
        self.age = age

    def __str__(self):
        return "Model: {},\nAge: {}".format(self.model, self.age)

class MazdaCar(Car):
    def __int__(self, model = None, age = None):
        self.manufacturer = 'Mazda'
        super().__init__(model, age)

mazda_car = MazdaCar('6', '2021')
print(mazda_car)
```

*Результат:*

```
Model: 6,
Age: 2021
```

Для того, щоб унаслідувати один клас від іншого потрібно під час оголошення дочірнього класу поряд з назвою в дужках вказати назву класу від якого буде відбуватись наслідування. Як видно із виводу програми в дочірньому класи доступні методи батьківського класу.

В Python також є можливість множинного наслідування, тобто дочірній клас може мати декілька батьківських класів.

Для того щоб унаслідувати дочірній клас від двох або більше батьківських їх потрібно вказати через кому в дужках поряд з назвою класу, так як на прикладі.

*Приклад:*

```
class Phone:
    def make_call(self):
        print('call')

class Camera:
    def take_photo(self):
        print('take_photo')

class Smartphone(Phone, Camera):
    pass

smartphone = Smartphone()
smartphone.make_call()
smartphone.take_photo()
```

*Результат:*

```
call
take_photo
```

Як видно з прикладу в дочірньому класі також доступні всі методи двох батьківських.

### 8.2.5. Поліморфізм

*Поліморфізм* в ООП – здатність об'єкта поводити себе по-різному, в залежності від ситуації і реагувати на певну дію строго специфічним для себе чином. Для реалізації цієї можливості в Python існує перевантаження та перевизначення методів.

*Перевантаження методу* – це зміна поведінки методу в залежності від кількості або типу аргументів.

Приклад перевантаження методу зображений нижче.

*Приклад:* *Результат:*

```
class Calculator:
    def sum(self, a, b = None):
        if b is not None:
            print(a + b)
        else:
            print(a)

calculator = Calculator()
calculator.sum(10)
calculator.sum(10, 50)
```

```
10
60
```

Перевизначення методу можливе під час наявності методів з однаковою назвою в дочірньому та батьківському класі. Приклад перевизначення методу можна побачити нижче.

*Приклад:*

```
class Car:
    def __init__(self, model = None, age = None):
        self.model = model
        self.age = age

class MazdaCar(Car):
    def __int__(self, model = None, age = None):
        self.manufacturer = 'Mazda'
        super().__init__(model, age)
```

Також з допомогою ключового слова *super* можливий виклик логіки із методу оголошеного в батьківському класі.

### 8.2.6. Абстракція

*Абстракція* в ООП – це використання лише того мінімального набору властивостей та методів, яких достатньо для виконання певної задачі. Тобто класи потрібно будувати таким чином, щоб об’єкт виконував одне завдання і не більше, тобто, якщо клас має записувати дані в базу даних, то він має тільки записувати дані в базу даних не більше, це називається рівнем абстракції. Тобто

програмне забезпечення має будуватись так щоб, в ньому присутні були різні рівні абстракції і класи проектувались лише під один рівень абстракції, не більше. Тобто в прикладі про базу даних може бути два класи один записує в базу інший читає із бази, але більше нічого вони не роблять, інші дії, наприклад препроцесінг даних перед збереженням в базу, має відбуватись в іншому класі на вищому рівні абстракції. Під час змішування рівнів абстракції в класах відбувається перемішування логіки, що шкодить подальшому розвитку проекту, подальші правки вносити стає все важче і важче. Тому краще витрати на початку трішки більше часу на грамотне проектування та розділення логікі на різні рівні абстракції та їх в свою чергу на різні класи, таким чином щоб клас виконував одну дію, або зберігав дані притаманні лише одній сутності.

### 8.2.7. Відправка повідомлень

*Відправка повідомлень* – це по факту виклик методів одних класів іншими. Таким чином інформуються інші класи про якісь події в іншому.

### 8.2.8. Повторне використання

*Повторне використання коду* – це використання одного і того ж коду в різних частинах програми без копіювання логіки. Всі вище описані патерни використовуються для забезпечення цього патерну.

## 8.3. Програма роботи

1. Ознайомитися з теоретичними відомостями, що до об’єктно-орієнтованого програмування.
2. Виконати ознайомчі завдання із теоретичних відомостей.
3. Виконати завдання згідно вашого варіанту. Додати коментарі до коду.
4. Оформити звіт.

## 8.4. Порядок виконання роботи

*Завдання 1:*

Написати програму, яка буде працювати з двома сутностями, баланс користувача та транзакції, які змінюють цю суму. В транзакції має бути два основні поля це: сума і тип, та в залежності від типу баланс користувача буде змінюватись: якщо тип транзакції кредит, то баланс має зменшуватись, якщо дебіт, то збільшуватись.

*Завдання 2:*

Написати програму, яка буде запитувати в користувача хто він: студент чи викладач. У випадку, якщо це викладач, запитати в нього: прізвище, ім'я, кафедру та посаду на кафедрі. У випадку якщо, то студент запитати: прізвище, ім'я, спеціальність та курс. Отримані дані записати у файл. Для написання програми використати ООП та його парадигми.

## 8.5. Контрольні питання

1. Назвіть основні парадигми ООП.
2. Що таке інкапсуляція?
3. Які є модифікатори доступу?
4. Що таке наслідування?
5. Що таке множинне наслідування?
6. Що таке поліморфізм?
7. Навіщо використовують *super?*
8. Що таке абстракція?
9. Що таке відправка повідомлень?
10. Що таке повторне використання коду?

Лабораторна робота №9

# Парсинг web-сторінок. Розробка сайтів засобами мови Python

## 9.1. Мета роботи

Познайомитися з принципами розробки та парсингу сайтів. Ознайомитися із принципами роботи із HTML та CSS. Вивчити структуру DOM документів. Навчитись працювати із Django-web framework.

## 9.2. Теоретичні відомості

### 9.2.1. HTML та DOM

Всі сучасні сайти в інтернеті використовують HTML як мову розмітки. *HTML* – це мова гіпертекстової розмітки, самий базовий блок інтернет сторінки, який визначає наповнення та структуру документа.

HTML використовує «*markup*» розмітку для відображення тексту, зображень та іншого.

Після того як браузер зробить HTTP запит на сервер та отримає HTML код сторінки, браузер аналізує його та будує DOM-дерево.

*DOM* – це об'єктна модель документа, яку браузер створює в пам'яті комп'ютера на основі HTML, який отримав із сервера.

Браузер побудувавши DOM-дерево використовує його для відображення сторінки та надає зручний API для використання його в JavaScript для отримання із нього інформації та модифікації.

HTML сторінки складаються із тегів, які можуть буть вкладені один в одного, атрибутів та коментарів.

В документах для створення розмітки окремі теги розміщуються в середині інших, та для деяких тегів можуть бути використані атрибути.

Приклад HTML документу зображений нижче:

```html
<html>
  <head>
    <title>Заголовок сторінки</title>
  </head>
  <body>
    <h1 style="color: red;">Заголовок</h1>
    <div>
      Контент сторінки
    </div>
  </body>
</html>
```

### 9.2.2. Парсинг

*Parsing* або *scraping* – конвертація даних призначена для перегляду людиною у web-браузера структуровані дані.

Для парсингу необхідно завантажити html сторінку та розпарсити її перетворивши в розпаршені структуровані дані.

Для вирішення першої задачі можна використати бібліотеку *requests* та його метод *get.*

*Приклад:*

```python
import requests
r = requests.get('https://nuwm.edu.ua/')
responce = r.text

print(r.status_code)
print(responce)
```

Даний приклад завантажить web-сторінку та відобразить її на екрані терміналу.

Далі необхідно знайти необхідний нам елемент для прасингу, для цього можна перейти на необхідний нам сайт, для прикладу візьмемо головну сторінку сайту Національного університету водного господарства та природокористування https://nuwm.edu.ua/. Знайдемо елемент із якого ми будемо парсити посилання тобто тег “a”. Для цього натиснемо правою

кнопкою миші та виберемо пункт “Переглянути” або в англомовній версії “Inspect”

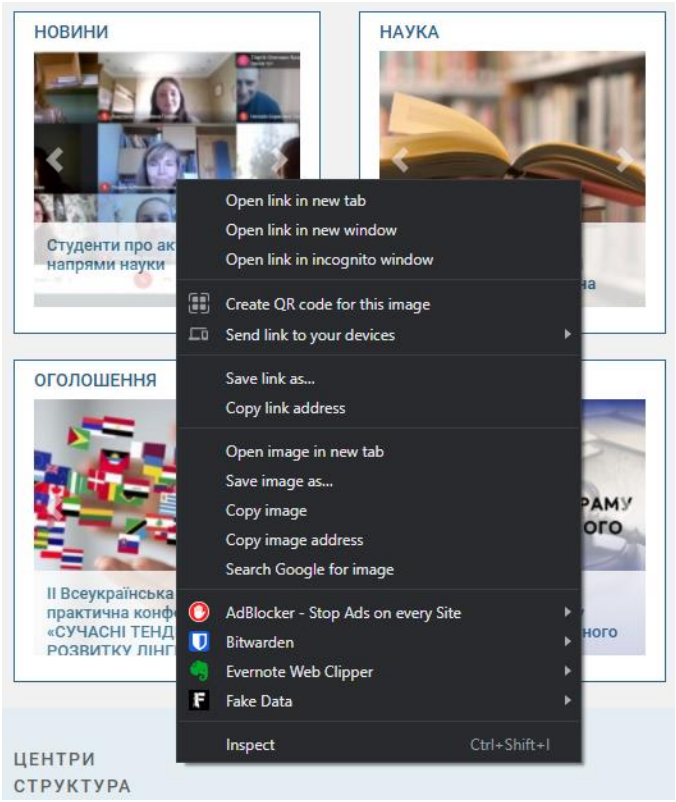


Рис.9.1. Приклад перегляду даних сайту https://nuwm.edu.ua/

Далі потрібно знайти унікальний шлях до елементу в меню яке відкрилось. Для цього потрібно вибирати тег та клас або id елемента.

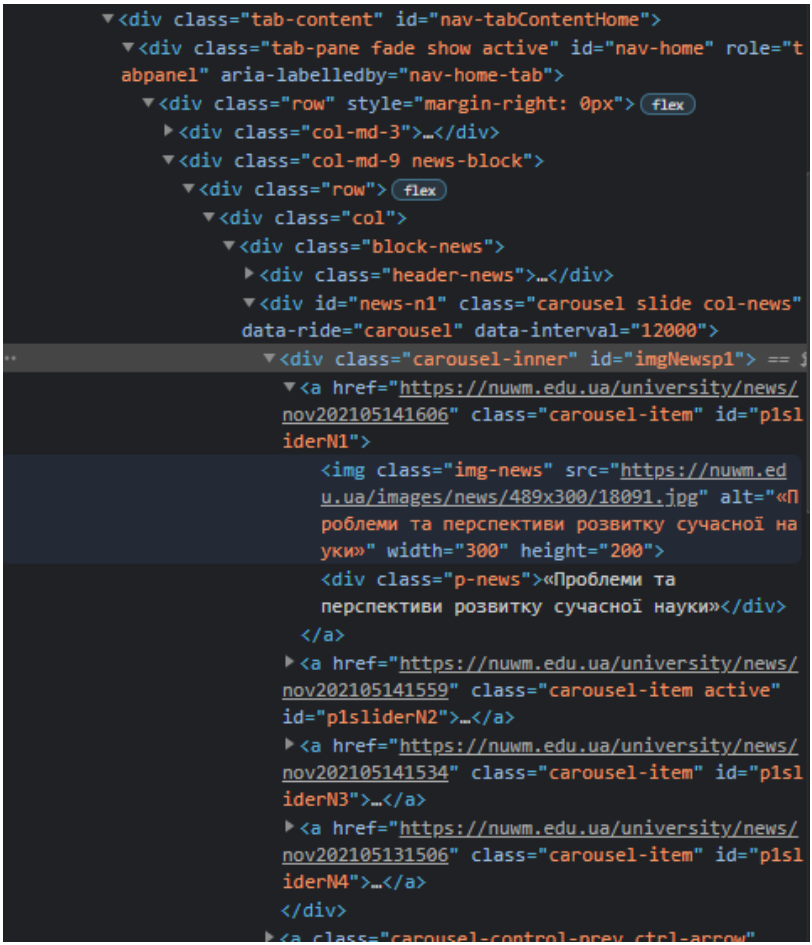


Рис.9.2. Приклад знаходження id елементу

Далі використовуючи бібліотеку bs4 знайдемо всі посилання які знаходяться в обраному слайдері.

*Приклад:*

```
import requests
from bs4 import BeautifulSoup

r = requests.get('https://nuwm.edu.ua/')
responce = r.text

soup = BeautifulSoup(responce)
for item in soup.find_all('div', {'class': 'carousel-item'}):
    print(item.find('a')['href'])
```

*Результат:*

```
https://nuwm.edu.ua/university/ads/nov2021050720221
https://start.nuwm.edu.ua/olimpiada
https://start.nuwm.edu.ua/item/138-%D1%80%D0%B5%D1%94%D1%81%D1%82%D1%80%D0%B0%D1%86%D1%96%D1%8F-%D1%94%D0%B2%D1%96-%D1%94%D1%84%D0%B2%D0%B2
https://www.facebook.com/events/183075797016698
http://www.gwp.org/
http://www.prostir.rivne.ua
https://www.salto-youth.net/
https://uhe.gov.ua/
https://www.eib.org/en/index.htm
```

### 9.2.3. Django

Для розробки сайтів часто використовується феріmфорк Django.

Для початку роботи з Django його необхідно встановити. Для цього потрібно запустити команду:

***pip install Django==3.2.3***

Після встановлення потрібно створити проект для Django за допомогою:

***django-admin startproject mysite .***

Створивши проект змінимо декілька налаштувань, для цього потрібно внести зміни у файл *mysite/settings.py*

У файлі *settings.py* знайдіть рядок, що містить TIME_ZONE і замініть його на ваш часовий пояс:

*TIME_ZONE = 'Europe/Kiev'*

Дуже часто сайти використовують бази даних для цього у файлі *settings.py* необхідно додати наступні рядки:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Щоб створити базу даних для нашого блогу, запустимо наступне в консолі: *python manage.py migrate*.

Після конфігурування та створення бази даних можна запустити веб-сервер для розробки командою:

***python manage.py runserver 0:8000***

Далі у браузері необхідно відрити сторінку за посиланням http://127.0.0.1:8000/, де буде відображатись базова сторінка проекту.

### 9.3. Програма роботи

1. Ознайомитися з теоретичними відомостями, що до парсингу web-сторінок та розробки сайтів.
2. Виконати ознайомчі завдання із теоретичних відомостей.
3. Виконати завдання згідно вашого варіанту. Додати коментарі до коду.
4. Оформити звіт.

### 9.4. Порядок виконання роботи

*Завдання 1:*

Написати парсер який буде витягувати посилання із сайту за вашим вибором.

*Завдання 2:*

Створити, налаштувати та запустити проект Django. У звіт вставити файл конфігурації, вивід команди *python manage.py*

*migrate* та сркіншот стандартної сторінки проекту Django в браузері.

### 9.5. Контрольні питання

1. Що таке HTML?
2. Що таке DOM?
3. Навіщо слугує *markup*?
4. З чого складаються HTML сторінки?
5. Наведіть приклад найпростішого HTML документу.
6. Що таке *Parsing*?
7. В чому різниця між *parsing* і *scraping*?
8. Наведіть алгоритм парсингу сторінки.
9. Для початку роботи з Django потрібно…?
10. Як створити проект для Django?

## Список літератури


1. Welcome to Python : веб сайт. URL: https://www.python.org/
2. Документація Python : веб сайт. URL: https://docs.python.org/2/
3. The Python Tutorial : веб сайт. URL: https://docs.python.org/3/tutorial/
4. Т. Гедис. Починаємо програмувати на Python: підручник. БХВ-Петербург, 4-е видання, 2019. 768 с. URL: https://fileskachat.com/getfile/69426_d9e0b302eb1cdcc520c14037fef1682b
5. М. Лутц. Вивчаємо Python, том 1: підручник. Діалектика, 5-е вид, 2020. 832 с. URL: https://balka-book.com/files/2019/09_02/10_41/u_files_store_3_1985456.zip
6. Е. Метиз. Пришвидшений курс Python: підручник. Видавництво Старого Лева, 2021. 600 с.
7. О. Швець. Занурення в патерни проектування : підручник. 2021. 393 с. URL: https://refactoring.guru/files/design-patterns-uk-demo.pdf
8. Присяжнюк, О. В. Методичні вказівки до виконання лабораторних робіт з навчальної дисципліни «Програмування. Частина 2. Програмування мовою Python» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація та комп'ютерно-інтегровані технології» спеціальностей 151 «Автоматизація та комп'ютерноінтегровані технології», 141 «Електроенергетика, електротехніка та електромеханіка» денної та заочної форм навчання: методичне забезпечення. Рівне : НУВГП. 2020, 165 с. URL: http://ep3.nuwm.edu.ua/17989/