МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни

“Інженерія прикладних інтелектуально-орієнтованих програмних продуктів”

для студентів спеціальностей
121 “Інженерія програмного забезпечення” та
122 “Комп’ютерні науки та інформаційні технології”
(всіх форм навчання)

2016

Методичні вказівки до виконання лабораторних робіт з дисципліни “Інженерія прикладних інтелектуально-орієнтованих програмних продуктів” для студентів спеціальностей 121 “Інженерія програмного забезпечення” та 122 “Комп’ютерні науки та інформаційні технології” (всіх форм навчання) / В.М. Льовкін. – Запоріжжя : ЗНТУ, 2016. – 80 с.

Автор: В.М. Льовкін, канд. техн. наук, доцент

Рецензент: А.О. Олійник, канд. техн. наук, доцент

Відповідальний
за випуск: С.О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
“Програмні засоби”

Протокол № 1
від “16” серпня 2016 р.

ЗМІСТ

ВСТУП

Дане видання призначене для отримання студентами всіх форм навчання фундаментальних знань з конструювання та розроблення інтелектуально-орієнтованого програмного забезпечення на основі використання сучасних засобів програмування у різних прикладних галузях (аналіз документів, оброблення природньої мови, наукові обчислення, розроблення web-додатків) та практичного засвоєння отриманих знань.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. Дані методичні вказівки містять тільки основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт, тому для виконання лабораторної роботи та при підготовці до її захисту необхідно ознайомитись з конспектом лекцій та опрацювати весь необхідний матеріал, наведений у переліку рекомендованої літератури, використовуючи також статті у інтернет-виданнях та актуальних наукових журналах з проблем штучного інтелекту.

Для одержання заліку з кожної роботи студент повинен у відповідності зі всіма наведеними вимогами розробити програмне забезпечення та оформити звіт, після чого продемонструвати на комп'ютері розроблене програмне забезпечення з виконанням усіх запропонованих викладачем тестів. Звіт виконують на білому папері формату А4. Текст розміщують з однієї сторони аркуша. Поля сторінки з усіх боків – 20 мм. Аркуші вміщують у канцелярський файл.

Усі завдання повинні виконуватись студентами індивідуально і не містити ознак плагіату як в оформленому звіті так і в розробленому програмному забезпеченні. Під час співбесіди при захисті лабораторної роботи студент повинен виявити знання щодо мети роботи, теоретичного матеріалу, методів виконання кожного етапу роботи, змісту основних розділів звіту з демонстрацією результатів на конкретних прикладах, практичних прийомів використання теоретичного матеріалу в розробленому програмному забезпеченні. Студент повинен вміти обґрунтувати всі прийняті ним проектні рішення та правильно аналізувати і використовувати на практиці отримані результати. Для базової самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

ЛАБОРАТОРНА РОБОТА № 1
ВИКОРИСТАННЯ БАЗОВИХ ТИПІВ ТА ЗАСОБІВ МОВИ ПРОГРАМУВАННЯ PYTHON ДЛЯ МУЛЬТИПАРАДИГМЕННОГО ПРОГРАМУВАННЯ

1.1. Мета роботи

1.1.1 Ознайомитись з основними можливостями, парадигмами, типами даних, синтаксичними особливостями та принципами мови програмування Python.

1.1.2 Навчитися розробляти програми мовою програмування Python на основі мультипарадигменного програмування.

1.2. Короткі теоретичні відомості

1.2.1 Середовища та інструментарій розроблення програм мовою Python

Python – популярна високорівнева мова програмування, яка використовується для розроблення програм та прикладних сценаріїв у різноманітних галузях.

Написання програмного коду на Python включає не тільки мову програмування, але й інтерпретатор, який необхідно встановити для початку роботи.

На даний момент існують наступні основні реалізації мови програмування Python:

– CPython – оригінальна стандартна реалізація мови програмування Python, написана мовою C, що дозволяє керувати компонентами, написаними мовами C та C++;

– Jython – альтернативна реалізація мови Python, основна мета якої – тісна інтеграція з мовою програмування Java. Реалізація Jython побудована з Java-класів, які виконують компіляцію програмного коду мовою Python у байт-код Java та передають отриманий байт-код віртуальній машині Java. Сценарії Jython можуть виступати у якості аплетів та сервлетів, створювати графічний інтерфейс з використанням механізмів Java тощо;

– IronPython – реалізація, призначена для забезпечення інтеграції програм Python з додатками, створеними для роботи в

середовищі Microsoft .NET Framework операційної системи Windows, а також у Mono – відкритому еквіваленті для операційної системи Linux. IronPython дозволяє програмам мовою Python грати роль як клієнтських, так і серверних компонентів, доступних з інших мов програмування .NET;

– PyPy – реалізація інтерпретатора мови програмування Python, написана за допомогою Python.

Дані реалізації дозволяють виконувати інструкції в інтерактивному режимі або створювати окремі модулі.

Останні версії операційних систем сімейства Linux (Debian, Ubuntu, Fedora) мають встановлений компонент Python за замовчанням.

Таким чином, процес інсталяції інтерпретатора залежить від операційної системи та обраної реалізації. Стандартним інтерпретатором є CPython, який зазвичай використовується за замовчуванням в операційних системах сімейства Linux. Завантажити його можна з сайту http://www.python.org.

Окрім використання стандартного інтегрованого середовища розробки IDLE з пакету CPython, яке дозволяє редагувати, запускати, зневаджувати програми мовою Python, можна також розробляти програми мовою Python за допомогою сучасних IDE. Розроблення програм мовою Python підтримується зокрема наступними IDE:

– Eclipse;
– Komodo;
– NetBeans;
– PythonWin;
– Wing;
– Microsoft Visual Studio.

Для створення програмних проектів на Python у середовищі Eclipse необхідно встановити плагін PyDev за допомогою меню Help → Eclipse Marketplace… або завантажити архів з плагіном та помістити дані в теку eclipse\dropins.

Після того, як плагін встановлено, можна створити PyDev Project (рис. 1.1), звідки можна задати налаштування інтерпретатора, який буде використовуватись для роботи з даним проектом.

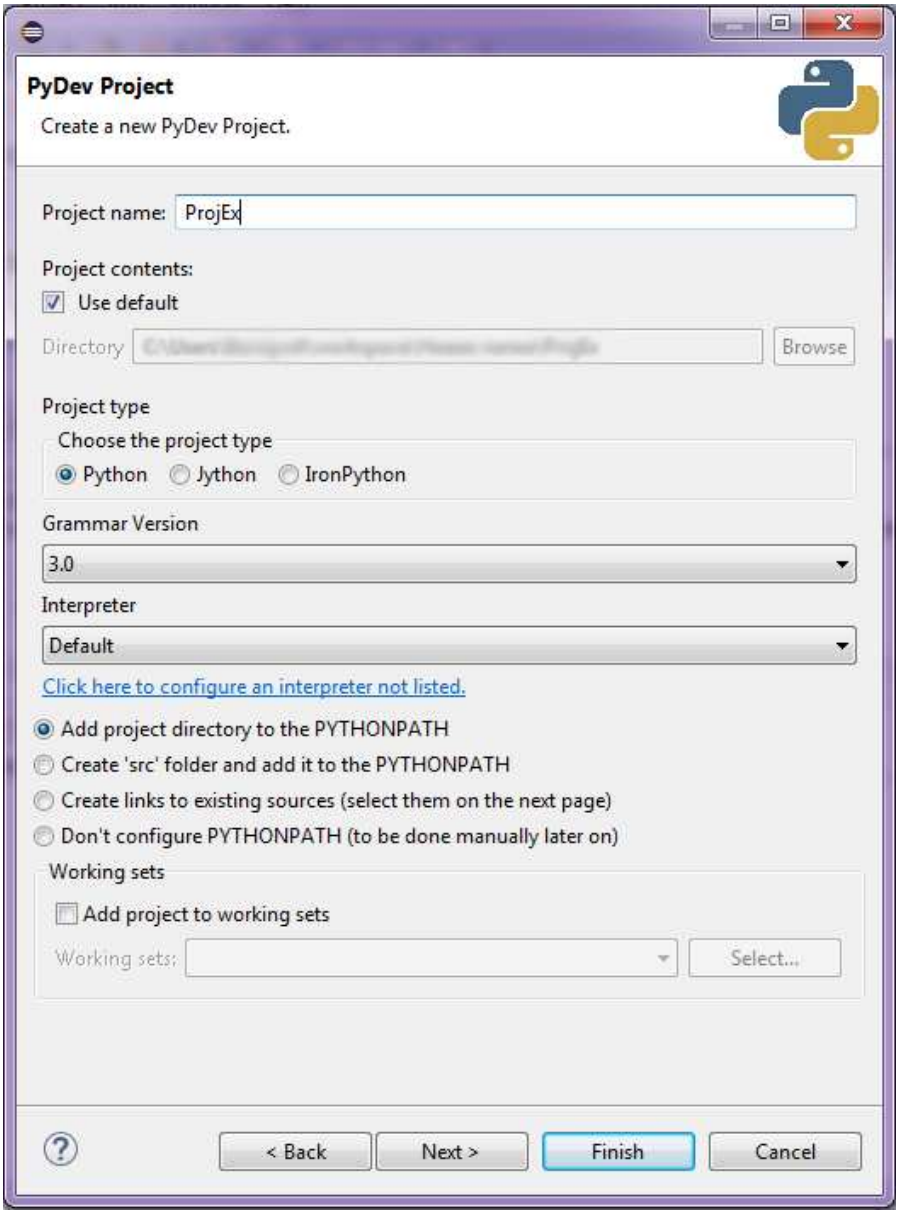

Рисунок 1.1 – Створення нового проекту PyDev

1.2.2 Основні типи мови програмування Python

Python має потужну довідкову систему. Довідкові дані можна отримати як за допомогою документації Python, доступної зокрема з меню IDLE, так і за допомогою відповідних функцій через інтерпретатор.

Функція dir повертає перелік всіх доступних атрибутів заданого об'єкту. Для того щоб отримати інформацію про призначення того чи іншого методу, необхідно передати його ім'я функції help.

У мові Python використовується динамічна типізація: типи даних визначаються автоматично, змінну не потрібно оголошувати.

У таблиці 1.1 представлені основні вбудовані типи об'єктів мови програмування Python.

Таблиця 1.1 – Основні вбудовані об'єкти в мові програмування Python

Тип об'єкту	**Приклад літералу/створення**
Числа	1234, 3.1415, 3+4j, Decimal, Fraction
Рядки	'spam', "guido's" , b'a\x01c'
Переліки	[1, [2, 'three'], 4]
Словники	{'food': 'spam', 'taste': 'yum'}
Кортежі	(1,'spam', 4, 'U')
Файли	myfile = open('eggs', 'r')
Множини	set('abc'), {'a', 'b', 'c'}
Інші базові типи	Самі типи, None, логічні значення
Типи структурних елементів програм	Функції, модулі, класи

Таблиця 1.1 містить далеко не повний перелік об'єктів, тому що об'єктами є всі дані, оброблення яких виконується в програмах мовою Python.

Приклад програми з використанням об'єктів числових та рядкових типів:

Python

```
tax = 12.5 / 100
price = 100.50
sum = price * tax
s='result:'
print(s+str(sum))
```

У таблиці 1.2 представлені шаблони виклику деяких вбудованих рядкових методів.

Таблиця 1.2 – Основні рядкові методи

Метод	**Опис**
S.capitalize()	Повертає рядок, в якому перша літера велика, а всі інші – маленькі
S.count(sub [, start [, end]])	Визначає, скільки разів заданий підрядок зустрічається в даному рядку
S.encode([encoding [,errors]])	Кодує рядок в заданому форматі
S.endswith(suffix [, start [, end]])	Перевіряє, чи завершується даний рядок заданим суфіксом
S.find(sub [, start [, end]])	Пошук заданого підрядка в рядку, повертає найменший індекс
S.isalnum() S.isalpha() S.isdecimal() S.islower() S.isnumeric() S.isspace() S.isupper()	Перевіряє, чи всі символи в рядку є алфавітно-цифровими літерами десятковими знаками в нижньому регістрі числовими пробілами у верхньому регістрі

Продовження таблиці 1.2

Метод	**Опис**
S.join(iterable)	Повертає рядок, що формується конкатенацією рядків у об’єкті iterable
S.lower() S.upper()	Повертає рядок, сформований з даного пониженням (підвищенням) регістру
S.replace(old, new [, count])	Повертає рядок, сформований з даного заміною заданих підрядків
S.rfind(sub [,start [,end]])	Пошук заданого підрядка в рядку, повертає найбільший індекс
S.rpartition(sep)	Розділяє даний рядок за останнім входженням заданого підрядка на три частини
S.split([sep [,maxsplit]])	Розділяє рядок на слова за заданим роздільним символом
S.splitlines([keepends])	Розділяє рядок на окремі рядки за символом нового рядка
S.startswith(prefix [, start [, end]])	Перевіряє, чи починається даний рядок з заданого префікса
S.strip([chars])	Повертає рядок з вилученими символами, що співпадають з заданими, на початку та в кінці даного рядка
S.swapcase()	Перетворює в рядку символи у верхньому регістрі на нижній та навпаки

У таблиці 1.3 представлені основні операції, які використовуються під час роботи з переліками.

Таблиця 1.3 – Літерали переліків та операції

Операція	**Інтерпретація**
L = [0, 1, 2, 3]	Елементи переліку з індексами 0..3
L = list(‘spam’)	Створення переліку з ітеровного об’єкту

Продовження таблиці 1.3

Операція	**Інтерпретація**
L = list(range(-4, 4))	Створення переліку з неперервної послідовності цілих чисел
L[i] L[i:j] len(L)	Елемент за індексом Зріз Довжина переліку
L1 + L2 L * 3	Конкатенація переліків Дублювання переліку
L.append(4) L.extend([5,6,7]) L.insert(I, X)	Методи додавання елементів у перелік
L.index(1) L.count()	Методи пошуку
L.sort() L.reverse()	Методи сортування, зміни порядку слідування елементів на зворотній
del L[k] L.pop() L.remove(2) L[i:j] = []	Зменшення переліку

У таблиці 1.4 представлені основні операції, які використовуються під час роботи зі словниками.

Таблиця 1.4 – Літерали словників та операції

Операція	**Інтерпретація**
D = {‘spam’: 2, ‘eggs’: 3}	Створення словника з двох елементів
D = dict(name=’Bob’, age=40) D = dict(zip(keyslist, valslist))	Альтернативні способи створення словників

Продовження таблиці 1.4

Операція	**Інтерпретація**
D[‘eggs’]	Доступ до елементу за ключем
‘eggs’ in D	Перевірка на входження ключа
D.keys() D.values() D.items() D.copy() D.update(D2) D.pop(key)	Методи: перелік ключів, перелік значень, перелік ключів та значень, копіювання, злиття, вилучення
len(D)	Визначення довжини словника
D[key] = 42 del D[key]	Додавання/змінювання ключів, вилучення ключів

У таблиці 1.5 представлені основні інструкції, які використовуються в процесі створення програм мовою Python.

Таблиця 1.5 – Інструкції мови Python

Інструкція	**Роль**	**Приклад**
Присвоювання	Створення посилань	a, *b = ‘good’, ‘bad’, ‘ugly’
Виклик	Виклик функцій	log.write(“spam, ham”)
if/elif/else	Операція вибору	if “python” in text: print(text)
for/else	Обхід послідовності в циклі	for x in mylist: print(x)
while/else	Цикли загального призначення	while X > Y: print(‘hello’)
pass	Порожня інструкція	while True: pass

Продовження таблиці 1.5

Інструкція	**Роль**	**Приклад**
break	Вихід з циклу	while True: if exittest(): break
continue	Перехід на початок циклу	while True: if skiptest(): continue
def	Створення функцій та методів	def f(a, b, c=1, *d): print(a+b+c+d[0])
return	Повернення результату	def f(a, b, c=1, *d): return a+b+c+d[0]
yield	Функції генератори	def gen(n): for i in n: yield i*2
global	Глобальний простір імен	x = ‘old’ def function(): global x, y; x = ‘new’
import	Доступ до модулів	import sys
from	Доступ до атрибутів модулів	from sys import stdin
try/except/finally	Обробка виключень	try: action() except: print(‘action error’)
raise	Збудження виключень	raise endSearch(location)
with/as	Менеджери контексту	with open(‘data’) as myfile: process(myfile)
lambda	Анонімна функція	funcs = [lambda x: x**2, lambda x: x*3]

1.3. Завдання на лабораторну роботу

1.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

1.3.2. Розробити програмне забезпечення у відповідності з індивідуальним завданням, узгодженим з викладачем.

1.3.3. Виконати тестування розробленого програмного забезпечення.

1.3.4. Оформити звіт.

1.3.5. Відповісти на контрольні запитання.

1.4. Зміст звіту

1.4.1. Мета роботи.

1.4.2. Завдання до роботи.

1.4.3. Текст розробленого програмного забезпечення.

1.4.4. Результати тестування: вхідні дані та результати роботи програми.

1.4.5. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз.

1.5. Контрольні запитання

1.5.1. Який інструментарій може використовуватись для розроблення програм мовою Python?

1.5.2. Які парадигми програмування підтримуються мовою Python?

1.5.3. Які принципи лежать в основі програмування мовою Python?

1.5.4. Для яких цілей може використовуватися мова Python?

1.5.5. Які основні типи даних мови програмування Python?

1.5.6. Чим відрізняється динамічна типізація від статичної?

1.5.7. Чим відрізняється імперативна парадигма програмування від декларативної?

1.5.8. Які типи даних у Python є незмінними та яким чином змінити їх значення?

1.5.9. Які типи даних у Python можуть змінюватись?

1.5.10. Яким чином виконати форматування рядка?

1.5.11. Що таке перелік та чим він відрізняється від інших типів даних?

1.5.12. Які дії можна виконувати над переліками?

1.5.13. Що таке множина та яким чином її визначити?

1.5.14. Що таке словник та чим він відрізняється від інших типів даних?

1.5.15. Які дії можна виконувати над словниками?

1.5.16. Що таке кортеж та у яких випадках його необхідно використовувати?

1.5.17. Яким чином визначається цикл з постумовою у мові Python?

1.5.18. Яким чином визначити у Python аналог оператора switch у C/C++?

1.5.19. Яким чином визначити функцію у Python?

1.5.20. Чи може кількість параметрів у функції змінюватись?

1.5.21. Які особливості функцій у мові Python порівняно з мовою C++?

1.5.22. Яким чином визначити рекурсію в Python?

1.5.23. Чим відрізняється синтаксис мови Python від C/C++?

1.5.24. Яким чином виконується робота з файлами в Python?

1.5.25. Які існують області видимості в Python?

1.5.26. Яким чином згенерувати перелік або словник?

1.5.27. Що таке анонімна функція?

1.5.28. Чим відрізняється робота з функціями в мовах Python та C/C++?

1.5.29. Яким чином реалізується парадигма функціонального програмування в Python?

1.5.30. Яким чином виконується обробка виключень?

ЛАБОРАТОРНА РОБОТА № 2
РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ГРАФІЧНИМ ІНТЕРФЕЙСОМ НА ОСНОВІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ МОВОЮ PYTHON

2.1. Мета роботи

2.1.1 Ознайомитись з принципами реалізації об'єктно-орієнтованого програмування у мові Python та навчитись використовувати його для розроблення програмного забезпечення.

2.1.2 Навчитися розробляти сучасні графічні інтерфейси користувача для програм, написаних мовою Python.

2.2. Короткі теоретичні відомості

2.2.1 Об'єктно-орієнтоване програмування у мові Python

Для визначення класів у Python використовується наступна загальна форма:

Python

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Приклад оголошення класу:

Python

```
class MyClass:
    """A simple example class"""
    i = 12345
```

```python
def f(self):
    return 'hello world'
```

Метод для створення екземпляра-об’єкта може оголошуватись наступним чином:

```python
def __init__(self):
   self.data = []
```

Для організації наслідування використовується наступний синтаксис:

```python
class DerivedClassName(BaseClassName):
  <statement-1>
   .
   .
   .
   <statement-N>
```

2.2.2 Розроблення графічних інтерфейсів користувача програм мовою Python

tkinter – бібліотека для розроблення графічних інтерфейсів, що розповсюджується з відкритим вихідним кодом, яка стала стандартом для розроблення переносних графічних інтерфейсів мовою Python. Сценарії мовою Python, що використовують tkinter для побудови GUI, виконуються у Windows, X Window (Unix і Linux) та Macintosh OS X, створюючи на кожній з даних платформ графічний інтерфейс з властивим їм зовнішнім виглядом. Бібліотека може легко

розширюватись програмним кодом мовою Python, а також має багато додаткових пакетів: Pmw (стороння бібліотека віджетів), Tix (бібліотека віджетів, стандартна частина Python), PIL (розширення для оброблення зображень) і ttk (бібліотека віджетів Tk, що підтримує теми оформлення та стала стандартною частиною Python).

Існують і інші способи створення інтерфейсів програм, написаних мовою Python. Серед них зокрема і PyQt – інтерфейс Python до бібліотеки Qt.

Клас Tk є базовим і викликається за допомогою конструктора tkinter.Tk(screenName=None, baseName=None, className='Tk', useTk=1), що створює віджет верхнього рівня, який зазвичай є головним вікном додатку.

tkinter є бібліотекою, орієнтованою на події, в яких є головний цикл оброблення подій. Для запуску такого циклу використовується метод mainloop, а для виходу – метод quit.

До віджетів бібліотеки tkinter належать:

– Toplevel – вікно верхнього рівня, що зазвичай використовується під час створення багатовіконних програм;

– Button – кнопка;

– Label – мітка (напис без можливості редагування);

– Entry – віджет, що дозволяє ввести один рядок тексту;

– Text – віджет для введення багаторядкового тексту;

– Listbox – перелік, з якого можна обрати один або декілька елементів;

– Frame – рамка для організації віджетів всередині вікна;

– Checkbutton – віджет, що дозволяє вибрати деякий пункт у вікні;

– Radiobutton – дозволяє обрати тільки один пункт у вікні;

– Scale – віджет, що дозволяє вибрати значення з деякого діапазону;

– Scrollbar – віджет, що дозволяє прокручувати інший відмет;

– Menu – віджет для створення меню, що спливає або випадає;

– Menubutton – кнопка з меню;

– Canvas – основа для виведення графічних примітивів;

– Message – віджет аналогічний Label, що дозволяє розташовувати багаторядкові тексти;

– Spinbox – віджет, який дозволяє вибирати значення з заданої множини (поле та пара стрілок);

– PanedWindow – віджет, який дозволяє розділити простір та є контейнером для віджетів-нащадків;

– LabelFrame – прямокутна ділянка з написом, яка може містити інші віджети;

– OptionMenu – віджет, який надає фіксовану множину варіантів вибору у випадаючому меню.

Створення віджетів відбувається наступним чином (приклад для кнопки):

Python

```
w = tkinter.Button(parent, option=value, ...)
```

Перелік налаштувань для кнопки зокрема включає наступні:

– command – функція або метод, які викликаються, коли натискають на кнопку (використовується також для відповідних дій у наступних віджетах: Checkbutton, Radiobutton, Spinbox, Scrollbar, Scale);

– height – висота кнопки;

– width – ширина кнопки;

– text – текст, який відображається на кнопці;

– image – зображення, яке відображається на кнопці;

– bg – колір фону;

– fg – колір шрифту тексту на кнопці;

– font – шрифт тексту на кнопці.

Усі віджети мають наступні спільні методи:

– configure, config – метод, який використовується для конфігурування під час виконання програми;

– destroy – видалення віджета та його нащадків;

– focus_* – сімейство методів для керування фокусом введення з клавіатури (віджет, що має фокус, отримує всі події з клавіатури);

– update і update_idletasks – методи для роботи з чергою задач, виконання яких викликає оброблення відкладених задач;

– winfo_* – сімейство методів, які повертають значення відповідних властивостей віджетів (winfo_children(),winfo_class(), winfo_height(), winfo_name(), winfo_parent(), winfo_id(), winfo_width()), winfo_x(), winfo_y()).

Більш детально з повним переліком методів та властивостей кожного віджета можна зокрема ознайомитись у [35].

Для розміщення віджетів у вікні використовуються менеджери розташування (пакувальники). У Tkinter є 3 пакувальники: pack, place, grid.

Пакувальник pack() є найбільш актуальним. За допомогою властивості side ("left"/"right"/"top"/"bottom") можна вказати, до якої сторони батьківського віджета даний віджет повинен прилягати, а за допомогою властивості fill (None/"x"/"y"/"both") – чи необхідно розширювати простір, наданий віджету. Властивість expand (True/False) визначає, чи необхідно розширювати віджет, щоб він зайняв весь необхідний йому простір. Властивість in_ визначає, у який батьківський віджет помістити даний віджет.

Пакувальник grid() представляє собою таблицю з комірками, в яких розташовуються віджети. Має наступні аргументи:

– row – номер рядка, в якому необхідно розташувати віджет;

– rowspan – кількість рядків, яку займає віджет;

– column – номер стовпця, в якому необхідно розташувати віджет;

– columnspan – кількість стовпців, яку займає віджет;

– padx / pady – розмір зовнішнього бордюра;

– ipadx / ipady – розмір внутрішнього бордюра;

– sticky – визначає, яким чином розподілити зайвий простір;

– in_ – визначає, в який батьківський віджет помістити даний.

У бібліотеку віджетів ttk включені наступні віджети, які можна використовувати замість відповідних віджетів tk: Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale та Scrollbar.

У ttk включені також наступні нові віджети:

– Combobox – текстове поле з випадаючим переліком значень;

– Notebook – віджет, який керує колекцією вікон та відображає одне за раз. Будь-яке дочірнє вікно зв’язано з вкладкою, яку користувач може вибрати, щоб змінити вікно, яке відображається;

– Progressbar – віджет, який відображає статус довготривалих операцій;

– Separator – вертикальна або горизонтальна полоса розподілу;

– Sizegrip – віджет, який дозволяє користувачу змінити розмір вікна верхнього рівня;

– Treeview – ієрархічна колекція пунктів.

Усі ці класи віджетів є підкласами класу Widget.

Більш детально з повним переліком методів та властивостей розглянутих віджетів можна ознайомитись у [34].

Розглянемо приклад побудови графічного інтерфейсу для програми:

Python

```
from tkinter import *
from tkinter import ttk
root = Tk()
root.title('Example')
content = ttk.Frame(root, padding=(3,3,12,12))
frame = ttk.Frame(content, borderwidth=5,
relief="sunken", width=200, height=100)
namelbl = ttk.Label(content, text="Name")
name = ttk.Entry(content)
onevar = BooleanVar()
twovar = BooleanVar()
threevar = BooleanVar()
one = ttk.Checkbutton(content, text="One",
variable=onevar, onvalue=True)
two = ttk.Checkbutton(content, text="Two",
variable=twovar, onvalue=True)
three = ttk.Checkbutton(content, text="Three",
variable=threevar, onvalue=True)
ok = ttk.Button(content, text="OK")
cancel = ttk.Button(content, text="Cancel")
content.grid(column=0, row=0, sticky=(N, S, E, W))
frame.grid(column=0, row=0, columnspan=3, rowspan=2,
sticky=(N, S, E, W))
```

```
namelbl.grid(column=3, row=0, columnspan=2, sticky=(N,
W), padx=5)
name.grid(column=3, row=1, columnspan=2, sticky=(N, E,
W), pady=5, padx=5)
one.grid(column=0, row=3)
two.grid(column=1, row=3)
three.grid(column=2, row=3)
ok.grid(column=3, row=3)
cancel.grid(column=4, row=3)
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
content.columnconfigure(0, weight=3)
content.columnconfigure(1, weight=3)
content.columnconfigure(2, weight=3)
content.columnconfigure(3, weight=1)
content.columnconfigure(4, weight=1)
content.rowconfigure(1, weight=1)
root.mainloop()
```

На рис. 2.1 наведено інтерфейс, що відповідає даній програмі.

Рисунок 2.1 – Приклад графічного інтерфейсу

2.3. Завдання на лабораторну роботу

2.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

2.3.2. Розробити програмне забезпечення з графічним інтерфейсом користувача та ієрархією об’єктів у відповідності з індивідуальним завданням, узгодженим з викладачем.

Оголошення об’єктів винести в окремий модуль. Виконати документування коду для кожного модуля.

2.3.3. Виконати тестування розробленого програмного забезпечення.

2.3.4. Оформити звіт.

2.3.5. Відповісти на контрольні запитання.

2.4. Зміст звіту

2.4.1. Мета роботи.

2.4.2. Завдання до роботи.

2.4.3. Текст розробленого програмного забезпечення.

2.4.4. Результати тестування: вхідні дані, скрипти та результати роботи програми.

2.4.5. Інтерфейс роботи з програмою в декількох режимах.

2.4.6. Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

2.5. Контрольні запитання

2.5.1. Що таке модуль та пакет?

2.5.2. Яким чином файли трансформуються в простори імен?

2.5.3. Які існують шаблони проектування?

2.5.4. Які пакети використовуються для побудови графічних інтерфейсів у Python?

2.5.5. Які засоби має бібліотека tkinter?

2.5.6. Яким чином визначається клас у Python?

2.5.7. Яким чином реалізуються принципи об’єктно-орієнтованого програмування в Python?

ЛАБОРАТОРНА РОБОТА № 3
ВИКОРИСТАННЯ БІБЛІОТЕК PYTHON ДЛЯ ВИСОКОПРОДУКТИВНИХ НАУКОВИХ ОБЧИСЛЕНЬ

3.1. Мета роботи

3.1.1 Ознайомитись з основними можливостями бібліотеки NumPy та навчитися використовувати їх на практиці.

3.1.2 Ознайомитись з основними можливостями бібліотеки SciPy та навчитися використовувати їх на практиці для виконання високопродуктивних наукових обчислень.

3.2. Короткі теоретичні відомості

3.2.1 Бібліотека NumPy

Бібліотека NumPy забезпечує в якості основного типу даних N-мірний масив – ndarray. На рис. 3.1 представлено схему взаємодії між трьома фундаментальними об'єктами, що використовуються для представлення даних в *n*-мірному масиві.

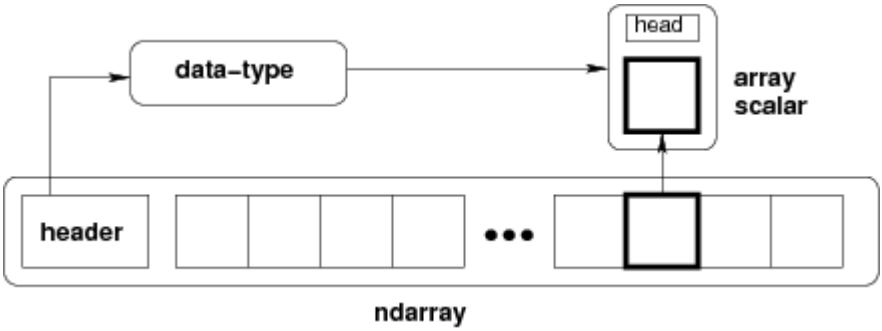

Рисунок 3.1 – Концептуальна схема організації даних у ndarray

Об'єкт ndarray містить елементи однакового типу та розміру. Основні атрибути об'єктів ndarray:

– ndarray.ndim – кількість вимірів масиву (ранг);

– ndarray.shape – розмір масиву, кортеж натуральних чисел, що представляють довжину масиву за кожною віссю;

– ndarray.size – кількість елементів масиву;
– ndarray.dtype – об’єкт, що описує тип елементів масиву;
– ndarray.itemsize – розмір кожного елементу в байтах;
– ndarray.data – буфер, що містить фактичні елементи масиву.

Масив можна утворити за допомогою функції *array* зі звичайних переліків або кортежів:

Python

```
>>> a=array([[2,3,4],[1,2,3]])
```

Окрім того можна утворити масив, заповнений нулями, за допомогою функції zeros, передаючи їй кількість елементів за кожною віссю. Для заповнення масиву одиницями призначена відповідна функція ones.

Функція arrange призначена для заповнення масиву значеннями в заданому інтервалі з заданим кроком:
– arange (x) – значення в інтервалі від 0 до *x*-1;
– arange (x, y) – значення в інтервалі від *x* до *y*;
– arange (x, y, z) – значення в інтервалі від *x* до *y* з кроком *z*.

Функція linspace створює масив з заданою кількістю елементів, розподілених рівномірно між значеннями *x* і *y*: linspace (x, y, num).

Арифметичні операції, функції sin, cos, exp тощо над масивами виконуються поелементно.

Функція diag() добуває діагональ або конструює діагональний масив.

Об’єкти типу ndarray мають велику кількість методів, частина з яких представлена в таблиці 3.1.

Таблиця 3.1 – Методи об’єктів типу ndarray

Метод	**Опис**
ndarray.item(*args)	Копіює елемент масиву в стандартний скаляр та повертає його.
ndarray.tolist()	Повертає масив у вигляді списку.
ndarray.itemset(*args)	Вставляє скалярне значення у масив.
ndarray.tofile(fid[, sep,	Записує масив у файл в текстовому

Продовження таблиці 3.1

Метод	**Опис**
format])	або двійковому вигляді.
ndarray.dump(file)	Виконує дамп масиву у вказаний файл
ndarray.dumps()	Виконує дамп масиву в рядок.
ndarray.astype(dtype[, order, casting, ...])	Повертає копію масиву, змінюючи тип.
ndarray.copy([order])	Повертає копію масиву.
ndarray.view([dtype, type])	Перетворює масив заданим чином, не змінюючи дані, та створює новий масив.
ndarray.fill(value)	Заповнює масив скалярними значеннями.
ndarray.reshape(shape[, order])	Повертає масив зі зміненими розмірами.
ndarray.resize(new_shape[, refcheck])	Змінює розмір масиву на місці.
ndarray.transpose(*axes)	Повертає транспонований масив.
ndarray.swapaxes(axis1, axis2)	Міняє місцями дві вісі масиву.
ndarray.ravel([order])	Повертає одномірний масив.
ndarray.take(indices[, axis, out, mode])	Повертає масив, сформований з елементів за заданими індексами.
ndarray.repeat(repeats[, axis])	Повторює елементи масиву.
ndarray.sort([axis, kind, order])	Сортує масив, виконуючи операцію на місці.
ndarray.argsort([axis, kind, order])	Повертає індекси елементів за розташуванням у відсортованому масиві.
ndarray.partition(kth[, axis, kind, order])	Змінює порядок елементів у масиві таким чином, що елемент зі вказаним індексом розташовується таким чином, що перед ним розташовані елементи з меншим значенням, а правіше – з рівним або більшим.

Продовження таблиці 3.1

Метод	**Опис**
ndarray.argmax([axis, out])	Повертає індекси елементів з максимальним значенням за заданою віссю.
ndarray.min([axis, out])	Повертає мінімум вздовж заданої вісі.
ndarray.argmin([axis, out])	Повертає індекси мінімальних значень вздовж заданої вісі.
ndarray.ptp([axis, out])	Повертає діапазон значень (maximum - minimum) вздовж заданої вісі.
ndarray.clip(a_min, a_max[, out])	Повертає масив, значення елементів якого обмежені значеннями [a_min, a_max].
ndarray.round([decimals, out])	Округляє кожний елемент масиву.
ndarray.trace([offset, axis1, axis2, dtype, out])	Повертає суму елементів вздовж діагоналей.
ndarray.sum([axis, dtype, out])	Повертає суму елементів вздовж заданої вісі.
ndarray.cumsum([axis, dtype, out])	Повертає кумулятивну суму елементів вздовж заданої вісі.
ndarray.mean([axis, dtype, out])	Повертає середнє елементів масиву вздовж заданої вісі.
ndarray.var([axis, dtype, out, ddof])	Повертає дисперсію елементів масиву вздовж заданої вісі.
ndarray.std([axis, dtype, out, ddof])	Повертає стандартне відхилення елементів масиву вздовж даної вісі.
ndarray.prod([axis, dtype, out])	Повертає добуток елементів за заданою віссю.
ndarray.cumprod([axis, dtype, out])	Повертає кумулятивний добуток елементів за заданою віссю.

Функція numpy.mat(data, dtype=None) представляє введені дані у вигляді матриці. Об’єкт матриця має зокрема наступні атрибути, що спрощують роботу з ними:

– A – повертає результат у вигляді об’єкту ndarray;

– T – повертає транспоновану матрицю;

Створити такий об’єкт можна наступним чином:

Python

```
>>> A = numpy.matrix('1.0 2.0; 3.0 4.0')
```

Бібліотека NumPy представляє нові базові типи для побудови масивів, ієрархія яких представлена на рис. 3.2. Елементи масивів мають ті ж самі атрибути і методи, що й ndarrays.

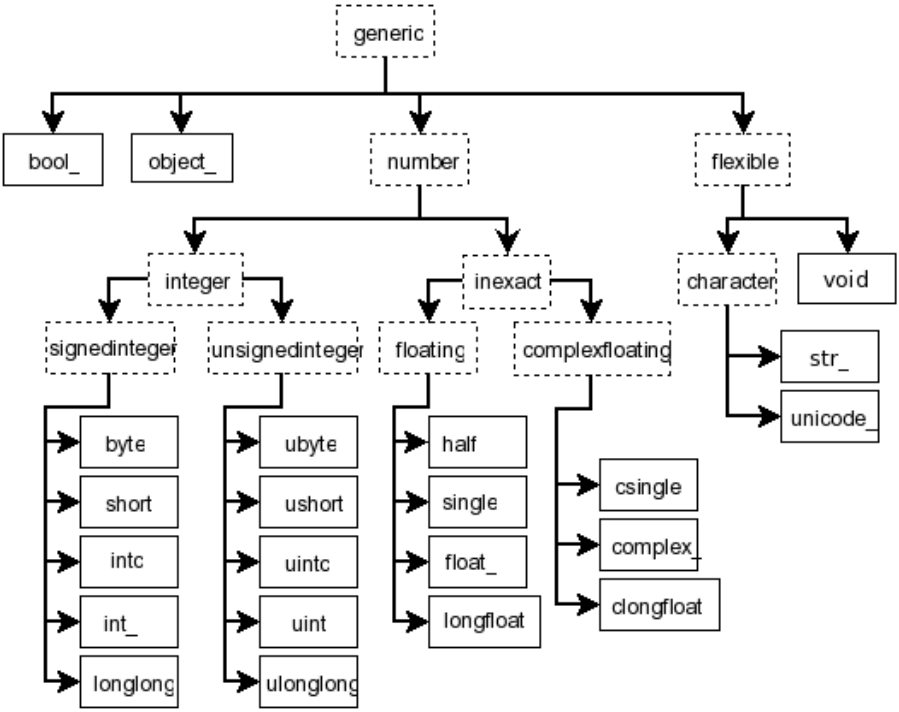

Рисунок 3.2 – Ієрархія типів об’єктів-елементів масиву

Об’єкт типу даних є екземпляром класу numpy.dtype і описує, яким чином інтерпретуються байти у блоці пам’яті фіксованого розміру.

Для створення таких об’єктів використовується конструктор:

Python

```
dt = numpy.dtype(numpy.int32)
```

Створений об’єкт має серед інших наступні атрибути:
– dtype.type – тип об’єкту;
– dtype.char – символьний код типу;
– dtype.num – унікальне число для кожного з типів;
– dtype.itemsize – розмір елементу об’єкту даного типу;
– dtype.byteorder – порядок байтів.

Для ітерації використовується об’єкт numpy.nditer, який має ряд налаштувань, що можуть змінювати процес ітерації. Зокрема параметр flags, який може бути використано в конструкторі, може визначати інший порядок обходу масиву або режим доступу до елементів ітератора.

За допомогою модуля numpy.ma можна створити маски масивів, які дозволяють виділити пропущені або невірні значення. Наприклад, один зі способів, яким такий масив можна створити:

Python

```
import numpy.ma as ma
mx = ma.masked_array(x, mask=[0, 0, 0, 1, 0])
```

Функція copyto(dst, src, casting='same_kind', where=None) дозволяє копіювати дані з одного масиву в інший, виконуючи необхідні перетворення.

Для об’єднання масивів використовуються наступні функції:

– hstack(tup) – для горизонтального об’єднання масивів у послідовність;

– vstack(tup) – для вертикального об’єднання масивів у послідовність;

– concatenate((a1, a2, ...), axis=0) – приєднує послідовність масивів;

– column_stack(tup) – об’єднує 1-D масиви у вигляді колонок у 2-

D масиви.

Для роз'єднання масивів використовуються наступні функції:

– array_split(ary, indices_or_sections[, axis]) – розділяє масив на підмасиви;

– hsplit(ary, indices_or_sections) – розділяє масив на підмасиви горизонтально;

– split(ary, indices_or_sections[, axis]) – розділяє масив на підмасиви;

– vsplit(ary, indices_or_sections) – розділяє масив на підмасиви вертикально.

Для додавання та вилучення елементів призначені наступні функції:

– delete(arr, obj[, axis]) – вилучає елементи з масиву вздовж заданої вісі;

– insert(arr, obj, values[, axis]) – вставляє значення вздовж заданої вісі перед заданими індексами;

– append(arr, values[, axis]) – додає значення в кінець масиву;

– unique(ar[, return_index, return_inverse, ...]) – виділяє унікальні елементи масиву;

– intersect1d(ar1, ar2[, assume_unique]) – знаходить перетин двох масивів;

– setdiff1d(ar1, ar2[, assume_unique]) – знаходить різницю двох масивів у розумінні множин;

– union1d(ar1, ar2) – знаходить об'єднання двох масивів.

Логічні функції включають логічні операції (logical_and(x1, x2[, out]), logical_or(x1, x2[, out]) , logical_not(x[, out]), logical_xor(x1, x2[, out])), перевірки типу елементів масиву (iscomplex(), isreal()), порівняння масивів (array_equal(a1, a2), greater(x1, x2[, out]), greater_equal(x1, x2[, out]), less(x1, x2[, out]), less_equal(x1, x2[, out])).

Бібліотека містить набір функцій лінійної алгебри (модуль numpy.linalg):

– dot(a, b[, out]) – матричний добуток двох масивів;

– vdot(a, b) – добуток двох векторів;

– matrix_power(M, n) – підносить квадратну матрицю у заданий ступінь;

– solve(a, b) – розв'язує лінійне матричне рівняння або систему лінійних рівнянь.

3.2.2 Бібліотека SciPy

Бібліотека SciPy побудована на основі NumPy та надає потужний науковий інструментарій.

Бібліотека SciPy зокрема містить наступні пакети:

– спеціальних функцій (scipy.special);

– чисельного розв'язання звичайних диференційних рівнянь (scipy.integrate);

– оптимізації (scipy.optimize);

– інтерполяції (scipy.interpolate);

– перетворення Фур'є (scipy.fftpack);

– оброблення сигналів (scipy.signal);

– лінійної алгебри (scipy.linalg);

– розріджених графів (scipy.sparse.csgraph);

– просторові структури даних та алгоритми (scipy.spatial);

– випадкових чисел та статистики (scipy.stats);

– багатовимірного оброблення зображень (scipy.ndimage);

– файлового вводу-виводу (scipy.io);

– включення коду C/C++ (scipy.weave);

– алгоритмів кластеризації (scipy.cluster);

– розріджених матриць (scipy.sparse).

Бібліотека scipy.linalg містить функції лінійної алгебри:

– inv(a[, overwrite_a, check_finite]) – обчислює зворотну матрицю;

– solve(a, b[, sym_pos, lower, overwrite_a, ...]) – розв'язує рівняння $a\ x = b$ для x;

– det(a[, overwrite_a, check_finite]) – обчислює детермінант матриці;

– lstsq(a, b[, cond, overwrite_a, ...]) – обчислює розв'язання рівняння $Ax = b$ методом найменших квадратів.

Підпакет scipy.integrate зокрема включає наступні функції:

– quad(func, a, b[, args, full_output, ...]) – обчислює визначений інтеграл;

– dblquad(func, a, b, gfun, hfun[, args, ...]) – обчислює подвійний інтеграл;

– tplquad(func, a, b, gfun, hfun, qfun, rfun) – обчислює потрійний інтеграл;

– nquad(func, ranges[, args, opts]) – інтегрування за декількома змінними.

Пакет оптимізації містить функції локальної, глобальної оптимізації, Розенброка, лінійного програмування тощо.

Функції локальної оптимізації включають:

– minimize(fun, x0[, args, method, jac, hess, ...]) – мінімізує скалярну функцію однієї або більше змінних одним з методів, представлених параметром method (‘Nelder-Mead’, ‘Powell’, ‘CG’, ‘BFGS’, ‘Newton-CG’, ‘Anneal’, ‘L-BFGS-B’, ‘TNC’, ‘COBYLA’, ‘SLSQP’, ‘dogleg’, ‘trust-ncg’, custom);

– minimize_scalar(fun[, bracket, bounds, ...]) – мінімізує скалярну функцію однієї змінної за допомогою методу, визначеного параметром method (‘Brent’, ‘Bounded’, ‘Golden’, custom);

– OptimizeResult – представляє результати оптимізації.

Функції глобальної оптимізації включають:

– basinhopping(func, x0[, niter, T, stepsize, ...]) – знаходить глобальний мінімум функції, використовуючи стохастичний алгоритм Basin-Hopping;

– differential_evolution(func, bounds[, args, ...]) – знаходить глобальний мінімум багатовимірної функції за допомогою методу диференційної еволюції.

Функція curve_fit(f, xdata, ydata[, p0, sigma, ...]) використовує нелінійний метод найменших квадратів для мінімізації відхилень функції від даних.

Функція linprog(c[, A_ub, b_ub, A_eq, b_eq, bounds, ...]) мінімізує лінійну цільову функцію з обмеженнями у вигляді нерівностей та лінійних рівностей на основі лінійного програмування.

Функція leastsq(func, x0[, args, Dfun, full_output, ...]) мінімізує суму квадратів множини рівнянь.

Бібліотека інтерполяції scipy.interpolate містить сплайн-функції і класи, одновимірні і багатовимірні інтерполяційні класи, поліноміальні інтерполятори Лагранжа lagrange(x, w) і Тейлора approximate_taylor_polynomial(f, x, degree, ...), оболонки для функцій FITPACK і DFITPACK.

Одномірна інтерполяція містить наступні функції:

– interp1d(x, y[, kind, axis, copy, ...]) – дозволяє інтерполювати одновимірну функцію;

– barycentric_interpolate(xi, yi, x[, axis]), krogh_interpolate(xi, yi, x[, der, axis]) – допоміжна функція для поліноміальної інтерполяції.

Багатомірна інтерполяція включає наступні функції:

– griddata(points, values, xi[, method, ...]) – інтерполювати неструктуровані D-вимірні дані;

– LinearNDInterpolator(points, values[, ...]) – кусково-лінійний інтерполянт у N вимірах;

– NearestNDInterpolator(points, values) – інтерполяція методом найближчого сусіда в N вимірах;

– interp2d(x, y, z[, kind, copy, ...]) – інтерполювати по двовимірній сітці;

– interpn(points, values, xi[, method, ...]) – багатовимірна інтерполяція на регулярних сітках.

Одновимірні сплайни включають:

– UnivariateSpline(x, y[, w, bbox, k, s, ext]) – одновимірний згладжуючий сплайн, що відповідає даній множині точок даних;

– InterpolatedUnivariateSpline(x, y[, w, ...]) – одновимірний інтерполяційний сплайн для даної множини точок даних.

Вищенаведені класи одномірних сплайнів мають наступні методи:

– UnivariateSpline.__call__(x[, nu, ext]) – оцінює сплайн в точці *x*;

– UnivariateSpline.derivatives(x) – повертає всі похідні сплайна в точці *x*;

– UnivariateSpline.get_coeffs() – повертає коефіцієнти сплайна.

Пакет алгоритмів кластеризації містить два модуля: vq, який підтримує векторне квантування і алгоритм k-середніх, та hierarchy, який мість функції для ієрархічної кластеризації.

Модуль scipy.cluster.vq включає наступні функції:

– whiten(obs) – нормалізує групу спостережень;

– kmeans(obs, k_or_guess[, iter, thresh]) – виконує алгоритм k-середніх на множині векторів спостережень, формуючи k кластерів;

– kmeans2(data, k[, iter, thresh, minit, missing]) – класифікує множину спостережень на k-кластерів, використовуючи алгоритм k-середніх.

Статистичні функції та розподіли ймовірностей містяться в підпакеті scipy.stats.

Статистичні функції підпакету включають обчислення декількох описових статистик переданого масиву describe(a[, axis, ddof]), середньогеометричного вздовж заданої вісі gmean(a[, axis, dtype]), середньогармонічного вздовж заданої вісі hmean(a[, axis, dtype]), ексцесу (Фішера або Пірсона) множини даних kurtosis(a[, axis, fisher, bias]), моди mode(a[, axis]), перевірку множини даних на нормальний ексцес kurtosistest(a[, axis]), нормальний розподіл normaltest(a[, axis]), асиметрії на відмінність від нормального розподілу skewtest(a[, axis]), обчислення n-го моменту середнього moment(a[, moment, axis]), асиметрії skew(a[, axis, bias]), коефіцієнту варіації variation(a[, axis]) тощо.

Бібліотека scipy.ndimage містить функції багатовимірного оброблення зображень: фільтри scipy.ndimage.filters (багатовимірний фільтр з гаусівською характеристикою gaussian_filter(input, sigma[, order, ...]) і одновимірний gaussian_filter1d(input, sigma[, axis, ...]), N-вимірний фільтр Лапласа generic_laplace(input, derivative2[, ...]) тощо), фільтри Фур'є scipy.ndimage.fourier, інтерполяції scipy.ndimage.interpolation, вимірювання scipy.ndimage.measurements, морфології scipy.ndimage.morphology.

3.2.3 Бібліотека matplotlib

Бібліотека matplotlib призначена для побудови 2D графіків, написана мовою Python.

matplotlib.pyplot – колекція функцій у командному стилі, що дозволяють працювати в стилі MATLAB.

Функції xlabel, ylabel додають мітки на відповідні вісі координат (matplotlib.axes.Axes.set_xlabel() через API);

Функція text додає текст на графік за вказаними координатами (x, y, text). Функція figtext додає текст на графік у відносних координатах (0-1).

Функція annotate('annotation', xy=(x1, y1), xytext=(x2, y2)) дозволяє вставити анотацію для зазначеної точки графіка.

Функція legend() виводить підпис для відповідних ліній на графіку.

Функція plot() відображає лінії і/або маркери на графіку. Наприклад:

Python

```
plot(x, y, 'bo')
```

Рядок формату задається символами, що задають стиль лінії (таблиця 3.2) та колір (таблиця 3.3).

Таблиця 3.2 – Представлення стилю ліній у рядку формату

Символ	Опис	Символ	Опис
'-'	суцільна лінія	'3'	маркер у вигляді тріода, направленого вліво
'--'	пунктирна лінія	'4'	маркер у вигляді тріода, направленого вправо
'-.'	штрихпунктирна лінія	's'	квадратний маркер
':'	пунктирна лінія	'p'	п'ятикутний маркер
'.'	маркер у вигляді крапки	'*'	маркер у вигляді зірочки
','	маркер у вигляді пікселя	'h'	маркер у вигляді шестикутника 1
'o'	маркер у вигляді кола	'H'	маркер у вигляді шестикутника 2
'v'	маркер у вигляді знаку ‘v’	'+'	маркер у вигляді знаку ‘+’
'^'	маркер у вигляді знаку ‘^’	'x'	маркер у вигляді знаку ‘x’
'|'	маркер у вигляді вертикальної лінії	'D'	маркер у вигляді ромба
'_'	маркер у вигляді горизонтальної лінії	'd'	маркер у вигляді малого ромба

Таблиця 3.3 – Представлення кольору у рядку формату

Символ	Колір	Символ	Колір
‘b’	Синій	‘m’	Пурпурний
‘g’	Зелений	‘y’	Жовтий
‘r’	Червоний	‘k’	Чорний
‘c’	Блакитний	‘w’	Білий

Функція hist() призначена для побудови гістограм та приймає в якості аргументів масив даних і кількість ділянок, на які буде розбито масив (за замовчуванням дорівнює 10).

Для побудови стовпчастих діаграм призначена функція bar(), що приймає в якості аргументів послідовності координат *x* (лівий край стовпця) та *y* (висота). Ширина прямокутників за замовчуванням дорівнює 0.8. Параметри можна задати за допомогою наступних ключових слів: width (ширина прямокутника), color (колір прямокутника). Для горизонтального зображення використовується функція barh().

Функція pie() призначена для побудови кругових діаграм та в якості аргументів приймає послідовність даних і параметри, які можна задавати за допомогою ключових слів: colors (кольори сегментів), labels (імена сегментів) тощо.

Функція show() відображає побудований графік.

Функції ylim() і xlim() встановлює межі вісі ординат, передаючи мінімальне та максимальні значення або за ключовими словами (ymin, ymax). Без аргументів функція повертає поточні межі вісі.

Функція vlines(x, ymin, ymax, colors=u'k', linestyles=u'solid', label=u'', hold=None, **kwargs) дозволяє будувати вертикальні лінії на графіку з абсцисою *x* та ординатами від *ymin* до *ymax*. Для горизонтальних ліній існує відповідна функція hlines().

Функція subplots(nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw) дозволяє створити графічне зображення, на якому розташовано декілька графіків. Параметри nrows та ncols задають кількість рядків та стовпців у сітці графіків. Функція повертає значення fig, об’єкт matplotlib.figure.Figure та ax, об’єкти підграфіків у заданій кількості.

Функція subplot() повертає новий підграфік за заданою позицією на сітці графіків.

Для того щоб задати властивості об’єкту використовується функція setp(). Властивості задаються за допомогою ключових слів, а першим параметром задається безпосередньо об’єктом.

Функція savefig() дозволяє зберегти поточну фігуру. Перший параметр задає шлях до файлу.

Функція fill_between(x, y1, y2=0, where=None, interpolate=False, hold=None, **kwargs) дозволяє заповнити багатокутники між двома кривими. Функція fill_betweenx() відповідно заповнює простір між двома горизонтальними кривими. Функція fill() відображає заповнені багатокутники.

За допомогою функції rc() можна визначити необхідні налаштування. Такі налаштування за замовчуванням можна задати за допомогою файлу matplotlibrc.

Функція figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, **kwargs) створює нову фігуру.

Функція close() закриває вікно з фігурою, а clf() очищує поточну фігуру.

matplotlib підтримує вирази TeX у будь-яких текстових виразах. Наприклад, щоб внести у якості назви напис $\sigma_i = 15$, необхідно скористатись командою

```Python
plt.title(r'$\sigma_i=15$')
```

Інструментар mplot3d додає можливості 3D-візуалізації у matplotlib, дозволяючи будувати стовпчасті діаграми, графіки розсіювання у трьохмірному вигляді. Для такої візуалізації призначені спеціальні об’єкти Axes3D, які мають відповідні функції plot, scatter, bar, а також plot_surface для побудови графіка поверхні тощо.

3.3. Завдання на лабораторну роботу

3.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

3.3.2. Розробити програмне забезпечення у відповідності з індивідуальним завданням, узгодженим з викладачем.

3.3.3. Виконати тестування розробленого програмного забезпечення.

3.3.4. Оформити звіт.

3.3.5. Відповісти на контрольні запитання.

3.4. Зміст звіту

3.4.1. Мета роботи.

3.4.2. Завдання до роботи.

3.4.3. Текст розробленого програмного забезпечення.

3.4.4. Результати тестування: вхідні дані та результати роботи програми.

3.4.5. Інтерфейс роботи з програмою в декількох режимах.

3.4.6. Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

3.5. Контрольні запитання

3.5.1. Які засоби надаються бібліотекою NumPy?

3.5.2. Які засоби надаються бібліотекою SciPy?

3.5.3. Для чого необхідна бібліотека matplotlib?

3.5.4. Які засоби містить бібліотека matplotlib?

3.5.5. Які графіки можна побудувати за допомогою бібліотеки matplotlib?

ЛАБОРАТОРНА РОБОТА № 4
РОЗРОБЛЕННЯ WEB-ДОДАТКІВ ТА РЕАЛІЗАЦІЯ ДОСТУПУ ДО СИСТЕМ КЕРУВАННЯ БАЗАМИ ДАНИХ ЧЕРЕЗ ПРОГРАМНІ ІНТЕРФЕЙСИ

4.1. Мета роботи

4.1.1 Навчитися використовувати програмні інтерфейси для доступу до баз даних.

4.1.2 Навчитися розробляти web-додатки за допомогою фреймворка Django.

4.2. Короткі теоретичні відомості

4.2.1 Програмний інтерфейс для роботи з системою керування базами даних MySQL

Модуль MySQLdb призначений для забезпечення програмного інтерфейсу для використання системи керування базами даних MySQL у програмах, написаних мовою Python. Програмний інтерфейс аналогічний даному модулю, написаний на Python та призначений для інтерпретаторів CPython, PyPy, IronPython, Jython, реалізовано в пакеті PyMySQL.

Функція connect(parameters...) – конструктор для створення з'єднання до бази даних, який повертає відповідний об'єкт. Параметри функції зокрема включають ім'я хосту host, з яким встановлюється з'єднання, логін user і пароль passwd для аутентифікації, назву бази даних db, з якою встановлюється з'єднання, port – TCP-порт MySQL сервера.

Приклад з'єднання з базою даних MySQL:

Python

```
import MySQLdb
mydb = MySQLdb.connect(host = 'localhost', user =
'skipper', passwd = 'mysecret', db = 'fish')
```

Об’єкти з’єднання містять наступні методи:

– commit(): якщо база даних та таблиці підтримують транзакції, то фіксує поточну транзакцію;

– rollback(): відхиляє поточну транзакцію;

– cursor([cursorclass]): створює новий курсор для виконання запитів;

– begin(self): починає транзакцію;

– close(self): закриття з’єднання з базою даних;

– ping(self, reconnect=True): пінгувати сервер;

– select_db(self, db): встановлює поточну базу даних.

Об’єкти класу Cursor мають наступні методи:

– callproc(self, procname, args=()) – виконати збережену процедуру з заданими аргументами;

– close(self) – закрити курсор;

– execute(self, query, args=None) – виконати запит;

– executemany(self, query, args) – виконує один запит для декількох наборів даних;

– fetchall(self) – видобуває всі записи;

– fetchmany(self, size=None) – видобуває декілька записів;

– fetchone(self) – видобуває наступний запис;

– nextset(self) – пересуває курсор на наступну множину результатів.

Наприклад, для виконання запиту до таблиці бази даних можна скористатися наступним скриптом:

Python

```
import MySQLdb
db=MySQLdb.connect(passwd="moonpie",db="thangs")
c=db.cursor()
max_price=5
c.execute("""SELECT spam, eggs, sausage FROM
breakfast
        WHERE price < %s""", (max_price,))
```

Для того щоб переглянути один запис з результату запиту, можна виконати наступну команду:

Python

```
>>> c.fetchone()
(3L, 2L, 0L)
```

4.2.2 Фреймворк Django для розроблення web-додатків

Django – популярний потужний фреймворк для розроблення web-додатків, написаних мовою Python, що використовує шаблон проектування MVC.

Після встановлення Django в системі з'явиться скрипт django-admin.py для оброблення задач скафолдінгу. Для того щоб створити файли проекту, необхідно скористатися командою

CMD

```
django-admin.py startproject mysite
```

Створений проект має наступну структуру:

```
mysite/
   manage.py #посилання на скрипт django-admin з наперед
                  #установленими змінними оточення, що вказують
               #на проект
   mysite/
      __init__.py
      settings.py #налаштування проекту
      urls.py #містить URL для відображення представлень
      wsgi.py #WSGI-обгортка додатку
```

Для створення додатку використовується команда:

CMD

```
python ./manage.py startapp polls
```

Django підтримує системи керування базами даних MySQL, PostgreSQL, SQLite3. Для того щоб задати відповідну систему, необхідно встановити відповідні значення у секції DATABASES файла settings.py. У даному файлі задається також перелік встановлених додатків (у перелік необхідно додати створений додаток («‘polls’,»).

Моделі Django відображають таблиці бази даних та надають місце для інкапсуляції бізнес-логіки. Усі моделі є нащадками базового класу django.db.models.Model та містять поля визначень. Наприклад, файл polls/models.py:

Python

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Для завдання відношень використовуються ForeignKey («множина-один»), ManyToManyField («множина-до-множини»), OneToOneField («один-до-одного»):

Python

```
class Manufacturer(models.Model):
```

```
    # ...
    pass

class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer)
```

Для класів можуть бути визначені методи моделі або змінено поводження бази даних за допомогою методів save() та delete(). Як і класи в Python моделі в Django можуть наслідуватися одна від одної.

Метод raw(raw_query, params=None, translations=None) може використовуватися для того, щоб виконувати запити SQL, які повертають екземпляри. Наприклад:

Python

```
Person.objects.raw('SELECT id, first_name, last_name,
birth_date FROM myapp_person')
```

Стандартне з'єднання бази даних повертається за допомогою об'єкта django.db.connection.

Після створення моделі необхідно доповнити базу даних новими таблицями за допомогою команди

CMD

```
python manage.py migrate
```

Для того щоб взаємодіяти зі створеною моделлю за допомогою інтерактивної оболонки необхідно скористатися командою

Python

```
python ./manage.py shell
>>> from polls.models import Question, Choice
```

```
>>> Question.objects.all()
[]
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?",
pub_date=timezone.now())
>>> q.save()
```

Для того щоб запустити вбудований веб-сервер, необхідно скористатися командою

CMD

```
python manage.py runserver
```

Після цього в браузері можна запустити http://127.0.0.1:8000/ та отримати доступ до сервера.

Представлення Django на отриманий запит повертають відповідь. Будь-який об'єкт мови Python може бути представленням. Єдина жорстка і необхідна вимога полягає в тому, що об'єкт Python, який викликається, повинен приймати об'єкт запиту в якості першого аргументу. Це означає, що мінімальне уявлення буде дуже простим:

Python

```
from django.http import HttpResponse

def index(request):
return HttpResponse("Hello, World")
```

Представлення-класи формуються наступним чином:

Python

```
from django.http import HttpResponse
```

```
from django.views.generic import View

class MyView(View):

    def get(self, request, *args, **kwargs):
        return HttpResponse("Hello, World")
```

Існують наступні стандартні представлення:

– базові django.views.generic.base (базовий View, TemplateView, який інтерпретує даний шаблон з використанням параметрів з URL, що містять контекст, RedirectView, що перенаправляє на заданий URL);

– параметризовані django.views.generic (detail.DetailView, що під час виконання у self.object містить об'єкт, над яким виконуються дії, та list.ListView, що є сторінкою, яка представляє перелік об'єктів self.object_list);

– django.views.generic.edit для редагування контенту (FormView, що відображує форму, CreateView, що відображує форму для створення об'єкту, повторного відображення форми та збереження об'єкту, UpdateView, що відображує форму для редагування існуючого об'єкту, повторного відображення форми та збереження змін до об'єкту, DeleteView, що відображує сторінку підтвердження та вилучає існуючий об'єкт) тощо.

Конфігурація URL вказує Django, яким чином за адресою запиту знайти Python-код. Django переглядає конфігурацію URL, яка визначена у змінній urlpatterns файла polls/urls.py:

Python

```
from django.conf.urls import patterns, url
from polls import views

urlpatterns = patterns('',
   url(r'^$', views.index, name='index'),
)
```

Функції url() передається чотири аргументи, два з яких обов'язкові: regex (регулярний вираз) і view (викликає відповідну функцію представлення, якщо задоволено регулярний вираз), а два необов'язкові: kwargs і name (ім'я URL, яке зокрема може використовуватися в шаблонах).

Деякі загальні представлення повинні мати шаблони, в загальному випадку розташовані за адресою ім'я_ додатку/templates/ ім'я_ додатку/ ім'я_шаблону.

Змінні в шаблонах представляються у вигляді {{ variable }}. Фільтри застосовуються у вигляді {{ variable|filter }}. Керуючий код визначається у вигляді {% tag %}.

Розглянемо детальніше приклад, для якого вище було створено модель і який передбачає створення набору опитувань, кожне з яких складається з деякої кількості варіантів, дозволяючи переглянути опитування, його результати та проголосувати.

Текст файлу /mysite/urls.py, де визначаються всі URL, які розпізнає даний проект:

Python

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

urlpatterns = patterns('',
   url(r'^polls/', include('polls.urls', namespace="polls")),
)
```

Текст файлу /mysite/polls/urls.py, який визначає всі URL, що розпізнаються даним додатком:

Python

```
from django.conf.urls import patterns, url
from polls import views

urlpatterns = patterns('',
```

```
    url(r'^$', views.IndexView.as_view(), name='index'),
    url(r'^(?P<pk>\d+)/$', views.DetailView.as_view(),
name='detail'),
    url(r'^(?P<pk>\d+)/results/$',
views.ResultsView.as_view(), name='results'),
    url(r'^(?P<question_id>\d+)/vote/$', views.vote,
name='vote'),
)
```

Текст файлу /mysite/polls/views.py, який визначає всі наявні представлення (перегляду переліку опитувань, конкретного опитування, його результатів та голосування):

Python

```
from polls.models import Question, Choice
from django.http import HttpResponseRedirect,
HttpResponse
from django.template import RequestContext, loader
from django.shortcuts import get_object_or_404, render
from django.core.urlresolvers import reverse
from django.views import generic

class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        """Return the last five published questions."""
        return Question.objects.order_by('-pub_date')[:5]

class DetailView(generic.DetailView):
    model = Question
    template_name = 'polls/detail.html'

class ResultsView(generic.DetailView):
```

```
    model = Question
    template_name = 'polls/results.html'

def vote(request, question_id):
    p = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice =
p.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        return render(request, 'polls/detail.html', {
            'question': p,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        return HttpResponseRedirect(reverse('polls:results',
args=(p.id,)))
```

Текст файлу index.html:

HTML

```
<b>Опитування</b>
    <ul>
    {% for question in latest_question_list %}
        <li><a href="{% url 'polls:detail' question.id %}">{{
question.question_text }}</a></li>
    {% endfor %}
    </ul>
```

На рис. 4.1 представлено результат відображення одного з опитувань.

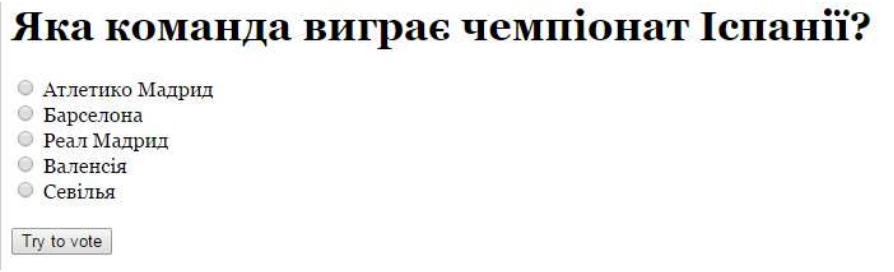

Рисунок 4.1 – Інтерфейс розробленого web- додатку з опитуванням

На рис. 4.2 представлені результати голосування, які з'являються на екрані після вибору варіанту та натиснення на кнопку 'Try to vote'.

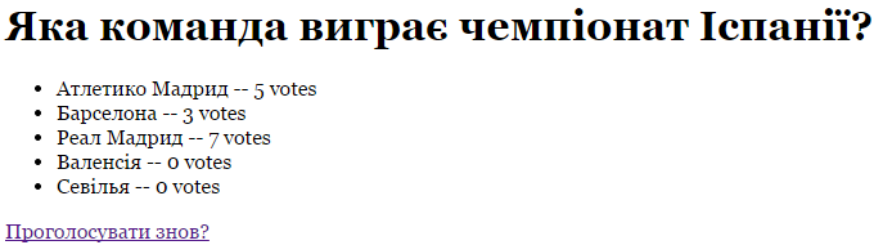

Рисунок 4.2 – Інтерфейс розробленого web- додатку з результатами опитування

Для створення адміністраторського облікового запису для підключення до адміністративної панелі необхідно використовувати команду

CMD

```
python manage.py createsuperuser
```

4.3. Завдання на лабораторну роботу

4.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

4.3.2. Узгодити з викладачем індивідуальне завдання.

4.3.3. Виконати проектування та реалізувати базу даних MySQL у відповідності з індивідуальним завданням.

4.3.4. Розробити web-додаток у відповідності з індивідуальним завданням.

4.3.5. Виконати тестування розробленого програмного забезпечення.

4.3.6. Оформити звіт.

4.3.7. Відповісти на контрольні запитання.

4.4. Зміст звіту

4.4.1. Мета роботи.

4.4.2. Завдання до роботи.

4.4.3. Концептуальна модель бази даних.

4.4.4. Текст програми з основними коментарями.

4.4.5. Інтерфейс роботи з програмою в декількох режимах.

4.4.6. Результати тестування: вхідні дані та результати роботи програми.

4.4.7. Перелік проблем, які були виявлені і розв’язані або не розв’язані під час роботи над проектом, з описом відповідної ситуації. Якщо проблему було розв’язано, то необхідно додатково описати прийняті рішення. Якщо проблему не було розв’язано, ситуацію має бути ідентифіковано максимально детально з поясненням причин.

4.4.8. Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

4.5. Контрольні запитання

4.5.1 Які існують способи збереження даних в програмах, написаних мовою Python?

4.5.2 Для чого призначена бібліотека MySQLdb?

4.5.3 Які існують високорівневі функції бібліотеки MySQLdb?

4.5.4 Яким чином виконати запит до бази даних та яким чином переглянути результати?

4.5.5 Що таке шаблон проектування MVC?

4.5.6 За допомогою яких команд виконується розроблення додатків у бібліотеці Django?

4.5.7 З яких файлів складається проект Django?

4.5.8 Яку структуру мають додатки Django?

4.5.9 Для чого необхідні та яким чином реалізуються моделі Django?

4.5.10 Яким чином визначаються представлення Django?

4.5.11 Для чого необхідні та яким чином визначаються і підключаються шаблони?

4.5.12 Яким чином можна використати статичні ресурси в Django?

ЛАБОРАТОРНА РОБОТА № 5
СИНТАКСИЧНИЙ АНАЛІЗ ДОКУМЕНТА ЗА ДОПОМОГОЮ ПАКЕТІВ ПАРСИНГУ

5.1. Мета роботи

5.1.1 Ознайомитись з іструментарем виконання синтаксичного аналізу документів.

5.1.2. Навчитися використовувати пакети Python для парсингу XML-документів.

5.2. Короткі теоретичні відомості

5.2.1 Модуль re

Для пошуку за шаблонами регулярних виразів призначені засоби модуля re. Даний модуль визначає функції для співставлення, компіляції рядків шаблонів у об’єкти шаблонів, співставлення цих об’єктів з рядками та вибору підрядків, які співпали, після знаходження відповідностей. Він також надає інструменти для виконання розбиття, заміни та інших операцій на основі регулярних виразів. Синтаксис, який використовується для створення шаблонів у даному модулі, представлено в таблиці 5.1.

Таблиця 5.1 – Синтаксис шаблонів re

Оператор	**Опис**
.	Будь-який символ (включаючи переводення рядка, якщо встановлено прапор DOTALL)
^	Початок рядка (кожного рядка в режимі MULTILINE)
$	Кінець рядка (кожного рядка в режимі MULTILINE)
R*	Нуль або більше співпадінь з попереднім регулярним виразом *R* (максимальна можлива кількість)
R+	Одне або більше співпадінь з попереднім регулярним виразом *R* (максимальна можлива кількість)
R?	Нуль або одне співпадіння з попереднім регулярним виразом *R*

Продовження таблиці 5.1

Оператор	**Опис**
R{m}	Точна кількість співпадінь *m* з попереднім регулярним виразом *R*
R{m,n}	Від *m* до *n* співпадінь з попереднім регулярним виразом *R*
R*?, R+?, R??, R{m,n}?	Те ж, що і *, + і ?, але відповідає мінімально можливій кількості співпадінь; відомі як мінімальні (нежадібні) квантифікатори (на відміну від інших, шукають та поглинають мінімально можливу кількість символів)
[...]	Набір символів (наприклад, всі літери: [a-zA-Z])
[^...]	Символи, які відсутні в наборі
R|R	Альтернативна відповідність
RR	Конкатенаіція відповідностей
(R)	Відповідає будь-якому виразу *R* всередині () і створює групу (зберігає підрядок, що співпав)
(?:R)	Те ж, що і (R), але просто відділяє частину регулярного виразу і не створює зберігаючу групу
(?=R)	Випереджаюча перевірка: відповідає, якщо є співпадіння *R* з наступними символами рядка, але не поглинає їх (наприклад, X (?=Y) відповідає символу X тільки, якщо за ним слідкує символ Y)
(?!R)	Відповідає, якщо вираз *R* не відповідає наступним символам; перевірка зворотня до (?=R)
(?P=name)	Відповідає тексту, знайденому попередньою групою з ім’ям *name*
(?#...)	Коментар
(?letter)	Флаг режиму; *letter*: a, i, L, m, s, u, x
(?<=R)	Ретроспективна перевірка: відповідає, якщо поточній позиції в рядку передує співпадіння з виразом *R*, що закінчується в поточній позиції
\число	Відповідає вмісту групи з заданим номером (нумерація з 1)

Продовження таблиці 5.1

Оператор	**Опис**
(?(id/name /yespattern | nopattern)	Намагається знайти співпадіння з шаблоном yespattern, якщо існує група з номером *id* або іменем *name*; інакше використовується шаблон nopattern
\A	Тільки початок рядка
\b	Порожній рядок на межі слова
\B	Порожній рядок не на межі слова
\d	Будь-яка десяткова цифра ([0-9] для ASCII)
\D	Будь-який символ, що не є десятковою цифрою ([^0-9] для ASCII)
\s	Бідь-який пробільний символ ([\t\n\r\f\v] для ASCII)
\S	Бідь-який непробільний символ ([^ \t\n\r\f\v] для ASCII)
\w	Будь-який алфавітно-цифровий символ ([a-zA-Z0-9_] для ASCII)
\W	Бідь-який не алфавітно-цифровий символ ([^a-zA-Z0-9_] для ASCII)
\Z	Відповідає тільки кінцю рядка

5.2.2 Пакет для парсингу XML і HTML lxml

Пакет lxml – XML-інструментар, який є з'єднанням Python з C-бібліотеками libxml2 і libxslt.

Головним об'єктом-конейнером для ElementTree API, який міститься в модулі lxml.etree, є Element.

Для додавання елементів-нащадків до батьківського елемента використовуються методи append і SubElement. Елементи є переліками, що дозволяє виконувати над ними дії як зі стандартними переліками в Python. Елементи мають атрибути: tag (ім'я елемента), text (текст всередині елемента), tail (текст, що слідує за елементом), attrib (словник, який містить імена атрибутів елемента XML та

відповідні їм значення), base (базовий URI), prefix (префікс простора імен), sourceline (номер рядка даного елемента під час парсингу).

Метод clear дозволяє очистити вміст елемента. Метод insert призначений для додавання нового нащадка в задану першим аргументом позицію. Метод remove вилучає заданий у вигляді атрибута елемент, який є нащадком даного. Метод set призначений для того, щоб створити або змінити заданий атрибут (перший аргумент) на задане значення (другий аргумент).

Метод find(path[, namespaces=D]) дозволяє виконувати пошук серед елементів-нащадків. Метод findall(path[, namespaces=N]) дозволяє в свою чергу повернути весь перелік таких елементів. Метод findtext дозволяє виконати пошук текстового вмісту всередині даного елемента. Метод get призначений для роботи зі значеннями атрибутів елементів. Метод getchildren повертає перелік усіх нащадків даного елемента. Метод getiterator дозволяє ітерувати нащадків даного елемента. Метод getroottree повертає сутність ElementTree, яка містить даний елемент. Метод items повертає перелік двохелементних кортежів (name, value) для кожного XML-атрибута даного елемента. Метод iterancestors повертає батьківські елементи (а також їх батьківські елементи тощо) даного елемента. Метод iterchildren повертає ітератор, який ітерує всіх нащадків даного елемента. Метод iterdescendants повертає ітератор за всіми нащадками даного елемента (нащадками нащадків тощо). Метод itersiblings повертає ітератор, який відвідує всі елементи, які знаходяться на одному рівні ієрархії з даним. Метод keys повертає перелік імен XML-атрибутів даного елемента. Метод xpath(s[, namespaces=N][, var=value][, ...]) дозволений для обчислення XPath-виразу.

Функція parse дозволяє видобувати інформацію з XML-документів. Результат виконання даної функції є класом ElementTree, який представляє XML-документ у вигляді дерева.

Функція ElementTree призначена для створення нового дерева документа на основі кореневого елемента. Для запису створеного дерева в файл призначений метод write класу ElementTree. Метод getroot повертає кореневий елемент дерева.

Підпакет lxml.html призначений для парсингу web-сторінок. Модуль soupparser даного підпакета soupparser містить функцію parse(input), яка формує з HTML-документа дерево ElementTree, та

функцію fromstring(s), яка повертає дерево вузлів, що представляє рядок тегів *s*.

Модуль etbuilder може використовуватися для більш зручного створення XML-документів. E(tag, *p, **kw) – об'єкт-фабрика, який створює сутності et.Element. Перший аргумент *tag* – ім'я елемента у вигляді рядка. Може бути використана будь-яка кількість позиційних аргументів *p*, за якими слідує будь-яка кількість аргументів-ключових слів. Рядкові аргументи додаються до вмісту тегів. Наприклад:

Python

```
>>> colElt=E('col', valign='top', align='left')
>>> et.tostring(colElt)
'<col align="left" valign="top" />'
```

Функція subElement(parent, child) данного модуля дозволяє створити елемент та додати його в якості наступного нащадка деякого батьківського вузла. Для додавання текстового вмісту до елемента використовується функція addText(node, s).

Підпакет lxml.sax реалізує підтримку подій моделі SAX для об'єктів ElementTree або Element. Для підтримки побудови нового дерева за допомогою подій SAX використовується обробник вмісту даного підпакету ElementTreeContentHandler. Для генерації подій з готових об'єктів ElementTree або Element призначені методи startElementNS, endElementNS, characters.

Для застосування інтерфейсу стандартної моделі SAX для Python використовується пакет xml.sax.

5.3. Завдання на лабораторну роботу

5.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

5.3.2. Розробити програмне забезпечення, яке дозволяє створювати, парсити, змінювати XML-файли у відповідності з індивідуальним завданням з лабораторної роботи № 4, узгодивши його додатково з викладачем.

5.3.3. Виконати тестування розробленого програмного забезпечення.

5.3.4. Оформити звіт.

5.3.5. Відповісти на контрольні запитання.

5.4. Зміст звіту

5.4.1. Мета роботи.

5.4.2. Завдання до роботи.

5.4.3. Текст розробленого програмного забезпечення.

5.4.4. Перелік проблем, які були виявлені і розв'язані або не розв'язані під час роботи над проектом, з описом відповідної ситуації. Якщо проблему було розв'язано, то необхідно додатково описати прийняті рішення. Якщо проблему не було розв'язано, ситуацію має бути ідентифіковано максимально детально з поясненням причин.

5.4.5. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз та аналіз ефективності використаних оптимізаційних технік.

5.5. Контрольні запитання

5.5.1. Яку функціональність надає модуль re?

5.5.2. Що таке шаблони регулярних виразів та для чого вони використовуються?

5.5.3. Який синтаксис можна використовувати для побудови регулярних виразів?

5.5.4. Які засоби стандартної бібліотеки Python можуть використовуватися для парсингу?

5.5.5. Що таке XML?

5.5.6. Які існують моделі оброблення XML-документів?

5.5.7. Які засоби надає пакет lxml?

5.5.8. Яку модель роботи з XML реалізує пакет lxml?

5.5.9. Яким чином виконати парсинг HTML-документів та що є результатом даного процесу?

5.5.10. Яким чином внести зміни в існуючий XML-файл?

ЛАБОРАТОРНА РОБОТА № 6
ОБРОБЛЕННЯ ПРИРОДНЬОЇ МОВИ

6.1. Мета роботи

6.1.1 Ознайомитись з основними інструментами оброблення природньої мови, які входять у склад бібліотеки NLTK мови програмування Python

6.1.2 Навчитися розв'язувати актуальні практичні завдання у галузі оброблення природної мови за допомогою бібліотеки NLTK.

6.2. Короткі теоретичні відомості

Natural Language Toolkit (NLTK) – пакет бібліотек та програм для обробки природньої мови.

Для початку роботи з бібліотекою NLTK необхідно встановити даний пакет з веб-сайту nltk.org. Окрім програм дана бібліотека включає великий обсяг даних, які можуть використовуватись під час навчання моделей та характеризуються широкою розмаїтістю. Дані для пакету NLTK включають тексти, граматики, навчені моделі тощо. Для того щоб завантажити дані на комп'ютер, необхідно після встановлення пакету скористатися командами:

Python

```
>>> import nltk
>>> nltk.download()
```

Після виконання останньої команди на екрані з'явиться вікно NLTK Downloader (рис. 6.1), з якого можна вибрати дані, які необхідно завантажити.

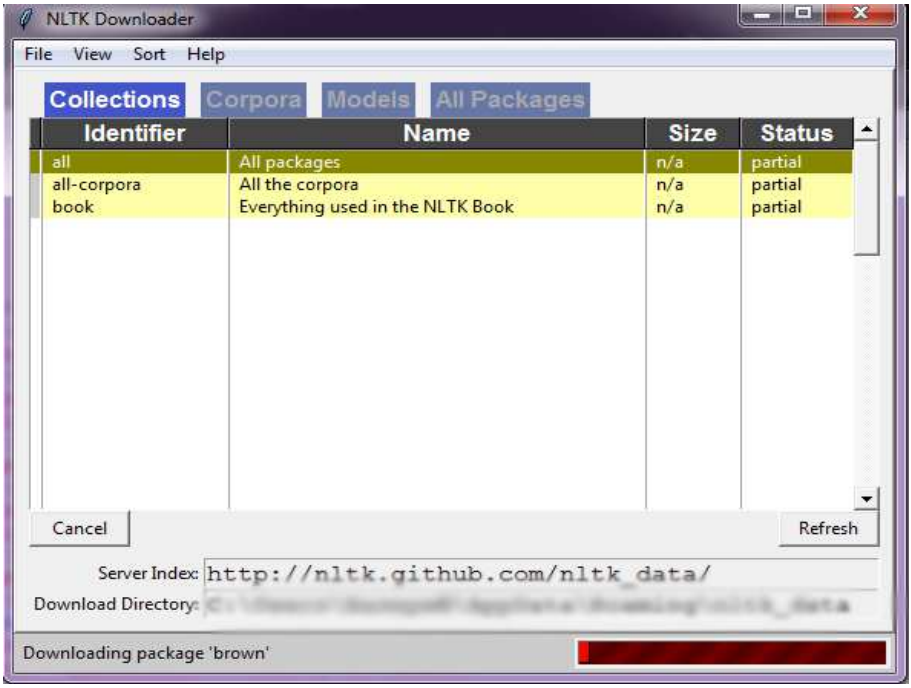

Рисунок 6.1 – Вікно завантаження даних пакету NLTK

У таблиці 6.1 представлені модулі пакету NLTK та завдання процесу оброблення мови та функціональність, що їм відповідає.

Таблиця 6.1 – Модулі NLTK

Завдання оброблення мови	**Модуль NLTK**	**Функціональність**
Доступ до бібліотеки документів	corpus	Стандартизовані інтерфейси до документів та словників
Обробка рядків	tokenize, stem	Розбиття на лексеми
Визначення словосполучень	collocations	Визначення n-грам та відповідної статистики
Розмітка за частинами мови	tag	Створення правил та розподіл слів у реченні за частинами мови

Продовження таблиці 6.1

Завдання оброблення мови	**Модуль NLTK**	**Функціональність**
Машинне навчання	classify, cluster, tbl	Дерева рішень, максимальна ентропія, наївні байєсівські методи, k-середніх
Чанкінг	chunk	Формування деревоподібної структури на основі відповідності регулярним виразам над тегами
Контекстно-вільні граматики	grammar	Визначення продукцій
Парсинг	parse, ccg	Формування деревоподібної структури, що представляє внутрішню структуру тексту
Семантична інтерпретація	sem, inference	lambda-обчислення, логіка першого порядку, перевірка моделі
Оціночні метрики	metrics	Точність, коефіцієнти узгодженості
Імовірність та оцінка	probability	Розподіл частот, згладжений розподіл імовірностей
Лінгвістичні дослідження	toolbox	Читання, виведення та маніпулювання базами даних Toolbox

Модуль пакету NLTK corpus містить тексти книг електронного архіву Project Gutenberg (gutenberg), web-тексти (webtext), тексти з 500 джерел, розподілені за окремими жанрами (brown), більше 10 000 новин, розподілених за 90 темами, при цьому вони можуть належати одночасно декільком темам (reuters), також є тексти, розподілені в часі, словники (перелік слів, транскрипції для вимовляння, порівняння слів у різних мовах), семантично-орієнтований словник англійської мови (wordnet), що містить визначення, синоніми, ієрархічні відносини між поняттями та більш специфічними значеннями.

Основна функціональність corpus забезпечується за допомогою наступних методів:

– fileids() – файли корпусу текстів;

– fileids([categories]) – файли корпусу, що відповідають заданим категоріям;

– categories() – категорії корпусу;

– categories([fileids]) – категорії корпусу, що відповідають заданим файлам;

– raw() – необроблений вміст корпусу;

– raw(fileids=[f1,f2,f3]) – необроблений вміст заданих файлів;

– words() – слова зі всього корпусу;

– words(categories=[c1,c2]) – слова за заданими категоріями;

– sents() – речення зі всього корпусу;

– encoding(fileid) – кодування файлу.

Приклад визначення кількості входжень заданого слова в деякий текст:

Python

```
>>> text3.count("smote")
5
```

Можна також завантажити власну колекцію текстів або отримати доступ до текстів книг, HTML-документів, RSS-стрічок через інтернет, локальних файлів, документів форматів PDF (pypdf), MS Word (pywin32) тощо. Наприклад, доступ до HTML-сторінки можна отримати наступним способом:

Python

```
>>> from urllib import request
>>> url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
>>> html = request.urlopen(url).read().decode('utf8')
```

Для оброблення даних RSS-стрічки необхідно скористатися пакетом feedparser та наступними командами:

Python

```
>>> import feedparser
>>> llog =
feedparser.parse("http://languagelog.ldc.upenn.edu/nll/?feed=
atom")
```

Розглянемо приклад визначення частин мови:

Python

```
>>> text = word_tokenize("And now for something
completely different")
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something',
'NN'), ('completely', 'RB'), ('different', 'JJ')]
```

Таким чином, and – єднальний сполучник, now, completely – прислівники, for – прийменник, different – прикметник, something – іменник.

Приклад створення власних регулярних виразів для визначення частин мови, до яких належать слова в тексті:

Python

```
>>> patterns = [
...     (r'.*ing$', 'VBG'),
...     (r'.*ed$', 'VBD'),
...     (r'.*es$', 'VBZ'),
...     (r'.*ould$', 'MD'),
...     (r'.*\'s$', 'NN$'),
...     (r'.*s$', 'NNS'),
...     (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),
...     (r'.*', 'NN')
... ]
>>> regexp_tagger = nltk.RegexpTagger(patterns)
>>> regexp_tagger.tag(brown_sents[3])
```

```
[('``', 'NN'), ('Only', 'NN'), ('a', 'NN'), ('relative', 'NN'),
('handful', 'NN'), ('of', 'NN'), ('such', 'NN'), ('reports',
'NNS'), ('was', 'NNS'), ('received', 'VBD'), ("''", 'NN'), (',',
'NN'), ('the', 'NN'), ('jury', 'NN'), ('said', 'NN'), (',', 'NN'),
('``', 'NN'), ('considering', 'VBG'), ('the', 'NN'),
('widespread', 'NN'), ...]
```

Приклад виконання чанкінга для речення «The little yellow dog barked at the cat.», який дозволяє побудувати в результаті деревоподібну структуру:

Python

```
>>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow",
"JJ"),
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),
("the", "DT"), ("cat", "NN")]

>>> grammar = "NP: {<DT>?<JJ>*<NN>}"

>>> cp = nltk.RegexpParser(grammar)
>>> result = cp.parse(sentence)
>>> print(result)
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

У результаті на екран можна вивести структуру, зображену на рис. 6.2.

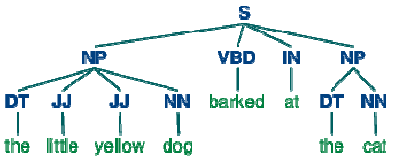

Рисунок 6.2 – Результат виконання чанкінгу

Розглянемо приклад виконання чанкінгу на основі власного регулярного виразу:

Python

```
grammar = r"""
  NP: {<DT|PP\$>?<JJ>*<NN>}
     {<NNP>+}
"""
cp = nltk.RegexpParser(grammar)
sentence = [("Rapunzel", "NNP"), ("let", "VBD"),
("down", "RP"), ("her", "PP$"), ("long", "JJ"),
("golden", "JJ"), ("hair", "NN")]

>>> print(cp.parse(sentence))
(S
  (NP Rapunzel/NNP)
  let/VBD
  down/RP
  (NP her/PP$ long/JJ golden/JJ hair/NN))
```

6.3. Завдання на лабораторну роботу

6.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

6.3.2. Розробити програмне забезпечення у відповідності з індивідуальним завданням, узгодженим з викладачем.

6.3.3. Виконати тестування розробленого програмного забезпечення.

6.3.4. Оформити звіт.
6.3.5. Відповісти на контрольні запитання.

6.4. Зміст звіту

6.4.1. Мета роботи.
6.4.2. Завдання до роботи.
6.4.3. Текст програми з основними коментарями.
6.4.4. Інтерфейс роботи з програмою в декількох режимах.
6.4.5. Результати тестування: вхідні дані та результати роботи програми.
6.4.6. Перелік проблем, які були виявлені і розв'язані або не розв'язані під час роботи над проектом, з описом відповідної ситуації. Якщо проблему було розв'язано, то необхідно додатково описати прийняті рішення. Якщо проблему не було розв'язано, ситуацію має бути ідентифіковано максимально детально з поясненням причин.
6.4.7. Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

6.5. Контрольні запитання

6.5.1. Яким чином виконується оброблення природної мови в бібліотеці NLTK?
6.5.2. Які основні модулі входять до складу бібліотеки NLTK?
6.5.3. Яким чином можна використати готові тексти для роботи програми на Python?
6.5.4. Яким чином розбити текст на лексеми?
6.5.5. Яким чином побудувати граматику в NLTK?
6.5.6. Що таке біграми та яким чином їх визначити у тексті?
6.5.7. Що таке чанкінг та яким чином він виконується?
6.5.8. Яким чином та за допомогою яких засобів виконується семантична інтерпретація?
6.5.9. Що таке контекстно-вільна граматика? Наведіть приклади.
6.5.10. Яким чином визначити частини мови слів тексту?

ЛАБОРАТОРНА РОБОТА № 7
МАШИННЕ НАВЧАННЯ

7.1. Мета роботи

7.1.1 Ознайомитись з основними пакетами, які використовуються для машинного навчання в програмах, написаних мовою Python.

7.1.2. Навчитися розробляти сучасні інтелектуальні системи з використанням методів машинного навчання.

7.2. Короткі теоретичні відомості

7.2.1 Бібліотека Pandas

Пакет pandas забезпечує аналіз даних з реального світу для програм, написаних мовою Python.

Структурами даних, які використовуються в даному пакеті, є:

– Series ([data, index, dtype, name, copy, ...]) – одновимірний проіндексований масив, який може містити дані будь-яких типів (s = Series(data, index=index), де *data* може бути даними типу dict (Series працює подібно словникам), ndarray (Series працює подібно масивам NumPy) або скалярного типу, а index – перелік міток для даних);

– DataFrame ([data, index, columns, dtype, copy]) – двовимірний проіндексований масив з колонками потенційно будь-якого типу. DataFrame в якості даних може використовувати dict з одномірних ndarray, двомірні ndarray, структуровані ndarray або типу запис, Series, DataFrame;

– Panel ([data, items, major_axis, minor_axis, ...]) – контейнер для тривимірних даних;

– Panel4D ([data, labels, items, major_axis, ...]) – контейнер для чотирьохвимірних даних;

– PanelND – модуль з множиною фабричних функцій, який дозволяє користувачу створювати *N*-вимірні контейнери даних.

Для того щоб визначити категоріальні значення, необхідно використовувати під час створення об’єкту Series атрибут dtype="category".

Приклад створення структури даних типу DataFrame:

Python

```
d = {'one' : Series([1., 2., 3.], index=['a', 'b', 'c']),
'two' : Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
df = DataFrame(d)
```

У DataFrame стовпці можна вибирати, додавати, вилучати аналогічно тому, як це виконується зі словниками. Індексація описана в таблиці 7.1. Окрім того можна отримувати доступ до значень елементів структур даних типу Series, вказуючи індекс в якості атрибута (sa.c). Більш складні умови вибору можна реалізовувати за допомогою методу map. Для визначення умов можуть також використовуватися метод where та окремо для DataFrame – query (умови визначаються з використанням синтаксису Python). Для виконання зрізів використовується метод select.

Таблиця 7.1 – Операції індексації у DataFrame

Операція	**Синтаксис**	**Результат**
Вибрати стовпчик	df[col]	Series
Вибрати рядок за міткою	df.loc[label]	Series
Вибрати рядок за цілочисельною позицією	df.iloc[loc]	Series
Виконати зріз рядків	df[5:10]	DataFrame
Вибрати рядки за вектором логічних значень	df[bool_vec]	DataFrame

Поелементні функції NumPy та інші функції (count, sum, mean, mad, median, min, max, mode, abs, prod, std, var, sem, skew, kurt, quantile, cumsum, cumprod, cummax, cummin) можуть застосовуватися до DataFrame, якщо дані числові.

Для роботи з даними структурами можуть використовуватися методи head ([n]) і tail ([n]), index (рядки), columns (для DataFrame), values для доступу до даних. DataFrame має методи add, sub, mul, div, пов’язані функції radd, rsub для виконання бінарних операцій і методи бінарного порівняння eq, ne, lt, gt, le, ge. Для того щоб застосувати

визначену користувачем функцію до двох структур даних типу DataFrame, необхідно використати метод combine, для виконання дій над однією структурою призначений метод apply (func[, axis]), у випадку необхідності роботи з невекторизованими даними використовуються методи applymap (func) і map (arg[, na_action]). Для підбиття статистичних підсумків над даними може використовуватися метод describe ([percentile_width, ...]).

Методи idxmin ([axis, out, skipna]) і idxmax ([axis, out, skipna]) повертають індекс з мінімальним/максимальним значенням, а функції cut (x, bins[, right, labels, retbins, ...]) і qcut (x, q[, labels, retbins, precision]) розділяють значення на задану кількість інтервалів.

Для зміни структур даних використовується метод reindex ([index]), а для зміни індексів – метод rename ([index]). Для ітерування призначені атрибути iteritems, iterrows, itertuples. Сортування виконується за допомогою методів sort_index ([ascending]) за індексами та order ([return_indexer, ascending]) за значеннями. Для сортування за рівнями використовується метод sortlevel ([level, ascending, ...]).

Об’єкт Series надає методи nsmallest ([n, take_last]) і nlargest ([n, take_last]), що повертають *n* найменших і відповідно найбільших значень.

Структури типу Series мають набір методів роботи з рядками, зокрема атрибут str, метод split ([pat, n, return_type]), який розділяє рядки, формуючи списки, методи replace ([to_replace, value, inplace, ...]) і findall (pat[, flags]), які працюють з регулярними виразами, cat (виконує конкатенацію рядків), len, lower, upper тощо.

Для того щоб ідентифікувати та вилучити дубльовані рядки в DataFrame призначені методи duplicated ([take_last]) та drop_duplicates ([take_last, inplace]).

Для того щоб змінити DataFrame, додаючи новий індекс, можна використати метод set_index (keys[, drop, append, ...]), а метод reset_index ([level, drop, name, inplace]) – для перетворення індексів у стовпці DataFrame.

Набір статистичних функцій включає pct_change, що обчислює зміну у відсотках протягом заданого періоду, cov – коваріацію, pearson, kendall, spearman – коефіцієнти кореляції.

Для роботи з пропущеними значеннями в даних, які можуть

бути представлені як NaN, None, inf, -inf, призначені функції: isnull(), notnull(), для заповнення пропущених значень – fillna ([value, method, axis, ...]), для вилучення індексів з пропущеними значеннями – dropna ([axis, inplace]), для інтерполяції пропущених значень – interpolate ([method, axis, limit, ...]).

Функція concat(objs[, axis, join, join_axes, ...]) призначена для конкатенації об'єктів вздовж однієї з осей, де *objs* – перелік або словник об'єктів Series, DataFrame, Panel.

Для об'єднання двох DataFrame об'єктів призначена функція merge(left, right[, how, on, left_on, ...]), яка за суттю близька до аналогічної функції в SQL. Якщо ключі співпадають, то результат обчислюється у вигляді декартового добутку.

Для об'єднання двох об'єктів DataFrames з різними індексами використовується метод join (other[, on, how, lsuffix, ...]).

7.2.2 Бібліотека scikit-learn

Бібліотека scikit-learn – бібліотека машинного навчання, яка підтримує алгоритми класифікації, регресії, кластеризації.

Алгоритми машинного навчання реалізовані за допомогою класів та функцій. Класи імплементують метод fit для навчання (наприклад, кластерів на навчальній вибірці) та методи get_params() і set_params() для визначення і встановлення параметрів оцінювача, а функції, наприклад, отримуючи навчальну вибірку, повертають масив цілочисельних міток, що відповідають різним кластерам. Набір методів кожного класу залежить від алгоритму машинного навчання, який він реалізує. Наприклад, для прогнозування може використовуватись метод predict(X), для оцінювання деяких параметрів – метод score(), а метод transform() – для перетворення набору даних (наприклад, під час вибору властивостей).

Базовими класами функцій оцінювання є базовий клас BaseEstimator, клас для всіх класифікаторів ClassifierMixin, клас для всіх оцінювачів кластерів ClusterMixin, клас для всіх функцій оцінювання регресійних моделей RegressorMixin, клас для всіх перетворювачів TransformerMixin. Метод score(X, y[, sample_weight]) класів ClassifierMixin та RegressorMixin повертає середню похибку на даній тестовій вибірці. Метод fit_predict(X[, y]) класу ClusterMixin

виконує кластеризацію на наборі даних *X* та повертає мітки класів.

Модуль sklearn.preprocessing включає методи масштабування, центрування, нормалізації та перетворення у двійкову форму, зокрема:

– Binarizer([threshold, copy]) – перетворює у двійкову форму за порогом;

– Normalizer([norm, copy]) – нормалізує вібірки індивідуально до одиничної норми.

Модуль sklearn.feature_extraction призначений для видобування властивостей з необроблених даних і на даний момент включає методи для видобування властивостей з тексту та зображень.

Функція DictVectorizer([dtype, ...]) перетворює перелік відображень властивість-значення у вектори, а функція FeatureHasher([...]) імплементує гешування властивостей .

Підмодуль sklearn.feature_extraction.image містить засоби, які видобувають властивості з зображень.

Підмодуль sklearn.feature_extraction.text містить засоби для побудови векторів властивостей з текстових документів: CountVectorizer([...]), який перетворює колекцію текстових документів у матрицю підрахунку лексем.

Модуль sklearn.feature_selection імплементує алгоритми вибору властивостей:

– GenericUnivariateSelect([...]) – одномірний селектор зі стратегією, що конфігурується;

– SelectKBest([score_func, k]) – вибір властивостей відповідно до *k* найвищих оцінок.

Модуль sklearn.lda імплементує лінійний дискримінантний аналіз за допомогою функції lda.LDA([n_components, priors]).

Модуль sklearn.neighbors імплементує алгоритм k-найближчих сусідів за допомогою наступних класів: NearestNeighbors([n_neighbors, ...]), навчання без учителя для імплементації пошуку сусідів, KNeighborsClassifier([...]), класифікатор, що імлементує голосування k-найближчих сусідів.

Модуль sklearn.naive_bayes імплементує алгоритм наївної байєсівської класифікації, включаючи наступні класи: GaussianNB, для поліноміальних моделей MultinomialNB([alpha, ...]), для багатомірних моделей Бернуллі BernoulliNB([alpha, binarize, ...]).

Модуль sklearn.multiclass імплементує наступні алгоритми

багатокласової класифікації за допомогою відповідних класів та функцій підбору стратегії і прогнозування:

– OneVsRestClassifier(estimator[, ...]), fit_ovr(estimator, X, y[, n_jobs]), predict_ovr(estimators, ...) – багатокласова стратегія «один-проти-інших»;

– OneVsOneClassifier(estimator[, ...]), fit_ovo(estimator, X, y[, n_jobs]), predict_ovo(estimators, classes, X) – багатокласова стратегія «один-проти-одного».

Модуль sklearn.cluster дозволяє виконувати кластеризацію непроіндексованих даних. Функції модуля:

– estimate_bandwidth(X[, quantile, ...]) – оцінює ширину полоси для використання з алгоритмом здвигу середнього;

– k_means(X, n_clusters[, init, ...]) – алгоритм кластеризації K-середніх.

Модуль sklearn.tree включає моделі для класифікації (клас DecisionTreeClassifier та його альтернатива ExtraTreeClassifier([criterion, ...])) та регресії (клас DecisionTreeRegressor та його альтернатива ExtraTreeRegressor([criterion, ...])) на основі дерев рішень та функцію export_graphviz(decision_tree[, ...]) для експорту дерев рішень у формат DOT.

Модуль sklearn.metrics включає функції оцінки, метрики продуктивності, попарні метрики (metrics.pairwise) та обчислення відстані.

Модуль sklearn.svm включає алгоритми методу опорних векторів, зокрема оцінювачі: C-SVM класифікація SVC([C, kernel, degree, gamma, coef0, ...]), лінійна SVM класифікація LinearSVC([penalty, loss, dual, tol, C, ...]).

Модуль sklearn.datasets включає засоби для завантаження даних та генерації вибірок. Функції завантаження даних дозволяють завантажувати імена файлів і дані з 20 новинних груп (fetch_20newsgroups([data_home, ...])), цифрові множини даних для класифікації (load_digits([n_class])) тощо. Функції генерації вибірок дозволяють згенерувати випадкову проблему класифікації на *n* класів (make_classification([n_samples, ...])), дані для бінарної класифікації (make_hastie_10_2([n_samples, ...])) тощо.

Розглянемо приклад практичного застосування алгоритмів

машинного навчання, представлених у бібліотеці scikit-learn, для розпізнавання зображень цифр, написаних вручну, розміром 8x8:

Python

```
        import matplotlib.pyplot as plt
        from sklearn import datasets, svm, metrics
        digits = datasets.load_digits()
        images_and_labels = list(zip(digits.images, digits.target))
        for index, (image, label) in
enumerate(images_and_labels[:8]):
                plt.subplot(2, 8, index + 1)
                plt.axis('off')
                plt.imshow(image, cmap=plt.cm.gray_r,
        interpolation='nearest')
                plt.title('Train: %i' % label)

        n_samples = len(digits.images)
        data = digits.images.reshape((n_samples, -1))
        classifier = svm.SVC(gamma=0.001)
        classifier.fit(data[:n_samples / 2], digits.target[:n_samples /
2])
        expected = digits.target[n_samples / 2:]
        predicted = classifier.predict(data[n_samples / 2:])
        print("Classification report for classifier %s:\n%s\n"
                % (classifier,
        metrics.classification_report(expected, predicted)))
        print("Confusion matrix:\n%s" %
metrics.confusion_matrix(expected, predicted))

        images_and_predictions = list(zip(digits.images[n_samples /
2:], predicted))
        for index, (image, prediction) in
enumerate(images_and_predictions[:8]):
                plt.subplot(2, 8, index + 9)
                plt.axis('off')
                plt.imshow(image, cmap=plt.cm.gray_r,
```

```
        interpolation='nearest')
                plt.title('Predict:%i' % prediction)
                plt.show()
```

На рис. 7.1 представлено результат виконання даної програми. Матриця неточностей за результатами машинного навчання:

```
[[87  0  0  0  1  0  0  0  0  0]
[  0 88  1  0  0  0  0  0  1  1]
[  0  0 85  1  0  0  0  0  0  0]
[ 0  0  0 79  0  3  0  4  5  0]
[ 0  0  0  0 88  0  0  0  0  4]
[ 0  0  0  0  0 88  1  0  0  2]
[ 0  1  0  0  0  0 90  0  0  0]
[ 0  0  0  0  0  1  0 88  0  0]
[ 0  0  0  0  0  0  0  0 88  0]
[ 0  0  0  1  0  1  0  0  0 90]]
```

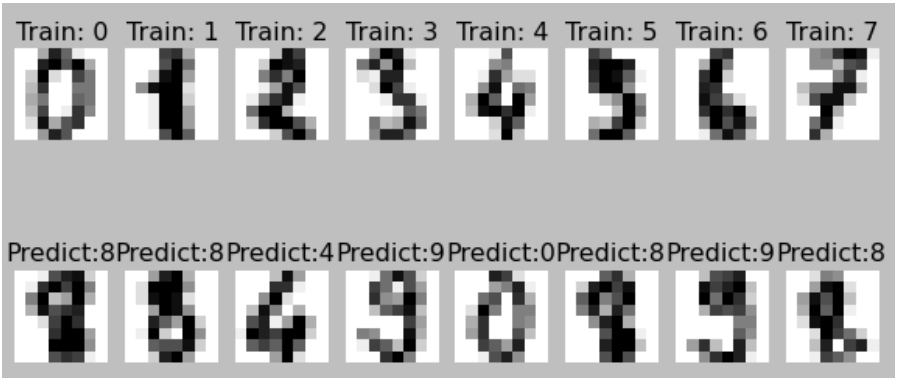

Рисунок 7.1 – Розпізнавання цифр, написаних вручну

7.3. Завдання на лабораторну роботу

7.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

7.3.2. Розробити програмне забезпечення у відповідності з індивідуальним завданням, узгодженим з викладачем.

7.3.3. Виконати тестування розробленого програмного забезпечення.

7.3.4. Оформити звіт.

7.3.5. Відповісти на контрольні запитання.

7.4. Зміст звіту

7.4.1. Мета роботи.

7.4.2. Завдання до роботи.

7.4.3. Опис алгоритма розв'язання задачі.

7.4.4. Текст програми з основними коментарями.

7.4.5. Інтерфейс роботи з програмою в декількох режимах.

7.4.6. Результати тестування: вхідні дані та результати роботи програми.

7.4.7. Перелік проблем, які були виявлені і розв'язані або не розв'язані під час роботи над проектом, з описом відповідної ситуації. Якщо проблему було розв'язано, то необхідно додатково описати прийняті рішення. Якщо проблему не було розв'язано, ситуацію має бути ідентифіковано максимально детально з поясненням причин.

7.4.8. Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

7.5. Контрольні запитання

7.5.1. Для чого призначена бібліотека pandas?

7.5.2. Які структури даних та для чого використовуються в пакеті pandas?

7.5.3. Які функції бібліотеки pandas можуть бути використані для введення/виведення даних?

7.5.4. Яким чином можна модифікувати дані, що зберігаються в структурах даних пакету pandas?

7.5.5. Яким чином виконується індексація даних у пакеті pandas?

7.5.6. Які функції для роботи з декількома структурами даних підтримуються бібліотекою pandas?

7.5.7. Яким чином виконується оброблення часових рядів у бібліотеці pandas?

7.5.8. Яку галузь знань охоплює машинне навчання та які завдання воно дозволяє розв’язати?

7.5.9. Для виконання яких завдань призначена бібліотека scikit-learn?

7.5.10. Які засоби вибору властивостей надає бібліотека scikit-learn?

7.5.11. Які засоби попереднього оброблення даних надає бібліотека scikit-learn?

7.5.12. Які засоби кластеризації надає бібліотека scikit-learn?

7.5.13. Які засоби класифікації надає бібліотека scikit-learn?

7.5.14. Які методи навчання без учителя представлені в бібліотеці scikit-learn?

7.5.15. Яким чином інтегруються засоби бібліотек pandas та scikit-learn?

7.5.16. Які засоби перетворення даних використовуються в бібліотеці scikit-learn?

7.5.17. Які засоби пониження розмірності представлені в бібліотеці scikit-learn?

7.5.18. Які регресійні моделі можуть бути застосовані за допомогою засобів scikit-learn?

7.5.19. Які метрики надаються бібліотекою scikit-learn?

7.5.20. За допомогою яких засобів бібліотеки scikit-learn можна оцінити коваріацію?

7.5.21. Яким чином можна візуалізувати результати машинного навчання?

7.5.22. Для чого необхідні методи крос-валідації та які інструменти для цього використовуються в бібліотеці scikit-learn?

7.5.23. Яким чином можна згенерувати дані для машинного навчання?

7.5.24. Дані яких типів та з яких джерел можна обробляти за допомогою засобів бібліотеки scikit-learn?

ЛІТЕРАТУРА

1. Лутц, М. Изучаем Python [Текст] / М. Лутц ; Пер. с англ. – 4-е издание. – СПб. : Символ-Плюс, 2011. – 1280 с., ил.

2. Прохоренок, Н. А. Python 3 и PyQt. Разработка приложений [Текст] / Н.А. Прохоренок. – СПб. : БХВ-Петербург, 2012. – 704 с. : ил.

3. Бизли, Д. Python. Подробный справочник [Текст] / Д. Бизли ; пер. с англ. – СПб. : Символ-Плюс, 2010. – 864 с., ил.

4. Downey, A.B. Think Python [Текст] / Allen B. Downey. – O'Reilly, 2012. – 300 p.

5. Gaddis, T. Starting Out with Python [Текст] / Tony Gaddis. – 2nd Edition. – Addison-Wesley, 2011. – 648 p.

6. Chun, W.J. Core Python Applications Programming [Текст] / Wesley J. Chun. – Third Edition. – Prentice Hall, 2012. – 888 p.

7. Forcier, J. Python Web Development with Django [Текст] / Jeff Forcier, Paul Bissex, Wesley Chun. – Addison-Wesley Professional, 2008. – 408 p.

8. Beazley, D. Python Cookbook [Текст] / David Beazley, Brian K. Jones ; Third Edition. – Sebastopol : O'Reilly Media, Inc., 2013. – 706 p.

9. Лутц, М. Программирование на Python [Текст] / М. Лутц ; Пер. с англ. – 4-е издание. – том I. – СПб. : Символ-Плюс, 2011. – 992 с., ил.

10. Лутц, М. Программирование на Python [Текст] / М. Лутц ; Пер. с англ. – 4-е издание. – том II. – СПб. : Символ-Плюс, 2011. – 992 с., ил.

11. Lukaszewski, A. MySQL for Python [Текст] / A. Lukaszewski. – Birmingham : Packt Publishing, 2010. – 440 p.

12. Hetland, M.L. Pythom From Novice to Professional [Текст] / Magnus Lie Hetland. – Second Edition. – Apress, 2008. – 688 p.

13. Payne, J. Beginning Python [Текст] : Using Python 2.6 and Python 3.1 / James Payne. – Wiley Publishing Inc., 2010. – 624 p.

14. Phillips, D. Python 3 Object Oriented Programming [Текст] / Dusty Phillips. – Birmingham : Packt Publishing, 2010. – 404 p.

15. Harwani, B.M. Introduction to Python Programming and Developing GUI Applications with PyQT [Текст] / B.M. Harwani. – Cengage Learning PTR, 2011. – 416 p.

16. Hellmann, D. The Python Standard Library by Example [Текст] / Doug Hellmann. – Addison-Wesley, 2011. – 1344 p.

17. Kiusalaas, J. Numerical Methods in Engineering with Python [Текст] / Jaan Kiusalaas. – New York : Cambridge University Press, 2010. – 432 p.

18. Younker, J. Foundations of Agile Python Development [Текст] / Jeff Younker. – Apress, 2008. – 416 p.

19. Hughes, J.M. Real World Instrumentation with Python [Текст] / J.M. Hughes. – O'Reilly Media, 2010. – 624 p.

20. Guttag, J.V. Introduction to Computation and Programming Using Python [Текст] / John V. Guttag. – The MIT Press, 2013. – 280 p.

21. Lambert, K.A. Fundamentals of Python : From First Programs through Data Structures [Текст] / Kenneth A. Lambert. – Cengage Learning, 2009. – 944 p.

22. Langtangen, H.P. A Primer on Scientific Programming with Python [Текст] / Hans Petter Langtangen ; 3rd ed. – Springer, 2012. – 798 p.

23. Liang, Y.D. Introduction to Programming Using Python [Текст] / Y. Daniel Liang. – Prentice Hall, 2012. – 576 p.

24. McKinney, W. Python for Data Analysis [Текст] / Wes McKinney. – Sebastopol : O'Reilly Media, Inc., 2013. – 470 p.

25. Bird, S. Natural Language Processing with Python [Текст] / Steven Bird, Ewan Klein, Edward Loper. – Sebastopol : O'Reilly Media, Inc., 2009. – 504 p.

26. Richert, W. Building Machine Learning Systems with Python [Текст] / Willi Richert, Luis Pedro Coelho. – Birmingham : Packt Publishing Ltd., 2013. – 290 p.

27. Gries, P. Practical Programming : An Introduction to Computer Science Using Python 3 [Текст] / Paul Gries, Jennifer Campbell, Jason Montojo ; Second Edition edition. – Pragmatic Bookshelf, 2013. – 350 p.

28. Anders, M. Python 3 Web Development Beginner's Guide [Текст] / Michel Anders. – Birmingham : Packt Publishing, 2011. – 336 p.

29. Perkins, J. Python Text Processing with NLTK 2.0 Cookbook [Текст] / Jacob Perkins. – Birmingham : Packt Publishing, 2010. – 272 p.

30. Overview – Python 3.4.3rc1 documentation [Електронний ресурс] / Режим доступу : https://docs.python.org/3/

31. NumPy Reference [Електронний ресурс] / Режим доступу : http://docs.scipy.org/doc/numpy-dev/reference/

32. SciPy Tutorial [Електронний ресурс] / Режим доступу : http://docs.scipy.org/doc/scipy/reference/tutorial/index.html

33. Matplotlib Release 1.4.2 [Електронний ресурс] / John Hunter, Darren Dale, Eric Firing, Michael Droettboom. – Режим доступу : http://matplotlib.org/Matplotlib.pdf

34. Graphical User Interfaces with Tk [Електронний ресурс] / Режим доступу : https://docs.python.org/2/library/tk.html

35. Tkinter 8.5 reference : a GUI for Python [Електронний ресурс] / Режим доступу : http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

36. Documentation of scikit-learn 0.15 [Електронний ресурс] / Режим доступу : http://scikit-learn.org/stable/documentation.html

37. Pandas : powerful Python data analysis toolkit [Електронний ресурс] / Режим доступу : http://pandas.pydata.org/pandas-docs/stable/

38. MySQL 5.6 Reference Manual. Including MySQL Cluster NDB 7.3-7.4 Reference Guide [Електронний ресурс] / Режим доступу : http://downloads.mysql.com/docs/refman-5.6-en.a4.pdf

39. Effective Django – Effective Django [Електронний ресурс] / Режим доступу : http://effectivedjango.com/

40. Django documentation [Електронний ресурс] / Режим доступу : https://docs.djangoproject.com/en/1.7/

41. lxml [Електронний ресурс] / Режим доступу : http://lxml.de/3.4/lxmldoc-3.4.2.pdf