МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

Н.Ю. Лазоренко, Б.І. Денисюк, Д.О. Кінь

ПРИКЛАДНЕ ПРОГРАМУВАННЯ В ГІС

Рекомендовано вченою радою Київського національного університету будівництва і архітектури як навчальний посібник для студентів галузі знань 19 «Архітектура і будівництво» спеціальності 193 «Геодезія та землеустрій» освітньо-кваліфікаційного рівня «бакалавр»

Київ 2023

УДК 528.001+681.518
Л17

Рецензенти: *К.О. Метешкін,* д-р техн. наук, професор, Харківський національний університет міського господарства ім. О. М. Бекетова;
О.Є. Куліковська, д-р техн. наук, професор, Криворізький національний університет;
А.О. Анненков, д-р техн. наук, професор, Київський національний університет будівництва і архітектури

Відповідальний за випуск Ю.О. Карпінський, д-р техн. наук, професор

Затверджено на засіданні вченої ради Київського національного університету будівництва і архітектури, протокол № 9 від 26 червня 2023 року.

Лазоренко Н. Ю.
Л17 Прикладне програмування в ГІС: навчальний посібник / Н.Ю. Лазоренко, Б.І. Денисюк, Д.О. Кінь. – Київ: КНУБА, 2023. – 218 с.

ISBN 978-966-627-253-2

Містить теоретичні відомості та загальні положення про базовий синтаксис мови програмування Python, підходи та принципи програмування для забезпечення автоматизації процесів геопросторового аналізу і моделювання об’єктів і явищ. Розглянуто застосування мови програмування Python бібліотеки ArcPy у ArcGIS та бібліотеки PyQGIS у QGIS для вирішення прикладних задач. Наведено приклади розв’язання прикладних задач у ГІС.

Призначено для студентів і аспірантів, які навчаються за галузі знань 19 «Архітектура і будівництво» спеціальності 193 «Геодезія та землеустрій» освітньо-кваліфікаційного рівня «бакалавр».

УДК 528.001+681.518

ISBN 978-966-627-253-2

ЗМІСТ

ВСТУП

Навчальний посібник підготовлено відповідно до розділів програми з навчальної дисципліни «Прикладне програмування в ГІС» освітньо-професійної програми КНУБА з підготовки здобувачів вищої освіти на першому (бакалаврському) рівні за спеціальністю 193 «Геодезія та землеустрій» спеціалізації «Геоінформаційні системи і технології» (далі – ОПП).

Основна мета цього навчального посібника полягає у формуванні компетентностей та знань, достатніх для вирішення різноманітних прикладних завдань за допомогою мови програмування Python в геоінформаційних системах, зокрема для автоматизації процесів геопросторового аналізу і моделювання об’єктів і явищ.

Навчальний посібник забезпечує формування компетентностей, які сприяють: поглибленню розуміння принципів географічного та геоінформаційного підходів до вивчення об’єктів і явищ реального світу, зважаючи на цілісність геосистеми та взаємодію об’єктів у просторі та часі; поглибленню знань і розуміння архітектури сучасних інструментальних геоінформаційних систем і системотехнічного осмислення прикладних задач та розроблення технологічних схем їх розв’язання за допомогою геоінформаційних систем; засвоєнню сучасних теоретичних, методичних і алгоритмічних основ розроблення програмного забезпечення та використання мови програмування Python з метою розширення прикладних функцій інструментальних ГІС і автоматизації процесів геопросторового аналізу і моделювання для розв’язання прикладних завдань у різних сфер економіки країни.

У цьому навчальному посібнику загально розглянуто синтаксис мови програмування Python, а саме: правила, функції, алгоритми з використанням конструкцій циклів та умов, роботу з файлами.

Докладно розглянуто види бібліотек Python для роботи з геопросторовими даними, бібліотеки для роботи з картографічними проєкціями, для аналізу і маніпулювання геопросторовими даними, для візуалізації геопросторових даних. Наведено джерела геопросторових даних у векторному і растровому форматах, джерела інших типів геопросторових даних, потрібних для геопросторового аналізу і моделювання з використанням мови програмування Python у геоінформаційній системі.

Для закріплення практичних навичок програмування прикладних задач у ГІС наведено приклади застосування Python у QGIS, а саме: створення сценаріїв у консолі Python, розширення, запуск коду під час запуску QGIS, завантаження проєктів, завантаження шарів, робота з растровими і векторними шарами, оброблення геометрії, геометричні предикати та операції, підтримка проєкцій, використання полотна карти, відображення карти та друк.

Також окремим розділом описано застосування Python в ArcGIS бібліотеки ArcPy, ArcToolbox, ресурси ArcGIS Python, експорт моделей, робота з пакетом arcpy у середовищі ArcGIS та функції arcpy.

До основних результатів вивчення прикладного програмування у ГІС належить формування у студентів фахових компетентностей та програмних результатів навчання, визначених в ОПП підготовки в КНУБА бакалаврів за спеціальністю 193 «Геодезія та землеустрій», зокрема студенти повинні

– *знати:* базовий понятійно-термінологічний апарат курсу програмування в ГІС; основи програмування на Python; основні відкриті джерела геопросторових даних; основні прийоми застосування відкритих бібліотек для роботи з геопросторовими даними; основні підходи і методи, пов’язані з розробленням, керуванням та аналізом даних в ГІС; технологію використання мови програмування Python в популярних ГІС: QGIS, ArcGIS;

– *вміти:* створювати технологічні схеми для розв’язання прикладних завдань в ГІС та повторити роботу цих схем під час написання програмного коду мовою програмування Python; використовувати мову програмування Python для маніпулювання просторовими об’єктами; застосовувати набуті навички з програмування в інструментальних ГІС: QGIS, ArcGIS.

Програмний код у навчальному посібнику наведено для програмних забезпечень QGIS версій 3.28 та 3.16, а також ArcGIS for Desktop версії 10.6.

1. PYTHON ЯК МОВА ПРОГРАМУВАННЯ. БАЗОВИЙ СИНТАКСИС PYTHON

1.1. Python як мова програмування

Python («Пайтон») – проста і потужна скриптова мова програмування, розроблена Ґвідо ван Россумом (Guido van Rossum). Перший реліз системи відбувся у 1991 році. Своє ім'я мова програмування отримала на честь британської скетч-групи Монті Пайтон (Monty Python).

Рис. 1.1. Логотип Python

Python є мовою:

- скриптовою – призначеною для написання сценаріїв (script – короткий опис дій для виконання системою);
- з динамічною типізацією (dynamic typing) – типи змінних і функцій встановлюються на етапі виконання програми в момент присвоєння значення, а не в момент оголошення змінної;
- з сильною (строгою) типізацією (strong typing) – не дозволяє змішувати у виразах різні типи та не виконує автоматично неявні перетворення, наприклад, не можна додати рядок і число;
- з неявною типізацією (implicit typing) – тип нових змінних (функцій та їхніх аргументів) задається транслятором (компілятором або інтерпретатором).

Переваги Python:

- простота у використанні і потужність;
- об'єктноорієнтована мова з необов'язковим використанням прийомів ООП;
- інтеграція з іншими мовами програмування;
- кросплатформеність;
- безкоштовність інтерпретатора;

– можливість використання для математичних і наукових досліджень;
– використання в ролі базової мови написання застосунків в ArcGIS.

Останньою версією Python у теперішній час є 3.9, однак в ArcGIS використовується версія 2.7. Причиною використання багатьма розробниками версії 2.7 стало порушення зворотної сумісності під час переходу між другою і третьою версіями.

Для початку роботи з Python необхідно завантажити (наприклад, за адресою www.python.org/downloads) і встановити інтерпретатор (у багатьох Linux-системах він встановлюється разом з базовою системою). Крім того, доречно використовувати спеціальне інтегроване середовище розробки (Integrated Development Environment – IDE), хоча для початку можна обмежитися і стандартним IDLE з дистрибутиву. Велика кількість сторонніх IDE також є досить якісними і зручними у використанні продуктами.

1.2. Загальні правила синтаксису Python

Написання першої програми і використання Python як калькулятора. Робота з Python починається з запуску терміналу Python Shell (рис. 1.2)

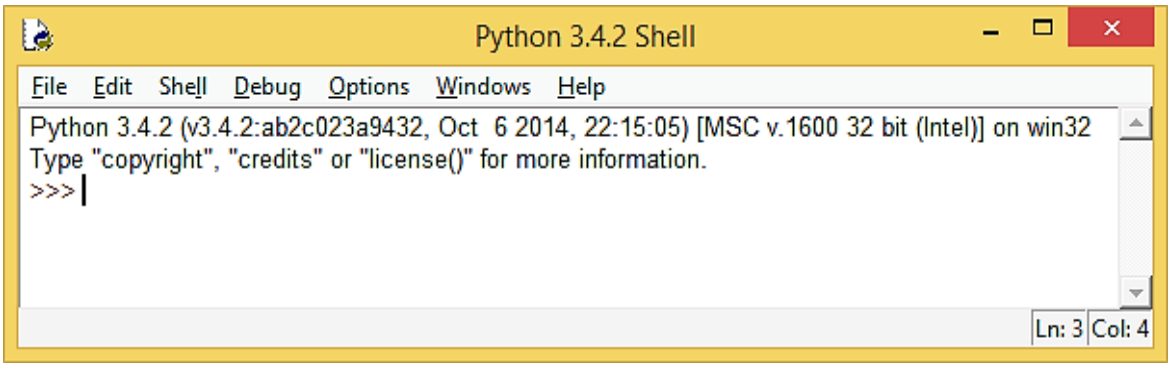

Рис. 1.2. Інтерактивна сесія роботи з Python.
Система очікує на введення команд

Напишемо нашу першу програму. Введемо у командному рядку (після символу запрошення «>>>«) print(«Hello world!»). Результат роботи представлено на рис. 1.3.

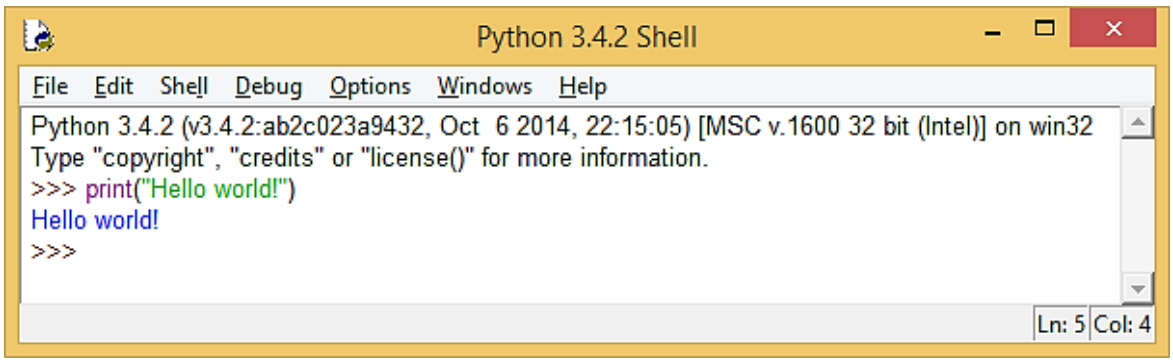

Рис. 1.3. Виконання програми «Hello world!»

Після створення цієї програми можна починати вважати себе програмістом.

Не обов'язково записувати всі команди у консолі. Можна створювати і редагувати файли у сценарному режимі IDLE, який викликається через меню File→New File. Файл з розширенням *.py може бути відредагованим в іншій IDE або текстовому редакторі, який не додає до тексту своїх символів розмітки. Для запуску файлів з розширенням *.py для виконання треба двічі клацнути лівою кнопкою миші. Якщо консольне вікно швидко закривається, вставте у кінці програми рядок «input()». В результаті інтерпретатор чекатиме на натискання клавіші Enter в кінці роботи програми.

Виконати програму, написану в сценарному режимі, можна за допомогою команди Run→Run Module або клавіші F5 (перед запуском програми її треба обов'язково зберегти). Результат виконання зображено на рис. 1.4.

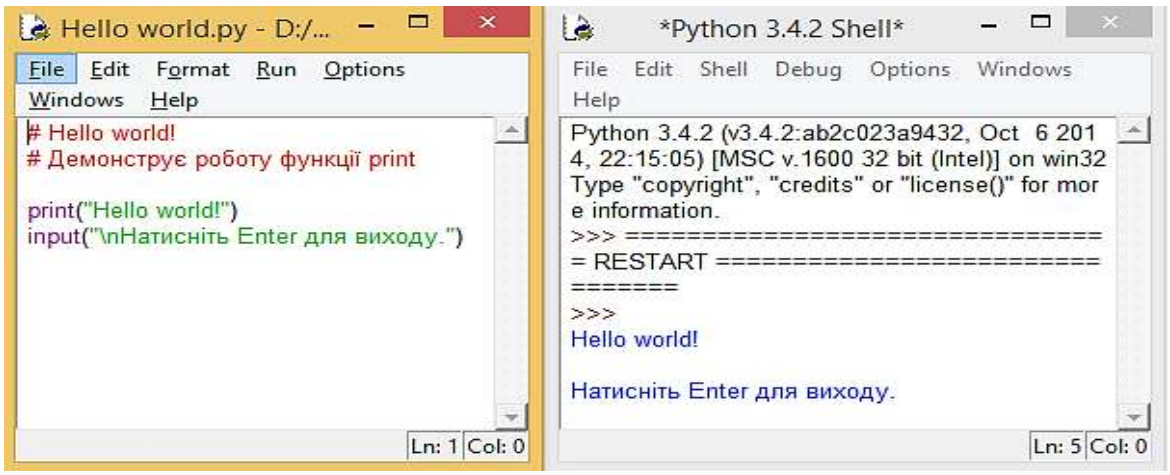

Рис. 1.4. Результат роботи програми Hello world, запущеної за допомогою IDLE

Зверніть увагу, що слова на екрані відображаються різними кольорами (функція підсвічування синтаксису). Так, зарезервовані службові слова мають фіолетовий колір, рядки – зелений, а результат роботи програми синього кольору. Схему кольорів можна змінювати в налаштуваннях.

Коментарі в Python починаються з символу «#». Всі рядки, розміщені праворуч від цього символу утворюють коментар. Добрим стилем вважають починати програму з коментарів, вказувати назву програми, її призначення, ім'я автора і дату створення. Інтерпретатор ігнорує коментарі і порожні рядки, які потрібні для покращення сприйняття коду.

1.3. Escape-послідовності

Escape-послідовності – це службові символи, які виконуються, але не відображаються на екрані під час обробці. Приклади найпростіших Escape-послідовностей наведено у табл. 1.1.

Таблиця 1.1

Escape-послідовності

Послідовність	Опис
`\n`	Перехід на новий рядок
`\t`	Знак табуляції
`\\`	Зворотній слеш. Виводить: \
`\'`	Одинарні лапки або апостроф. Виводить: '
`\"`	Подвійні лапки. Виводить: "
`\a`	Звук системного динаміка

Консоль Python можна використовувати як калькулятор, застосовуючи математичні оператори.

Математичні оператори:

`+`	додавання
`-`	віднімання
`*`	множення
`/`	ділення
`//`	ділення націло
`%`	остача від ділення

`-x`	зміна знака числа x
`abs(x)`	модуль числа x
`divmod(x,y)`	пара (x//y, x%y)
`x**y`	піднесення до степеня
`pow(x,y [,z])`	піднесення до степеня [по модулю, якщо модуль заданий]

1.4. Типи даних у Python

У Python використов – уються такі типи даних (табл. 1.2):

- цілі числа — `int`;
- числа з рухомою крапкою (дробові числа) – `float`;
- комплексні числа – complex;
- рядки – `' '` або « »;
- кортежі – `tuple`;
- списки – `list`;
- словники – `dict`;
- множини – `set` та `frozenset`.

Таблиця 1.2

Функції перетворення типів

Функція	Опис	Приклад	Результат
int(x)	Перетворює значення x у ціле число	int(″5″)	5
float(x)	Перетворює значення x у десятковий дріб	loat(″5″)	5.0
str(x)	Перетворює значення x у рядок	str(5)	'5'
bool(x)	Перетворює значення x у бульову змінну згідно зі стандартною процедурою перевірки істинності	bool(5)	True

1.5. Загальні правила синтаксису Python

Стосовно синтаксису Python є угода про те, як писати код. З текстом угоди можна ознайомитися за посиланням: http://pep8.ru/doc/pep8/.

Зазначимо деякі загальні правила синтаксису Python:

1. Кінець рядка є кінцем інструкції (крапка з комою не потрібні).

2. Вкладені інструкції об'єднуються в блоки за величиною відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блока відступ був однаковий. Традиційно відступом у Python вважають чотири пробіли.

3. Вкладені інструкції в Python записують відповідно до одного і того самого шаблону, коли основна інструкція завершується двокрапкою, за якою розміщується вкладений блок коду, зазвичай з відступом під рядком основної інструкції.

Правила створення імен змінних:

– ім'я змінної може складатися лише з цифр, літер і знаків підкреслення;

– ім'я змінної не може починатися з цифри.

Неписані правила створення імен змінних:

– ім'я повинно описувати суть змінної;

– будьте послідовними;

– поважайте традиції мови (наприклад, починайте ім'я змінної з малої літери);

– слідкуйте за довжиною імені.

1.6. Робота з рядками

Рядки – це послідовності символів з довільним доступом, які використовують для зберігання і подання текстової інформації, тому за допомогою рядків можна працювати з усім, що може бути подане у текстовій формі. Рядки належать до незмінного (immutable) типу даних.

Рядки можна виділяти як одинарними, так і подвійними лапками. Лапки одного типу можуть бути вкладеними у лапки іншого типу. Довгі рядки можна розбивати на коротші з допомогою зворотного слеша.

```
>>> str0 = "Well, I really want to encourage a kind of fantasy, \
a kind of magic. I love the term 'magic realism'"
>>> str0
"Well, I really want to encourage a kind of fantasy, a kind of magic. I love the
 term 'magic realism'"
```

Великі набори рядків або тексти можна записувати у потрійних лапках.

```
>>> str0 = """
Well, I really want to encourage \na kind of fantasy,\
\na kind of magic. \nI love the term 'magic realism'.
"""
>>> str0
"\nWell, I really want to encourage \na kind of fantasy,\na kind of magic. \nI l
ove the term 'magic realism'.\n"
>>> print(str0)

Well, I really want to encourage
a kind of fantasy,
a kind of magic.
I love the term 'magic realism'.
```

Базові операції з рядками

Конкатенація (складання)

```
>>> str1 = 'We love '
>>> str2 = 'GIS'
>>> print(str1 + str2)
We love GIS
```

Множення

```
>>> print((str1 + str2 +'\t')*3)
We love GIS     We love GIS     We love GIS
```

Обчислення довжини рядка

```
>>> len(str1*2)
16
```

Отримання доступу до елемента рядка за його індексом. До будь-якого символу рядка можна отримати доступ через його індекс. Перший символ рядка має індекс 0. Індекси можуть мати від'ємні значення. У разі відліку з кінця рахунок починається з -1.

```
>>> str3 = 'Gilliam'
>>> str3[1]
'i'
>>> str3[-3]
'i'
```

Отримання зрізу

Зріз – це механізм гнучкого керування рядком на основі індексації. Підрядок може бути виділений за допомогою зрізу – вох індексів, розділених двокрапкою, перший з яких позначає початок зрізу, а другий – кінець (символ під цим індексом не належить до зрізу).

```
>>> str3[0:4]
'Gill'
>>> str3[1:-4]
'il'
>>> str3[0:5]
'Gilli'
>>> str3[1:-3]
'ill'
>>> str3[5:]
'am'
>>> str3[:2]
'Gi'
>>> str3[:]
'Gilliam'
```

Послідовність символів з рядка можна вибирати з певною циклічністю

```
>>> str3[::1]
'Gilliam'
>>> str3[::2]
'Glim'
>>> str3[1:6:2]
'ila'
>>> str3[::-1]
'mailliG'
```

Рядковий метод – це свого роду «функція» рядка. Основна відмінність рядкового методу від функції, полягає в тому, що функція може бути викликана незалежно від будь-чого (наприклад, input()), а рядковий метод обов'язково застосовується до конкретного рядка. Щоб запустити метод, треба до імені змінної, яка посилається на рядок, додати крапку і після неї вписати назву методу і дужки (рядкові методи можуть приймати аргументи). Рядкові методи створюють нові значення, початковий рядок не змінюється (табл. 1.3, табл. 1.4).

```
>>> str3.upper()
'GILLIAM'
>>> str3
'Gilliam'
```

Таблиця 1.3

Рядкові методи

Метод	Опис
upper()	Повертає рядок, символи якого зведені до верхнього регістру
lower()	Повертає рядок, символи якого приведені до нижнього регістру
title()	Повертає рядок, в якому перша літера кожного слова велика, а решта – малі
find(str,[start],[end])	Пошук підрядка в рядку. Повертає номер першого включення або -1. Star та end задають в разі потреби інтервал пошуку в рядку
rfind(str,[start],[end])	Пошук підрядка в рядку. Повертає номер останнього включення або -1
index(str,[start],[end])	Пошук підрядка в рядку. Повертає номер першого включення або викликає ValueError
rindex(str,[start],[end])	Пошук підрядка в рядку. Повертає номер останнього включення або викликає ValueError
replace(old,new [,max])	Повертає рядок, у якому включення рядка old змінюється на new. Необов'язковий параметр max обмежує кількість замін
split(symbol)	Поділяє рядок за вказаним символом
isdigit()	Відповідає на запитання, чи складається рядок з цифр
isalpha()	Відповідає на питання, чи складається рядок з літер (розділові знаки і пробіл не літери)
isalnum()	Відповідає на питання, чи складається рядок з літер і цифр
islower()	Відповідає на питання, чи складається рядок з символів в нижньому регістрі
isupper()	Відповідає на питання, чи складається рядок з символів в верхньому регістрі
istitle()	Відповідає на питання, чи починаються слова в рядку з великої літери
startswith(str)	Відповідає на питання, чи починається рядок з шаблону str
endswith(str)	Відповідає на питання, чи закінчується рядок шаблоном str
name.join(list)	Збирання рядка зі списку з роздільником name

Закінчення табл. 1.3

Метод	Опис
capitalize()	Переводить перший символ рядка у верхній регістр, а решту – в нижній
center(width [,fill])	Повертає відцентрований рядок завширшки width, по краях якого розміщено символ fill (за замовчуванням – пробіл)
count(str [,start] [,end])	Повертає кількість вкладень підрядка, що не перетинаються, у діапазоні [start, end]. За замовчуванням start = 0, end = довжина рядка
expandtabs([tabsize])	Повертає копію рядка, у якому всі символи табуляції замінено одним або декількома пробілами, залежно від поточного стовпця. Якщо tabsize не вказаний, розмір табуляції вважають рівним восьми пробілам
lstrip([chars])	Видалення пробільних символів на початку рядка
rstrip([chars])	Видалення пробільних символів в кінці рядка
strip([chars])	Видалення пробільних символів на початку і в кінці рядка
partition(шаблон)	Повертає кортеж, що містить частину перед першим шаблоном, сам шаблон, і частину після шаблону. Якщо шаблон не знайдено, повертає кортеж, що містить два пустих рядки, а потім сам рядок
rpartition(sep)	Повертає кортеж, що містить частину перед останнім шаблоном, сам шаблон і частину після шаблону. Якщо шаблон не знайдено, повертає кортеж, що містить два пустих рядка, а потім сам рядок
swapcase()	Переводить символи нижнього регістру у верхній, а верхнього – у нижній
zfill(width)	Робить довжину рядка не меншою за width, за потреби заповнюючи перші символи нулями
ljust(width, fillchar)	Робить довжину рядка не меншою за width, за потреби заповнюючи останні символи символом fillchar
rjust(width, fillchar)	Робить довжину рядка не меншою за width, за потреби заповнюючи перші символи символом fillchar
format(*args, **kwargs)	Форматує рядок

Таблиця 1.4

Складені оператори присвоєння

Оператор	Зразок	Рівносильний запис
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5

1.7. Логічні оператори

У Python будь-яке значення можна розглядати як істинне (**`True`**) або хибне (**`False`**) і використовувати як умову. Більшість значень (від'ємні і додатні числа та рядки) є логічно істинними, хибними вважають пусті рядки і число 0.

Логічні оператори:

– **`and`** – логічне «і». Якщо обидва операнда, поєднані оператором and, true, то і весь вираз true. Якщо обидва операнда істинні, результатом буде останнє значення. Якщо будь-який з операндів хибний, результатом буде перше хибне значення.

```
>>> 'a' and 'b'
'b'
>>> ' ' and 'b'
'b'
>>> '' and 'b'
''
>>> 'a' and 'b' and 'c'
'c'
```

– **`or`** – логічне або. Якщо хоча б один з операндів, поєднаних оператором or, true, то і весь вираз true. Якщо операнд істинний, or негайно повертає результат, інші операнди ігноруються.

```
>>> 'a' or 'b'
'a'
>>> ' ' or 'b'
' '
>>> '' or 'b'
'b'
>>> 'a' or print('Hello')
'a'
```

Приклад використання **and** та **or** в одному виразі:

```
>>> 1 and 2 or 3
2
>>> 0 and 2 or 3
3
>>> 1 and '' or 3
3
```

– **not** – логічне «не». Використовують для зворотної зміни логічного результату виразу. Якщо вираз true, то після застосування оператора not він стає false, і навпаки.

```
>>> a = 1
>>> not a
False
```

2. РОЗГАЛУЖЕННЯ І ЦИКЛИ В PYTHON

2.1. Модулі і генерація випадкових чисел

Модулі – це файли, що містять код, який можна використовувати в інших програмах. Модуль зазвичай являє собою сукупність функцій, які належать до однієї і тієї самої області. Функції з імпортованих модулів викликаються за допомогою точкової нотації: назва_модуля.ім'я_функції.

Функції, пов'язані з генерацією випадкових чисел й отриманням випадкових результатів, містяться в модулі *random*. З цього модуля можна використовувати такі функції:

– *randint*()– приймає два цілочисельних аргументи і повертає випадкове ціле число, що лежить в діапазоні між ними;

– *randrange*() – повертає випадкове ціле число, від 0 (включно) до введеного як аргумент цілого числа (не включно).

```
>>> import random
>>> random.randint(5,10)
8
>>> random.randrange(10)
6
```

2.2. Умовні конструкції з «If»

Конструкція «If» дозволяє програмі обрати, виконувати певний фрагмент коду чи пропустити його, залежно від того, що вказав користувач. У разі використання конструкції «If» завжди задається *умова*, тобто вираз – істинний або хибний.

Блок – це один або декілька рядків з однаковим відступом, єдина конструктивна одиниця. Відступ треба вказувати пробілами (стандартно – чотирма) або знаками табуляції, однак не рекомендується використовувати обидва варіанти в одній програмі.

Синтаксис умовної конструкції «If» (найпростіша умовна конструкція):

```
if <умова>:
   <блок>
```

Якщо <умова> дотримується, то <блок> виконується, якщо ні – пропускається (табл. 2.1).

```
>>> if 5==5.0:
        print('5=5.0')


5=5.0
>>> if 5=='шість':
        print('5=6')


>>>
```

Таблиця 2.1

Оператори порівняння

Оператор	Значення	Приклад	Істинність
==	Дорівнює	7 == 7	True
!=	Не дорівнює	7 != 7	False
>	Більше	7 > 5	True
<	Менше	7 < 5	False
>=	Більше або дорівнює	7 >= 5	True
<=	Менше або дорівнює	7 <= 5	False

Умовна конструкція з використанням **else**:

```
if <умова>:
<блок 1>
else:
<блок 2>
```

```
>>> if 'Python' < 'Visual Basic':
        print ('Python better than Visual Basic')
else:
        print('Visual Basic better than Python')


Python better than Visual Basic
```

Якщо <умова> є істинною, виконується <блок 1>, якщо вона не істинна – <блок 2>.

Ця конструкція також може бути записана в короткій формі:

```
<блок 1> if <умова> else <блок 2>
```

Умовна конструкція з додатковими умовами **elif** (скорочено від `else if`) і необов'язковим `else` в кінці. Буде виконано блок, який йде після першої умови, що набула значення `True`. Якщо всі умови набудуть значення `False`, блок буде виконаний після фінального `else`.

```
if <умова 1>:
    <блок 1>
elif <умова 2>:
    <блок 2>
…
elif <умова N>:
    <блок N>
else:
    <блок N + 1>
```

```
print ('З допомогою цієї програми ви дізнаєтеся свою \
оцінку по п'ятибальній системі.')
mark = int(input('Введіть кількість балів, яку ви набрали протягом \
семестру з дисципліни:'))
if mark >=90:
    print ('\nВаша оцінка п'ять. Держава може дати вам підвищену стипендію.')
elif mark >=74:
    print ('\nВаша оцінка чотири. Держава може дати вам звичайну стипендію.')
elif mark >=60:
    print ('\nВаша оцінка три. Вам вдалося спекатися цього предмету.')
else:
     print ('\nВи не набрали достатньої кількості балів. До зустрічі\
наступного року.')
```

2.3. Цикл «while»

Синтаксис циклу **while**:

```
while <умова>:
<блок 1>
else: # необов'язкова частина
<блок 2> # виконується, якщо вихід з циклу був
здійснений не інструкцією break
```

break – здійснює вихід за межі циклу

continue – здійснює перехід на початок циклу (до рядка-заголовка)

```
# Програма обчислення факторіалу числа
f = int(input("Введіть число, факторіал якого необхідно знайти: "))
factr = 1
i = 0
while i < f:
    i += 1
    factr = factr * i
print("Факторіал", f, "дорівнює", factr)
```

У циклі while блок коду буде виконуватися, якщо умова є істинною, доки не виявиться хибною. Контроль над циклом while здійснює керівна змінна (та, що є в умові і зіставляється з деяким значенням/значеннями), яку слід ініціалізувати перед запуском циклу (якщо керівна змінна не матиме ніякого значення в момент першого зіставлення з об'єктом, програма видасть помилку). Початкова умова циклу while повинна бути хибною, інакше тіло циклу не буде виконане.

Програма з нескінченним циклом (у момент запуску «зависає»):

```
number = 0
while number <=5:
    print(number)
```

Приклад використання інструкції continue:

```
# пошук парних чисел
print("Знайдемо усі парні числа між 0 і 15")
n = 15
while n:
    n -= 1
    if n % 2 != 0: continue
    print(n, end = " ")
```

2.4. Цикл «for»

Цикл «for» призначений для перебору заданої множини елементів і виконання над ними різних операцій у тілі циклу.

Синтаксис циклу for:

```
for <елемент> in <об'єкт ітерації>:
    <блок>
```

До елементів послідовностей можна звертатися по індексу за формою <змінна>.<індекс>. Якщо у рядку 10 символів, то їх нумерують індексами від 0 до 9, або від −10 до −1.

Приклади використання циклу «`for`»:

```
>>> for i in "GIS":
        print(i, end = '\n')


G
I
S
```

```
# пошук цисел кратних 3 у заданому діапазоні
print("Пошук цисел кратних 3 у заданому діапазоні")
l = int(input("Введіть ліву межу діапазону пошуку: "))
r = int(input("Введіть праву межу діапазону пошуку: "))
print("\nВсі числа діапазону:")
for i in range(l,r+1):
    print(i, end = ' ')
print("\nЧисла кратні 3:")
for i in range(l,r+1):
    if i%3 == 0:
        print(i, end = ' ')
```

2.5. Функція `range`

Функція **`range()`** – вбудована функція Python, що створює список чисел (арифметичну прогресію), який, як правило, використовується для перебору. Синтаксис функції:

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

Аргументи функції range() повинні бути цілими числами. За замовчуванням значення аргументу start – 0, step – 1.

```
>>> for k in range(5,0,-1):
        print(k,end = ' ')


5 4 3 2 1
```

3. ПОСЛІДОВНОСТІ І ФУНКЦІЇ В PYTHON

Послідовності в Python – ітерабельні об'єкти, до елементів яких є ефективний доступ з використанням цілочисельних індексів. Доступ може бути здійснений через спеціальний метод ***__getitem__(self, key)***, підтримується також метод ***__len__()*** для повернення довжини послідовності.

`__getitem__(self, key)` – #метод, де доступ за індексом (або ключем).

Послідовності поділяються на змінні (елементи можна модифікувати) та незмінні. Рядки належать до незмінних послідовностей (наприклад, неможливо надати нове значення доступному за індексуом елементу рядка).

3.1. Список

У Python немає масивів у традиційному розумінні цього терміна. Замість них з метою зберігання однорідних (і неоднорідних) об'єктів використовують списки.

Список (`list`) – впорядкований змінюваний набір об'єктів довільних типів (табл. 3.1). Для створення списку треба взяти у квадратні дужки послідовність розділених комами значень. Списки задають трьома способами:

простий перелік

```
>>> l = [1, 1.1, "GIS"]
>>> l
[1, 1.1, 'GIS']
```

перетворення рядка у список

```
>>> l = list("GIS")
>>> l
['G', 'I', 'S']
```

використання спискових включень (у прикладі містяться квадрати всіх парних чисел від 0 до 9)

```
>>> l = [x**2 for x in range(10) if x%2 == 0]
>>> l
[0, 4, 16, 36, 64]
```

Обравши елемент за списком йому можна надати нове значення, однак створити таким способом новий елемент неможливо. Якщо надати значення елементу, якого раніше не було, виникне помилка.

Видалити елемент списку можна командою **`del`**, довжина списку зменшиться на одиницю.

Таблиця 3.1

Методи списків

Метод	Опис
append(значення)	Додає значення до кінця списку
clear()	Очищення списку; з'явився у Python 3.3. Те саме, що і **del l[:]**, де l – назва списку
copy()	Поверхнева копія списку; з'явився у Python 3.3. Те ж саме, що і **l[:]**, де l – назва списку
count(значення)	Повертає кількість значень у списку
extend(L)	Розширює список, додаючи в кінці всі елементи списку L
index(x, [start [, end]])	Повертає положення (індекс) першого елемента x. Необов'язкові параметри start та end задають діапазон пошуку
insert(i, x)	Вставляє у i-й елемент списку значення x
pop([i])	Повертає значення в i-й позиції та видаляє цей елемент зі списку. Якщо параметр i методу не переданий, повертається і видаляється останній елемент зі списку
remove(x)	Видаляє зі списку перший елемент, що має значення x
reverse()	Розвертає порядок елементів у списку
sort()	Сортує елементи за зростанням. Якщо надати параметру методу reverse значення True, список буде відсортований за спаданням

Методи списків, на відміну від рядкових методів, змінюють сам список, тому результат виконання не потрібно записувати у змінну.

```
>>> subjects = ["Office","ArcGIS","MapInfo"]
>>> print(subjects, "- Інженер-геоінформатик має володіти цим програмним
забезпеченням")
['Office', 'ArcGIS', 'MapInfo'] - Інженер-геоінформатик має володіти цим
програмним забезпеченням
>>> print("На першому курсі було освоєно", subjects.pop(0))
На першому курсі було освоєно Office
>>> print("Залишилося освоїти", subjects)
Залишилося освоїти ['ArcGIS', 'MapInfo']
>>> print("Хоча знання Python теж не завадить.")
Хоча знання Python теж не завадить.
>>> subjects.append("Python"); print("Тому варто освоїти", subjects)
Тому варто освоїти ['ArcGIS', 'MapInfo', 'Python']
>>> print("Однак цими", len(subjects), "програмами список не вичерпується
")
Однак цими 3 програмами список не вичерпується
```

3.2. Кортеж

Кортеж (`tuple`) — це незмінюваний список об'єктів. Для створення кортежу треба взяти у круглі дужки послідовність розділених комами значень (не обов'язково однорідних).

Кортеж можна записувати в одному або кількох рядках. Кортеж можна використовувати у лівій частині оператора присвоювання. Значення кортежу у лівій частині оператора присвоювання пов'язуються з аналогічними елементами правої частини. Кортеж може складатися з одного елемента.

```
>>> t = (4, 4.5, 'Hi')
>>> t
(4, 4.5, 'Hi')
>>> (a, b, c) = t
>>> print (a, b, c)
4 4.5 Hi
>>> d, e, f = t
>>> print(d, e, f)
4 4.5 Hi
>>> a = 5; b = 10
>>> a, b = b, a
>>> print(a, b)
10 5
>>> g = 8
>>> g
8
>>> g = 8,
>>> g
(8,)
```

Довжина кортежу обчислюється функцією `len()`:

```
>>> len(t)
3
>>> len(g)
1
```

Списки мають таку ж функціональність як і кортежі, а також деякі додаткові можливості, завдяки яким вони стають значно гнучкішим інструментом. Однак в деяких випадках доцільно користуватися саме кортежами.

Переваги використання кортежів:

- кортежі працюють швидше;
- незмінюваність кортежів дає змогу використовувати їх для створення констант;
- кортежі застосовують в деяких структурах даних, від яких Python вимагає незмінних значень.

Вкладені послідовності – це списки (кортежі), які містяться всередині інших списків (кортежів). Недоречно використовувати вкладення з глибиною більш ніж два, оскільки це провокує помилки.

Для доступу до елементів вкладених послідовностей застосовують множинну індексацію. Наприклад, `list[x][y]` звертається до елемента списку list під номером x, а з нього, в свою чергу, береться елемент, що стоїть у позиції y.

Розпаковування послідовності – створення для кожного елемента послідовності своєї змінної.

```
>>> personal_data = [("Андрій","Шевченко"),'0442454545']
>>> name, surname, phone = personal_data
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    name, surname, phone = personal_data
ValueError: not enough values to unpack (expected 3, got 2)
>>> personal, phone = personal_data
>>> print(phone)
0442454545
>>> name, surname = personal
>>> print(name)
Андрій
>>> print(surname)
Шевченко
```

3.3. Словник

Словник (`dictionary`) – невпорядкована змінювана структура довільних даних, призначена для зберігання пар елементів (ключ – значення). Інколи словники називають асоціативними масивами або хеш-таблицями.

Аналогічно до списків з кортежами словники можна записувати або в одному рядку, або розділяти на декілька рядків, як повинні закінчуватися комами. У словниках немає номерів позицій, доступ до значень здійснюється за допомогою ключа; звернутися до ключа через значення неможливо. Використання неіснуючого ключа спричинює помилку інтерпретатора.

Методи створення словників (табл. 3.2):

1. За допомогою літерала (виразу, що створює об'єкт). Потрібно ввести ключ, поставити двокрапку і ввести відповідне ключу значення; пари «ключ – значення» розділити комами; весь набір взяти у фігурні дужки.
2. За допомогою функції **`dict`**.
3. За допомогою методу **`fromkeys`**.
4. За допомогою генератора словників.

Нові пари значень додаються у словник таким способом:

словник[новий ключ] = нове значення

Видалення пар значень із словника виконують так:

del словник[ключ]

За спроби видалити зі словника неіснуючу пару інтерпретатор видає помилку.

Особливості словників:

- кожен ключ повинен бути унікальним;
- ключ повинен бути незмінного типу (рядок, число або кортеж);
- значення можуть бути не унікальними і змінними.

Таблиця 3.2

Словникові методи

Метод	Опис
clear()	Очищує словник
copy()	Повертає копію словника
fromkeys(seq[, value])	Створює словник з ключами seq і значенням value (за замовчуванням None)
get(key[, default])	Повертає значення за отриманим ключем. Якщо ключа в словнику немає, повертає значення None або значення за замовчуванням, якщо задане
items()	Повертає набір усіх пар у словнику. Кожна пара являє собою кортеж з двох елементів: перший – ключ, другий – відповідне значення
keys()	Повертає набір усіх ключів словника
pop(key[, default])	Видаляє ключ і повертає його значення. Якщо ключа немає, повертає default (за замовчуванням викликає виключення)
setdefault(key[, default])	Повертає значення ключа, але якщо його немає, створює у словнику новий ключ зі значенням default (за замовчуванням None)
update([other])	Оновлює словник, додаючи пари значення – ключ з other. Існуючі ключі переписуються.
values()	Повертає набір усіх значень словника

```
>>> mobile = {"050":"МТС","063":"Life","066":"МТС",
          "067":"Київстар","068":"Київстар","073":"Life",
          "091":"ТриМоб","092":"Peoplenet","093":"Life",
          "094":"Інтертелеком","095":"МТС","096":"Київстар",
          "097":"Київстар","098":"Київстар","099":"МТС"}
>>> mobile.values()
dict_values(['Київстар', 'МТС', 'МТС', 'Life', 'Life', 'ТриМоб', 'Peoplenet', 'M
TC', 'Київстар', 'Київстар', 'Київстар', 'Life', 'Інтертелеком', 'Київстар', 'MT
C'])
>>> mobile.get('092')
'Peoplenet'
```

3.4. Функції

Функція – це окремий блок команд, який може неодноразово використовуватися в роботі програми. Функція – це об’єкт, який приймає аргументи і повертає значення. Функція може бути будь-якої складності і повертати будь-які значення (навіть інші функції).

Порядок оголошення функції такий:

1. Пишемо зарезервоване слово **def** (скорочено від define), яке означає оголошення функції.

2. Вводимо ім'я функції. Принципи іменування функцій такі самі як і для змінних.

3. У круглих дужках вказуємо параметри функції – змінні, значення яких будуть використані під час виклику функції і передані їй. Функція може не мати параметрів.

4. У наступному рядку з відступом пишемо тіло функції. У Python функції з пустим тілом заборонені. Функція може закінчуватися словом **return**, після нього вказують параметр (кілька параметрів, розділених комами), який функція передає програмі.

5. До функції можна давати пояснення за допомогою **рядка документації**. Це обов'язково має бути перший рядок тіла функції, взятий у потрійні лапки. IDE використовують рядок документації як підказку. Рядок документації можна викликати за допомогою методу __doc__.

Приклад функції:

```
def safe_div(x,y):
    """Do a safe division :-)
for fun and profit"""
    if y != 0:
        z = x / y
        return z
    else:
        print ("Yippie-kay-yay!")
```

Використання створеної функції:

```
>>> safe_div(5,6)
0.8333333333333334
>>> safe_div(5,0)
Yippie-kay-yay!
>>> print(safe_div.__doc__)
Do a safe division :-)
for fun and profit
>>> new_func = safe_div
>>> new_func(5,6)
0.8333333333333334
```

Зверніть увагу, що у Python функції є значеннями, які можна присвоювати.

Приклад функції від Гвідо ван Россума:

```
def gcd(a, b):
   """Знаходження найменшого спільного дільника
   двох чисел"""
   while a != 0:
      a,b = b%a,a
   return b
```

Усім параметрам функції можна задавати параметри за замовчуванням (ці значення будуть передаватися функції, якщо її виклик не містить аргументів). Після виклику функції вказані параметри зіставляються зліва направо, а решта набувають значень за замовчуванням (якщо вони задані конкретно). Значення за замовчуванням слід надавати або всім параметрам функції, або жодному.

```
>>> def login (username = "student", password = None):
        print("username = ", username, ";", "password = ", password)


>>> login("root", "123456789")
username =  root ; password =  123456789
>>> login("guest")
username =  guest ; password =  None
>>> login()
username =  student ; password =  None
>>> login(password = "qwerty")
username =  student ; password =  qwerty
```

Особливістю Python є те, що значення за замовчуванням обчислюються і асоціюються лише один раз – у момент оголошення функції.

```
>>> val = 5
>>> def our_val (my_val = val):
        print(my_val)


>>> our_val(3)
3
>>> our_val()
5
>>> val = 10
>>> our_val()
5
```

Функція може приймати довільну кількість позиційних аргументів (значення параметрів визначаються, виходячи лише з порядку, в якому наведено аргументи, що передаються функції в разі її виклику: перший параметр отримує значення першого аргументу, другий – параметр другого аргументу і т. д.) У такому разі перед іменем аргументів ставлять «*».

```
>>> def function (*arguments):
        return arguments

>>> function(1, 2, 3, 'q', 'w', 'e')
(1, 2, 3, 'q', 'w', 'e')
>>> function()
()
>>> function('q')
('q',)
```

Функція може приймати будь-яку кількість поіменованих аргументів (значення аргументів безпосередньо надаються параметрам під час виклику функції). У такому разі перед іменем аргументів ставлять «**». Іменовані аргументи дають змогу передавати значення у будь-якому порядку і вносять ясність у код.

```
>>> def function(**arguments):
        return arguments

>>> function(a = 1, b = 2, c = 3)
{'c': 3, 'b': 2, 'a': 1}
>>> function()
{}
>>> function (GIS = 'gvSIG')
{'GIS': 'gvSIG'}
```

Lambda-функція – це функція, яка має довільну кількість аргументів (зокрема й необов'язкові аргументи) і повертає значення одного виразу. Lambda-функції не можуть містити інструкцій або більше одного виразу. Lambda-функції були запозичені з Lisp. Lambda-функція не має імені, але може бути викликана через змінну, якій вона надається.

Синтаксис lambda-функції:

lambda <змінні>: <вираз>

змінні – список аргументів, розділених комами. Аргументи не потрібно брати у дужки;

вираз – вираз, результат якого повертається функцією (ключового слова «return» немає). Область видимості виразу містить локальні змінні і аргументи.

```
>>> def sum (x,y):
        return x + y

>>> sum2 = lambda x, y: x+y
>>> sum(1,1)
2
>>> sum2(1,1)
2
```

Lambda-функції можна використовувати, не надаючи їх змінним:

```
>>> (lambda x: x*x)(4)
16
```

Перевагою застосування lambda-функції є те, що вона є виразом і може бути використана всередині іншого виразу.

```
>>> for l in map(lambda i: i.upper(), ['g', "i", 's']):print(l, end ="")

GIS
```

Функція map приймає як аргумент функцію і список та застосовує функцію до кожного елемента списку.

3.5. Області видимості

Область видимості – це спосіб представлення різних частин програми, відділених одна від одної. Будь-яка змінна, створена у глобальній області видимості, називається *глобальною*, а змінна, створена всередині функції, *локальною* (рис. 3.1).

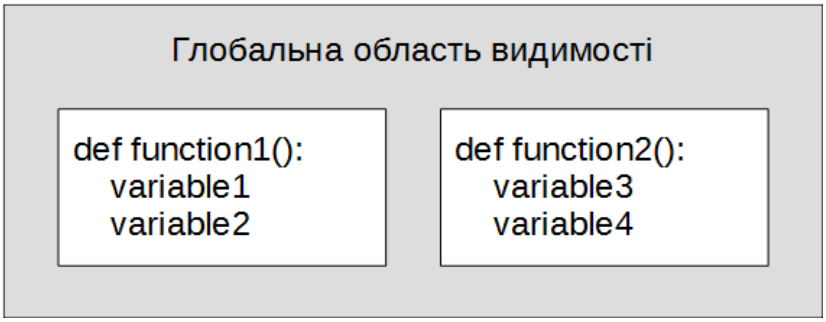

Рис. 3.1. Глобальна і дві локальні області видимості

Всередині функції можна змінити значення глобальної змінної за допомогою службового слова **global**.

```
>>> x = 5
>>> def func():
        global x
        print ('x дорівнює', x)
        x = 2
        print('Змінюємо глобальне значення x на', x)


>>> func()
x дорівнює 5
Змінюємо глобальне значення x на 2
>>> print(x)
2
```

У наведеному прикладі зарезервоване слово global використовується для оголошення того, що x – це глобальна змінна, тому зміна значення x всередині функції змінить значення x в основному блоці програми.

4. РОБОТА З ФАЙЛАМИ І ОБРОБЛЕННЯ ВИНЯТКОВ У PYTHON. ФУНКЦІЇ

Python дає змогу зчитувати рядки з текстових файлів, які зручно використовувати для тривалого зберігання нескладних даних (рис. 4.1). Крім того, у більшості операційних систем є базові інструменти для перегляду та редагування тексту.

Розглянемо роботу з текстовими файлами на прикладі.

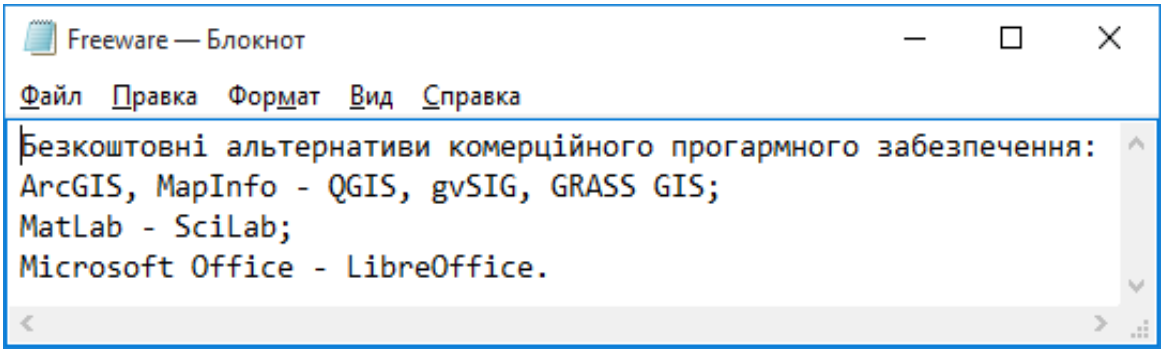

Рис. 4.1. Вміст файлу Freeware.txt

4.1. Відкриття і закриття файла

Для отримання інформації з файла його потрібно спочатку відкрити за допомогою функції open.

`open(name)` – вбудована функція, що відкриває файл «name»; якщо не вказати повний шлях до файла, програма шукатиме файл «name» у тій самій директорії, в якій знаходиться сама програма. Додатковими аргументами функції можуть бути режим доступу до текстових файлів, параметр кодування та інші.

```
>>> f = open('D:\Workspace\Freeware.txt', 'r', encoding = 'utf-8')
```

Таблиця 4.1

Режими доступу до текстових файлів

Режим	Опис
'r'	Читання з текстового файла (режим за замовчуванням). Якщо файла немає, Python повідомить про помилку
'w'	Запис у текстовий файл. Якщо файл існує, його вміст буде видалений і замінений на новий, якщо файл не існує, він буде створений

Закінчення табл. 4.1

Режим	Опис
'a'	Запис у текстовий файл. Якщо файл існує, нові дані будуть дописані у кінці, якщо файл не існує, він буде створений
'b'	Відкриття файла у двійковому режимі
't'	Відкриття файла у текстовому режимі (режим за замовчуванням)
'r+'	Читання і запис у текстовий файл. Якщо файла не існує, Python повідомить про помилку
'w+'	Запис і читання з текстового файлу. Якщо файл існує, його вміст буде замінений, якщо файл не існує, він буде створений
'a+'	Запис і читання з текстового файла. Якщо файл існує, нові дані будуть дописані у кінець, якщо файл не існує, він буде створений

Режими можна об'єднувати, наприклад 'rb' – зчитування у двійковому режимі, якщо файла не існує, Python повідомить про помилку.

Відкритий файл став доступним через змінну «f», яка являє собою файловий об'єкт. До файлових об'єктів можна застосовувати методи, наведені у табл. 4.2.

Таблиця 4.2

Файлові методи

Метод	Опис
read()	Зчитує з файла вказану кількість символів і повертає ці символи як рядок. Якщо кількість символів не вказано, весь текстовий файл повертається одним рядком
close()	Закриває файл. Закритий файл не може читатися і записуватися доти, доки не буде відкритий знову
readline()	Зчитує символи з поточного рядка тексту. Якщо передати методу кількість символів від початку рядка, він поверне ці символи як рядок. Якщо не передавати жодного числа, він прочитає усі символи з поточної позиції до кінця рядка. Якщо усі символи рядка прочитані, місце поточного рядка займає наступний за ним

Закінчення табл. 4.2

Метод	Опис
readlines()	Зчитує текстовий файл у список, елементами якого стають рядки
write()	Записує рядок у файл. Якщо потрібно, щоб у файлі було декілька рядків, між ними необхідно записувати символи \n
writelines()	Записує список рядків тексту у файл

Використання методу `close()`:

```
>>> f = open('D:\Workspace\Freeware.txt', 'r', encoding = 'utf-8')
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    f.read()
ValueError: I/O operation on closed file.
```

4.2. Читання текстового файлу

У разі посимвольного читання файла методом `read()` Python запам'ятовує, де була зупинка під час читання попереднього разу і наступний виклик `read()` починається з місця, де закінчив роботу попередній. Якщо дочитати файл до кінця, то черговий виклик `read()` поверне пустий рядок.

```
>>> f = open('D:\Workspace\Freeware.txt', 'r', encoding = 'utf-8')
>>> print(f.read(5))
 Безк
>>> print(f.read(5))
оштов
>>> print(f.read())
ні альтернативи комерційного прогармного забезпечення:
ArcGIS, MapInfo - QGIS, gvSIG, GRASS GIS;
MatLab - SciLab;
Microsoft Office - LibreOffice.
>>> print(f.read())

>>>
```

Для того щоби повернутися на початок фала, його слід закрити і знову відкрити.

Символи з поточного рядка тексту можливо читати методом `readline()`:

```
>>> f.close()
>>> f = open('D:\Workspace\Freeware.txt', 'r', encoding = 'utf-8')
>>> print(f.readline(5))
 Безк
>>> print(f.readline())
оштовні альтернативи комерційного прогармного забезпечення:

>>> print(f.readline())
ArcGIS, MapInfo - QGIS, gvSIG, GRASS GIS;

>>>
```

Зверніть увагу, що разом з кожним рядком текстового файла на екран виводиться пустий рядок. Це зумовлене тим, що кожен з рядків закінчується escape-символом '\n'.

Читання файла у список:

```
>>> f = open('D:\Workspace\Freeware.txt', 'r', encoding = 'utf-8')
>>> lines = f.readlines()
>>> print(lines)
['\ufeffБезкоштовні альтернативи комерційного прогармного забезпечення:\n', 'Arc
GIS, MapInfo - QGIS, gvSIG, GRASS GIS;\n', 'MatLab - SciLab;\n', 'Microsoft Offi
ce - LibreOffice.']
>>>
```

Перебір рядків файла:

```
>>> f.close()
>>> f = open('D:\Workspace\Freeware.txt', 'r', encoding = 'utf-8')
>>> for line in f:
        print(line, end='')


 Безкоштовні альтернативи комерційного прогармного забезпечення:
ArcGIS, MapInfo - QGIS, gvSIG, GRASS GIS;
MatLab - SciLab;
Microsoft Office - LibreOffice.
```

Як і для читання, для запису файл потрібно відкрити у певному режимі.

```
>>> f = open('D:\Workspace\Freeware_new.txt', 'w', encoding = 'utf-8')
>>> f.write('Nero Burning Rome - CDBurnerXP\n')
31
>>> f.write('Symantec Endpoint Protection - Comodo Internet Security\n')
56
>>> f.close()
```

Файл *Freeware_new.txt* створюється як пустий текстовий файл, готовий для запису будь-якого тексту з програми. Якби файл *Freeware_new.txt* вже існував, то він був би замінений на пустий файл, а попередній вміст цього файла був би стертий (рис. 4.2). Метод write() не вставляє автоматично символ переходу на новий рядок після записаного у файл рядка.

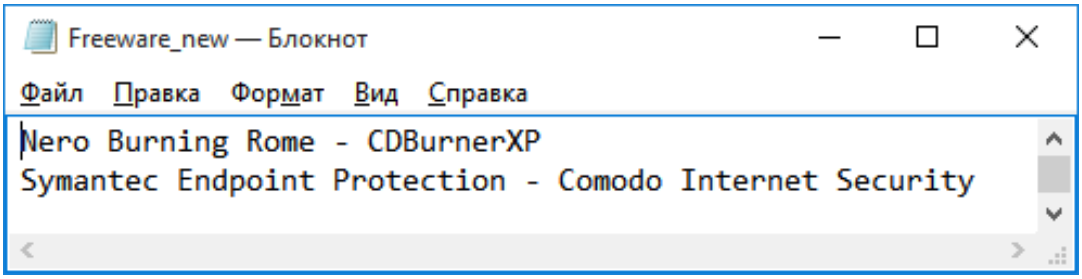

Рис. 4.2. Вміст файла Freeware_new.txt.

4.3. Збереження структурованих даних у файлах

Перевагою текстових файлів як засобу збереження даних є те, що типовий набір операцій з ними доступний у будь-якому текстовому редакторі, однак їхня функціональність обмежена, оскільки вони можуть зберігати лише послідовності символів. В то же час може виникнути потреба зберігати дані більш складної природи: списки або словники.

Для консервації (тобто збереження в недоторканній формі у файлі) даних потрібне під’єднання двох нових модулів:

pickle – консервування і зберігання структур даних у файлах;

shelve – забезпечення довільного доступу до законсервованих об'єктів у файлах.

Законсервовані дані зберігаються у формі послідовності байтів, доступ на запис і читання – послідовний.

Створюємо три списки і відкриваємо на запис новий файл:

```
>>> import pickle, shelve
>>> variety = ["Помідори", "Капуста", "Груші"]
>>> shape = ['Цілі', 'Кубиками', 'Кружальцями']
>>> brand = ["Чумак", "Верес", "Бондюель"]
>>> f = open('D:\Workspace\pickles1.dat', "wb")
```

Консервуємо списки і зберігаємо їх у файл pickles1.dat (рис. 4.3):

```
>>> pickle.dump(variety, f)
>>> pickle.dump(shape, f)
>>> pickle.dump(brand, f)
>>> f.close()
```

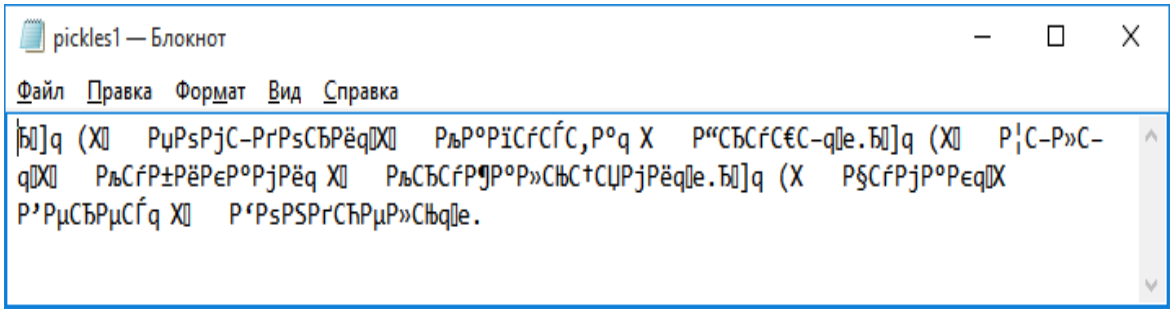

Рис. 4.3. Вміст файла pickles1.dat

Розконсервація даних (табл. 4.3):

```
>>> f = open('D:\Workspace\pickles1.dat', "rb")
>>> variety = pickle.load(f)
>>> shape = pickle.load(f)
>>> brand = pickle.load(f)
>>> print(variety, '\n', shape, '\n', brand)
['Помідори', 'Капуста', 'Груші']
 ['Цілі', 'Кубиками', 'Кружальцями']
 ['Чумак', 'Верес', 'Бондюель']
>>> f.close()
```

Таблиця 4.3

Функції консервації і розконсервації

Функція	Опис
dump(об'єкт, файл[, bin])	Записує законсервовану версію об'єкта у файл. Якщо параметр bin = True, об'єкт записується у двійковому форматі, якщо False (за замовчуванням) – у текстовому форматі
load(файл)	Розшифровує черговий законсервований об'єкт з файла і повертає його

4.4. Полиці для зберігання консервованих даних

За допомогою модуля shelve можна створити полицю, яка працює подібно до словника та у якій передбачено довільний доступ до елементів.

Створення полиці:

```
>>> s = shelve.open('D:\Workspace\pickles2.dat')
```

Функція shelve.open() працює аналогічно до функції open() з тією різницею, що shelve.open() відкриває не текстовий файл, а файл з консервованими об'єктами. Ця функція приймає один аргумент – ім'я файла. Іншим необов'язковим аргументом є режим доступу.

Додаємо на полицю три списки:

```
>>> s = shelve.open('D:\Workspace\pickles2.dat')
>>> s['variety'] = ['Гриби', "Горох", "Кукурудза"]
>>> s['shape'] = ['Цілі', 'Кубиками', 'Соломкою']
>>> s['brand'] = ['Наша ціна', 'Вигідна ціна', 'Порядна ціна']
```

Полиця s за функціональністю не відрізняється від словника (однак модуль shelve ефективніше працює з пам'яттю). Ключу «variety» відповідним є значення ['Гриби', «Горох», «Кукурудза»]. Ключем у складі полиці може бути лише рядок.

Переконаємося, що всі зміни внесено у файл:

```
>>> s.sync()
```

Файл полиці також оновлюється після його закриття методом close().

Дістанемо з полиці збережений на ній список (табл. 4.4):

```
>>> print("Торгові марки: ", s['variety'])
Торгові марки:  ['Гриби', 'Горох', 'Кукурудза']
```

Таблиця 4.4

Режими доступу до полиці

Режим	Опис
«c»	Відкриття файла на читання або запис. Якщо файл не існує, він буде створений
«n»	Створення нового файла для читання або запису. Якщо файл існує, його вміст буде змінений

Закінчення табл. 4.4

Режим	Опис
«r»	Читання з файлу. Якщо файлу не існує, Python повідомить про помилку
«w»	Запис у файл. Якщо файлу не існує, Python повідомить про помилку

4.5. Опрацювання винятків

Опрацювання виключень (`exception handling`) – механізм, призначений для опису реакції програми на помилки часу виконання та інші можливі проблеми (виключення), які можуть виникнути під час виконання програми і призводять до втрати сенсу (до неможливості) подальшої обробки програмою її базового алгоритму.

Виникнення помилки:

```
>>> num = float("Hello")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    num = float("Hello")
ValueError: could not convert string to float: 'Hello'
```

Спроба конвертувати рядок у число з рухомою комою згенерувала виняток ValueError, оскільки символи в рядку мають неочікуване значення (це не цифри і не символ розділення цілої і дробової частини у десятковому дробі).

Типовим способом опрацювання («перехоплення») винятків є конструкція **`try/except`**. Спочатку записують оператор try, який відділяє фрагмент коду, що здатен викликати виняток. Потім пишуть оператор except і блок коду, який виконується лише в тому випадку, якщо система згенерувала виняток. Для того щоби точно вказати, який тип винятку буде оброблений, його треба записати після слова except. В одній конструкції з try може бути довільна кількість послідовних операторів except. Закінчувати конструкцію з оператором try можна фінальним блоком else, який виконується, якщо блок try спрацьовує без помилок.

Перехоплення винітку:

```
>>> try:
        num = float(input("Введіть число: "))
except:
        print("Ви ввели не число.")


Введіть число: Привіт
Ви ввели не число.
```

Доречно скрізь перехоплювати винятки, де взаємодія програми з зовнішнім середовищем наводить на думку про можливі помилки (наприклад, відкриття файлів, конвертація даних) (рис. 4.5). Якщо ви вирішили перехопити можливий виняток, але не знаєте його тип, то слід створити такий виняток самостійно і подивитися назву.

Наприклад:

```
>>> 5/0
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    5/0
ZeroDivisionError: division by zero
```

Була викликана помилка типу `ZeroDivisionError`.

Таблиця 4.5

Деякі типи виніткяв

Тип виключення	Опис
IOError	Генерується, якщо неможливо виконати операцію введення/виведення, наприклад, відкрити на читання неіснуючий файл
FloatingPointError	Генерується в разі невдалого виконання операції з рухомою крапкою
OverflowError	Генерується, коли результат арифметичної операції надто великий для подання
ZeroDivisionError	Генерується, якщо в операції ділення або знаходження залишку другий аргумент – нуль
AttributeError	Генерується, якщо об'єкт не має цього атрибута (значення або методу)
BufferError	Генерується, якщо операція, пов'язана з буфером, не може бути виконана

Закінчення табл. 4.5

Тип виключення	Опис
ImportError	Генерується, якщо не вдалося імпортувати модуль або його атрибут
IndexError	Генерується, якщо у послідовності не знайдено елемент з вказаним індексом
KeyError	Генерується, якщо у словнику не знайдено вказаний ключ
MemoryError	Генерується, якщо недостатньо пам'яті
NameError	Генерується, якщо не знайдено ім'я (функції або змінної)
SyntaxError	Генерується, якщо інтерпретатор знаходить в коді синтаксичну помилку
IndentationError	Генерується, якщо інтерпретатор знаходить в коді неправильні відступи
TabError	Генерується, якщо інтерпретатор знаходить в коді змішування табуляції і пробілів
TypeError	Генерується, якщо стандартна операція або функція застосовується до об'єкта невідповідного типу
ValueError	Генерується, якщо стандартна операція або функція приймає аргумент потрібного типу, але з невідповідним значенням
UnicodeError	Генерується, якщо виникає помилка в кодуванні unicode у рядках

Конструкція `try/except/else`:

```
>>> try:
        num = int(input("Введіть число: "))
except ValueError:
        print("Ви ввели не число.")
else:
        print("Ви ввели число", num)


Введіть число: 5
Ви ввели число 5
```

5. БІБЛІОТЕКИ PYTHON ДЛЯ РОБОТИ З ГЕОПРОСТОРОВИМИ ДАНИМИ

5.1. Види бібліотек Python для роботи з геопросторовими даними

Мовою програмування Python написано ряд бібліотек для роботи з геопросторовими даними:

– бібліотеки для читання і запису геопросторових даних;

– бібліотеки для роботи з картографічними проєкціями;

– бібліотеки для аналізу і маніпулювання геопросторовими даними безпосередньо у програмах на Python;

– інструменти для візуалізації геопросторових даних.

5.1.1. Бібліотеки для читання і запису геопросторових даних

Існує дві популярні бібліотеки для читання і запису геопросторових даних – **GDAL** та **OGR**.

На жаль, з назвами цих двох бібліотек виникла певна плутанина. Geospatial Data Abstraction Library (GDAL) спочатку була бібліотекою для роботи з растровими геопросторовими даними, у той час як окрема бібліотека OGR призначалася для роботи з векторними даними. Тепер ці дві бібліотеки частково об’єднані та, як правило, завантажуються і встановлюються разом під назвою GDAL. Для того щоб уникнути плутанини, будемо називати цю комбіновану бібліотеку як GDAL/OGR і використовувати назву GDAL лише в разі звернення до бібліотеки роботи з растровими даними.

За замовчуванням GDAL підтримує читання 116 різних форматів растрових файлів і зберігання у 58 різних форматах. OGR за замовчуванням підтримує читання 56 різних форматів векторних файлів і зберігання у 30 форматах. Завдяки цьому GDAL/OGR є одним з найпотужніших перекладачів геопросторових даних і, звісно, найкориснішою з вільно доступних бібліотек для читання і зберігання геопросторових даних.

Архітектура GDAL. GDAL використовує таку модель даних для опису растрових просторових даних (рис. 5.1).

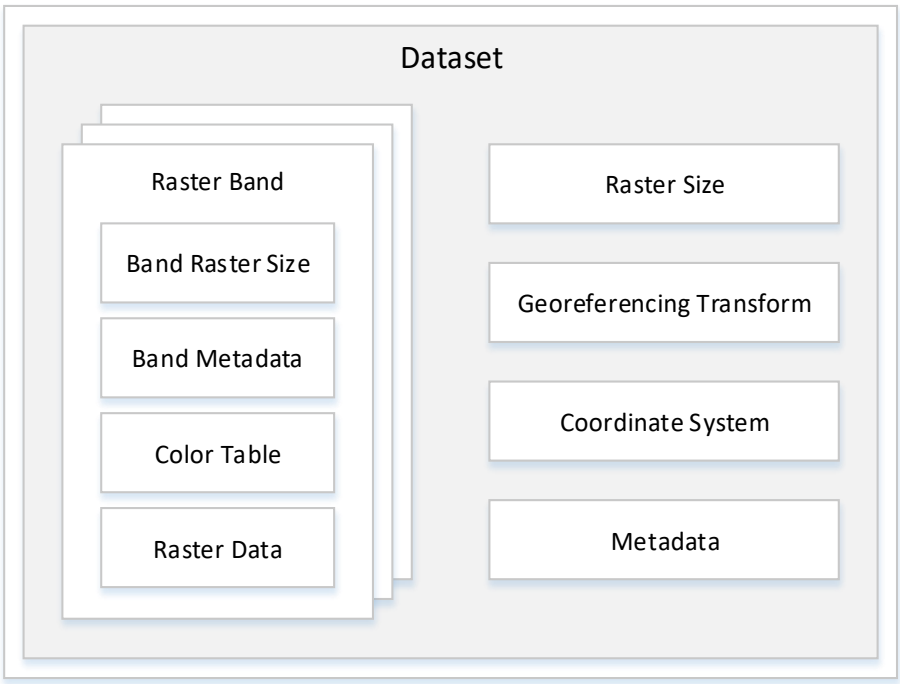

Рис. 5.1. Модель даних для опису растрових просторових даних

Розглянемо складові цієї моделі:

Dataset (набір даних) містить всі растрові дані у вигляді колекції растрових «груп», зокрема інформацію, яка є спільною для всіх цих груп. Набір даних звичайно міститься в одному файлі.

Raster band (растрова група) – це смуга, канал або шар у зображенні. Наприклад, дані зображення RGB зазвичай мають окремі канали (смуги) для червоного, зеленого і синього компонентів зображення.

Raster size (розмір растру) визначає загальну ширину і висоту зображення в пікселях.

Georeferencing transform (прив'язка) конвертує координати растра з (x, y) у просторові координати, тобто координати на поверхні Землі. GDAL підтримує два типи прив'язки: афінне перетворення (яке іноді називають лінійним перетворенням) та опорних точок місцевості:

Афінне перетворення (affine transformation) — це математична формула, яка дає змогу перетворювати растрові дані (рис. 5.2).

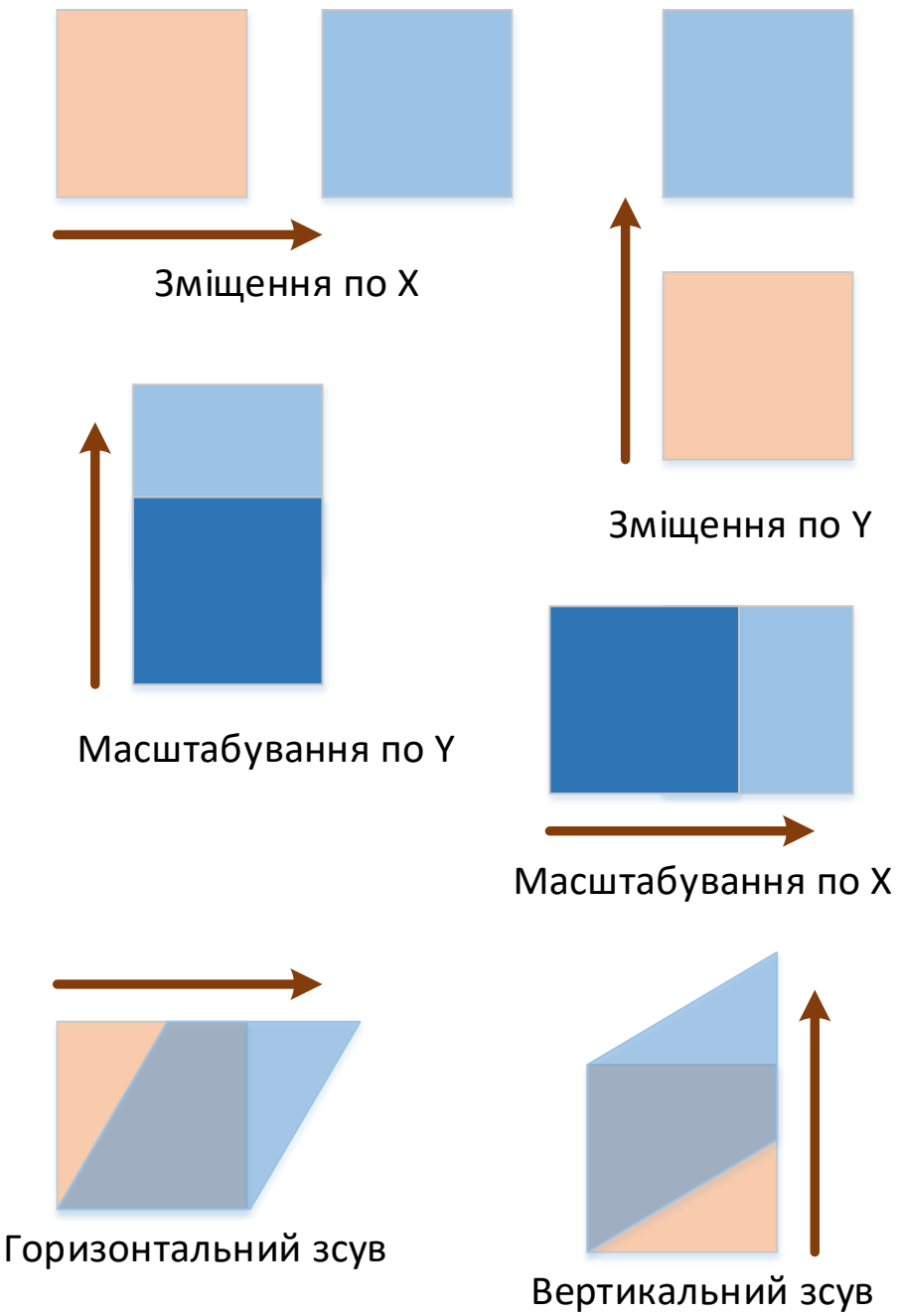

Рис. 5.2. Афінне перетворення

За один раз на застосувати більш як одну таку операцію, що дає змогу виконувати складні перетворення, такі як оберти.

Опорні точки місцевості (Ground Control Points — GCPs) зіставляють одну або декілька позицій з растра з їхніми еквівалентними координатами прив'язки, як це показано на рис. 5.3.

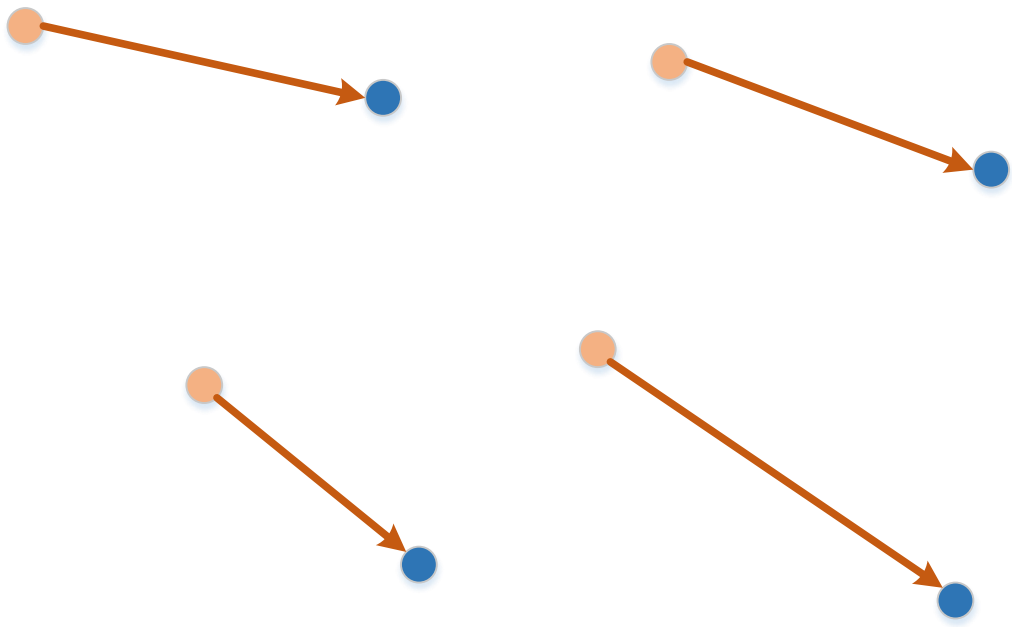

Рис. 5.3. Опорні точки місцевості

Зверніть увагу, що GDAL не переводить координати за допомогою GCP – це залишається застосунку і зазвичай охоплює складні математичні функції для виконання перетворень.

Coordinate system (система координат) описує координати прив'язки, створені трансформуванням координат. Система координат містить проєкцію і датум, а також одиниці растра (піксель) і масштаб, що використовуються растровими даними.

Metadata (метадані) містять додаткову інформацію про набір даних загаломлому.

Кожна растрова група містить таке (серед іншого):

– *Band raster size* (розмір растрової групи) — це розмір (кількість пікселів по горизонталі і ліній у висоту) даних у групі. Він може бути таким самим як і розмір растра для повного набору даних. У такому разі набір даних у повному просторовому розрізненні або дані груп можуть потребувати масштабування, щоб відповідати набору даних.

– Деяка група метаданих (*band metadata*) містить додаткову інформацію, що стосується до цієї групи.

– *Color table* (таблиця кольорів) – описує, як піксельні значення перетворюються у кольори.

– Власне растрові дані (*raster data*).

GDAL надає ряд драйверів, які дають змогу читати (та іноді записувати) різні типи растрових геопросторових даних. У разі читання

файла GDAL автоматично вибирає підходжий драйвер залежно від типу даних; для записування спочатку треба обрати драйвер, а потім вказати драйверу створити новий набір даних, який потрібно записати.

5.1.2. Приклад коду

Приклад коду GDAL. Файл цифрової моделі рельєфу (ЦМР, DEM – Digital Elevation Model) містить значення висот. У наступному прикладі програми ми використовуємо GDAL, для того щоби розрахувати середнє значення висот, що містяться у файлі ЦМР. Для цього прикладу використовуємо файл ЦМР, завантажений з ресурсу набору даних висот GLOBE (рис. 5.4).

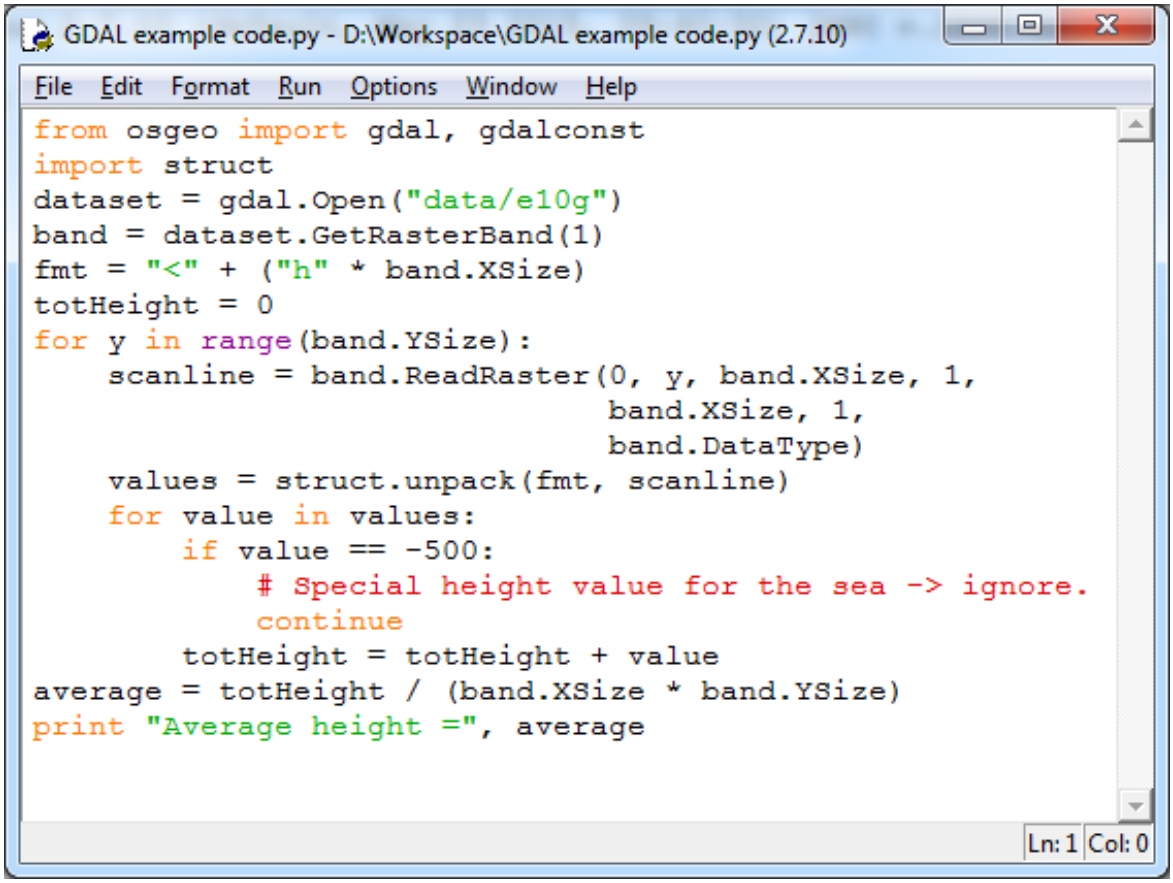

Рис. 5.4. Приклад коду для визначення цифрової моделі рельєфу

Ця програма отримує одну растрову групу з файла ЦМР, а потім зчитує її по одному растровому рядку. Далі за допомогою бібліотечного модуля Python зчитуються окремі значення висот з растрових рядів. Оскільки у наборі даних GLOBE для представлення океану використовується особливе значення висоти − 500, ці дані вилучають з

розрахунків. Дані, що залишилися, використовуються для обчислення середньої висоти у метрах для всіх даних ЦМР-файла.

Архітектура OGR. OGR використовує наведену далі модель для роботи з векторними геопросторовими даними (рис. 5.5):

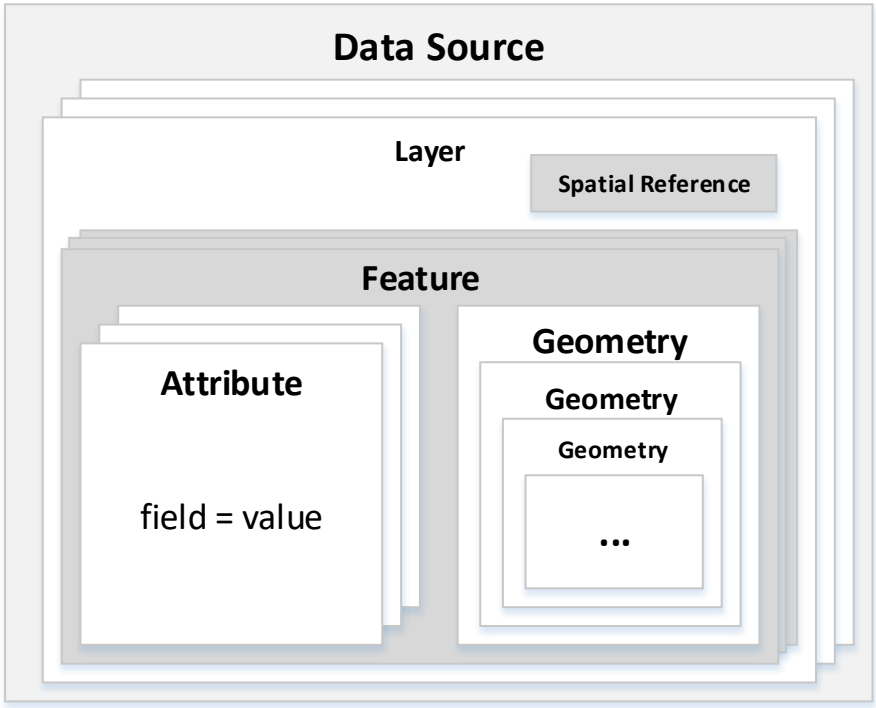

Рис. 5.5. Модель для роботи з векторними геопросторовими даними

Розглянемо цю архітектуру докладніше.

Джерело даних (**data source**) — це файл, з яким ви працюєте. Джерелом даних не обов'язково повинен бути файл, це може бути URL чи щось інше, що містить дані.

Джерело даних містить один чи більше шарів (**layers**), що містять набори зв'язаних даних. Наприклад, одиничне джерело даних, що репрезентує країну, може складатися з кількох шарів «територія» (terrain), «горизонталі» (contour lines), «дороги» (roads) та «межі міст» (city boundaries). Інші джерела даних можуть містити лише один шар. Кожен шар має просторову прив'язку і список об'єктів.

Просторова прив'язка (**spatial reference**) визначає проєкцію і початок відліку (датум), що використовуються даними шару.

Об'єкт (**feature**) є відповідним деякому важливому елементу всередині шару. Наприклад, об'єктом може бути держава, місто, дорога, острів тощо. Кожен об'єкт має список атрибутів і геометрію.

Атрибути (**attributes**) містять додаткову мета-інформацію про об'єкти. Наприклад, атрибут може задавати ім'я для об'єкта місто, кількість його населення або унікальний ідентифікатор (ID), що використовується для отримання додаткової інформації про об'єкт із зовнішньої бази даних.

Геометрія (**geometry**) описує фізичну форму або розташування об'єкта. OGR, як і GDAL, надає ряд драйверів, які дають змогу читати (та іноді записувати) різні типи векторних геопросторових даних. Під час читання файла, OGR обирає потрібний драйвер автоматично; у разі записування, слід спочатку обрати драйвер, а потім вказати йому створити нове джерело даних для запису.

Приклад коду OGR. Наведений приклад програми використовує OGR для читання вмісту шейп-файла і виведення на екран значення атрибута ІМ'Я для кожного об'єкта разом з типом геометрії.

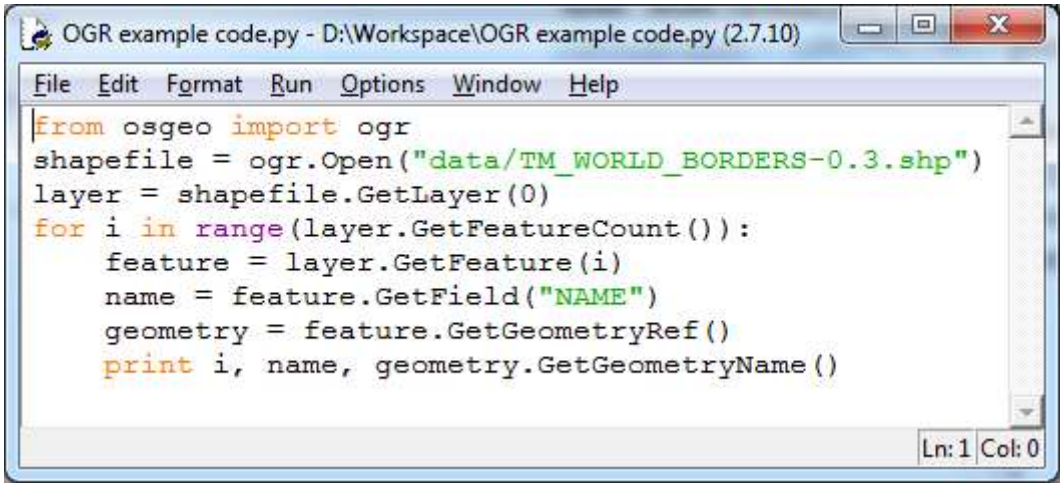

Рис. 5.6. Код для читання вмісту шейп-файлу

5.1.3. Документація

GDAL і OGR добре задокументовані, але з деякою складністю для Python-програмістів. GDAL / OGR-бібліотека і пов'язані інструменти командного рядка написані на C і C++. Реалізація бібліотек GDAL / OGR доступна декількома мовами, зокрема Python, але документація написана для C++ версії бібліотек. Через це читання документації може стати викликом – не лише через те, що описи методів виконано для C++, а й тому що реалізація бібліотек та Python зробила назви методів і класів більш «пайтонівськими».

На щастя, бібліотеки Python в основному самодокументовані завдяки рядкам документації, прописаним у файлах бібліотек Python. Це означає, що ви можете вивчити документацію, використовуючи вбудовані функції або методи Python, які можу бути запущені з командного рядка, наприклад:

```
>>> help (gdal)
Help on module osgeo.gdal in osgeo:

NAME
    osgeo.gdal

FILE
    c:\python27\lib\osgeo\gdal.py

DESCRIPTION
    # This file was automatically generated by SWIG (http://www.swig.org).
    # Version 2.0.4
    #
    # Do not make changes to this file unless you know what you are doing--modif
y
    # the SWIG interface file instead.

CLASSES
    __builtin__.object
        AsyncReader
        ColorEntry
```

Не всі методи задокументовані, тому може виникнути потреба використовувати C++ документацію на веб-сайті GDAL для отримання детальнішої інформації. Деякі рядки документації були безпосередньо скопійовані з C++ документації – але загалом документація для GDAL/OGR досить якісна і сприяє швидкому використанню цих бібліотек.

5.1.4. Доступність

GDAL/OGR доступна для більшості версій Microsoft Windows, сучасних Unix машин, зокрема Linux і Mac OS X. Основний веб-сайт GDAL знаходимо за адресою: http://www.gdal.org/.

Основний веб-сайт OGR: http://www.gdal.org/ogr

Для завантаження GDAL/OGR потрібно перейти за посиланням Downloads на основному веб-сайті GDAL. Користувачі Windows можуть скористатися набором FWTools, що містить широкий набір геопросторових застосунків для win32 машин, зокрема GDAL/OGR та

їхню реалізацію на Python. FWTools можна знайти за посиланням: http://fwtools.maptools.org.

Користувачі Mac OS X можуть отримати бібліотеки з http://www.kyngchaos.com/software/frameworks.

Зверніть увагу, що для розглядуваних прикладів потрібні бібліотеки GDAL версії 1.9 і вище.

Оскільки GDAL/OGR – вільне програмне забезпечення, їхній код доступний на веб-сайті, і його можна скомпілювати самостійно. Більшість користувачів, однак, вважають, що простіше використовувати вже скомпільовані бінарні версії бібліотек.

5.2. Бібліотеки для роботи з картографічними проєкціями

Однією з проблем під час роботи з геопросторовими даними є те, що геодезичні області (точки на поверхні Землі) відображаються в двовимірній декартовій площині за допомогою картографічної проєкції. Тому щоразу, коли у вас є геопросторові дані, ви повинні знати проекцію, яку використовують ці дані. Потрібно також знати датум (модель форми Землі), передбачений за даними.

Загальною проблемою в роботі з геопросторовими даними є те, що треба конвертувати дані з однієї проекції /датуму до інших. На щастя, є бібліотека Python pyproj, яка полегшує це завдання.

5.2.1. Pyproj

Pyproj – це «обгортка» Python навколо іншої бібліотеки під назвою PROJ.4, що є це абревіатураою для четвертої версії бібліотеки PROJ. Спочатку PROJ була написана Геологічною службою США для роботи з картографічними проекціями, і широко використовувалася у геопросторовому програмному забезпеченні протягом багатьох років. Бібліотека pyproj уможливлює доступ до функціональності PROJ.4 зсередини Python-програм.

Бібліотека pyproj складається з кількох частин (рис. 5.7).

Pyproj складається з двох класів: Proj та Geod. Proj конвертує значення з широти і довготи до координат (x,y) звичайних карт і навпаки. Geod виконує розрахунки відстані і кутів у полярних координатах (Great Circle). Обидва класи побудовані на основі бібліотеки PROJ.4. Розглянемо ці два класи докладніше.

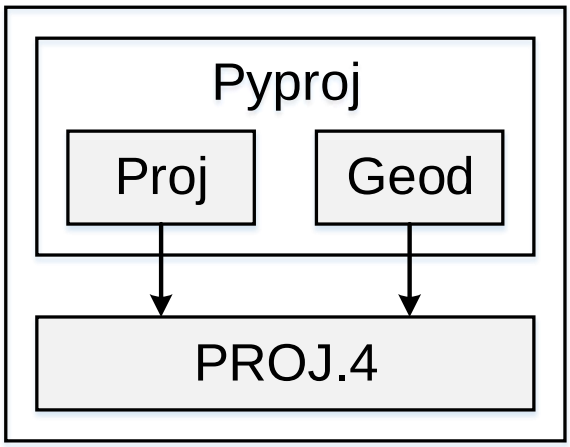

Рис. 5.7. Бібліотека pyproj

Proj – це клас картографічних перетворень, який дає змогу конвертувати географічні координати (тобто значення широти і довготи) у картографічні координати (значення x, y, за замовчуванням у метрах), і навпаки.

Створюючи новий примірник Proj, слід вказати проєкції, датум та інші значення, які використовують в описі того, як виконується проєкція. Наприклад, щоб використовувати поперечну проєкцію Меркатора та еліпсоїд WGS84, рльоібно виконате таке:

```
projection = pyproj.Proj(proj='tmerc',
ellps='WGS84')
```

Після створення екземпляра Proj, його можна використовувати для перетворення широти і довготи у координати (x, y) за допомогою цієї проєкції. Ви можете скористатися ним, щоб здійснити зворотну проєкцію, тобто перетворення з координат (x, y) у значення з широтою і довготою.

Корисна функція `transform`() може бути використана для перетворення координат безпосередньо з однієї проєкції в іншу. Для цього варто лише передати початкові координати, об'єкт Proj, який описує проєкцію початкових координат, і бажану кінцеву проєкцію. Це може бути дуже корисним у перетворенні окремих і масових координат.

Geod – це клас геодезичних обчислень, який дає змогу виконувати різні кутові розрахунки. До кутових розрахунків вдаються, наприклад,

для точного обчислення відстані між двома точками на поверхні Землі. Клас Geod, однак, надає ще більше можливостей.

Метод fwd () приймає початкову точку, азимут (кутовий напрям) і відстань та повертає кінцеву точку і зворотній азимут (зворотній кут між кінцевою і початковою точкою) (рис. 5.8).

Рис. 5.8. Приклад застосування методу `fwd ()`

Метод inv () приймає дві координати і повертає напрям і зворотний азимут, а також відстані між ними (рис. 5.9).

Рис. 5.9. Приклад застосування методу `inv ()`

За допомогою методу npts () розраховує координати заданої кількості точок, віддалених одна від одної на однакову відстань уздовж геодезичної лінії, що проходить від початкової до кінцевої точки (рис. 5.10).

Рис. 5.10. Приклад методу `npts` ()

Створюючи новий об'єкт Geod, потрібно вказати еліпсоїд, використаний для виконаннія геодезичних обчислень. Еліпсоїд може бути вибраний з ряду попередньо визначених еліпсоїдів, або можна ввести параметри для еліпсоїда (екваторіальний радіус, полярний радіус та ін.) самостійно.

5.2.2. Приклад коду

Наступний приклад починається з місця, вказаного за допомогою координат UTM zone 17. Використання двох об'єктів Proj визначає UTM Zone 17 і проєкції широта / довгота. Це транслює місцеві координати у значення широти і довготи. Потім, використовуючи об'єкт Geod, розрахуємо значення координат широти/довготи для іншої точки за 10 км на північний схід від початкової точки (рис. 5.11).

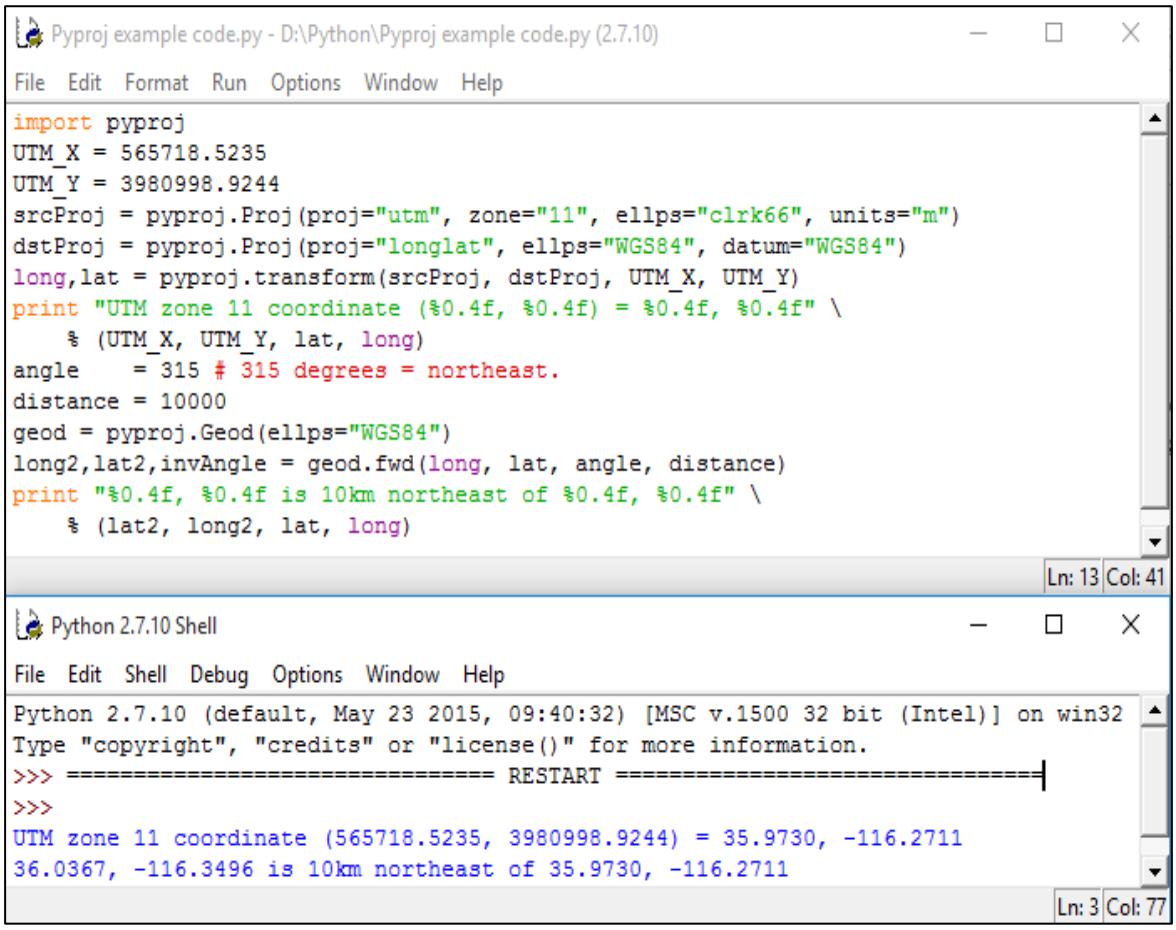

Рис. 5.11. Фрагмент коду геодезичного обчислення

5.2.3. Документація

Документація, доступна на веб-сайті pyproj і в каталозі docs, що надається разом з вихідним кодом, є щонайкращою. Вона описує, як використовувати різні класи і методи, з якою метою і які параметри для цього потрібні. Проте роль документації є досить незначною, коли доходить до параметрів, використовуваних для створення нового об'єкта Proj.

Екземпляр класу Proj ініціалізується з контрольними параметрами пар значень картографічної проєкції. Пари значень можуть бути взяті зі словника або як іменовані аргументи, або як Proj4 рядок (сумісний з командою proj).

Документація містить посилання на веб-сайт, де наведено ряд стандартних картографічних проєкцій і пов'язаних з ними параметрів, але попри розуміння того, що означають ці параметри, як правило, доцільним є заглиблитися у вивчення самої документації PROJ. Документація для PROJ всеохоплююча і заплутана, тим більше, що

основну інструкцію складено для третьої версії PROJ з доповненнями для пізніших версій. Спроба розібратися у цьому може бути справжньою проблемою.

На щастя, в більшості випадків взагалі немає потреби звертатися до документації PROJ. Працюючи з геопросторовими даними і використовуючи GDAL чи OGR, можна легко отримати проєкцію як «рядок proj4», який може бути безпосередньо переданий до ініціалізатора Proj. Якщо потрібно конкретно вказати проєкцію, можна вибрати проєкцію і еліпсоїд за допомогою параметрів proj = «...» та ellps = «...» відповідно. Якщо ви хочете зробити щось більше, ніж це, ви повинні будете звернутися до документації PROJ по більш докладну інформацію. Докладніше за посиланням: http://trac.osgeo.org/proj/.

5.2.4. Доступність

Pyproj доступна для MS Windows, з вихідним кодом, представленим для інших платформ. Адреса основної веб-сторінки pyproj: https://github.com/jswhit/pyproj.

Встановлення бібліотеки залежить від використовуваної операційної системи.

5.3. Бібліотеки для аналізу і маніпулювання геопросторовими даними

Оскільки як геопросторові дані працюють з геометричними об'єктами, такими як точки, лінії й полігони, часто доводиться виконувати різні розрахунки, використовуючи ці геометричні об'єкти. На щастя, є деякі дуже потужні інструменти для роботи саме з такими даними. З причин, вказаних далі, для роботи з цим типом обчислювальної геометрії ми користуватимемося Python-бібліотекою – Shapely.

5.3.1. Shapely

Shapely – це пакет Python для маніпулювання й аналізу двовимірних геопросторових геометричних об'єктів (рис. 5.12). Shapely оснований на бібліотеці GEOS, яка реалізує широкий спектр маніпуляцій геопросторовими даними в C++. Натомість GEOS базується на бібліотеці під назвою Java Topology Suite, яка забезпечує ту саму функціональність для Java-програмістів. Shapely забезпечує Python-

інтерфейс для GEOS, що дає змогу виконувати маніпуляції з даними безпосередньо з програм на Python.

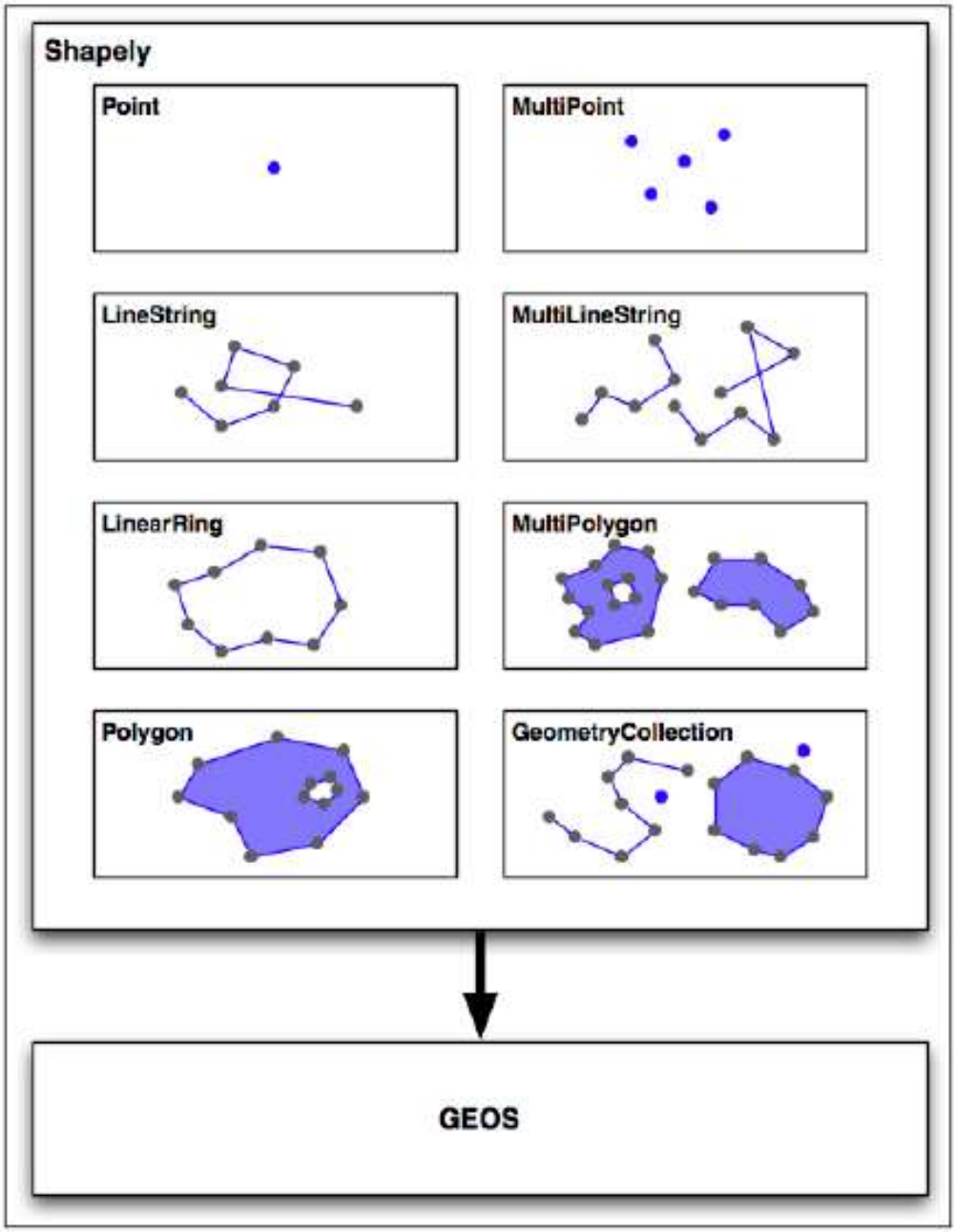

Рис. 5.12. Бібліотека Shapely

Архітектура. Вся функціональність Shapely побудована на GEOS. Дійсно, Shapely для роботи потребує встановленої GEOS.

Shapely складається з восьми основних класів, що представляють різні типи геометричних форм:

1. Клас Точка (Point) – це одинична точка у просторі. Точки можуть бути двовимірним (x, y), або тривимірним (x, y, z).

2. Клас Лінія (LineString) – це послідовність з'єднаних разом точок, що формують лінію. LineStrings може бути простою (сегменти лінії не перетинаються) або комплексною (де два відрізки LineString перехрещуються).

3. Клас Кільце (LinearRing) – це лінія, яка закінчується у початковій точці. Сегменти LinearRing не можуть перетинатися або дотикатися.

4. Клас Полігон (Polygon) – це заповнена область, можливо, з однією або більше незаповнених областей всередині.

5. Клас Множина точок (MultiPoint) – це набір точок.

6. Клас Мультилінія (MultiLineString) – це набір LineStrings.

7. Клас Мультиполігон (MultiPolygon) – це набір полігонів.

8. Клас Геометрична колекція (GeometryCollection) –це набір будь-яких комбінацій точок, ліній, кілець і полігонів.

Так само добре, як і представляти різні типи геометричних об'єктів, Shapely надає ряд методів і атрибутів для маніпуляцій та аналізу цих об'єктів. Наприклад, клас LineString містить атрибут length, еквівалентний довжині всіх відрізків, складових LineString, і метод crosses(), який повертає істину, якщо дві LineStrings перетинаються. Інші методи дають змогу обчислити перетин двох полігонів, розширення або розмивання об'єктів, спростити геометрію об'єкта, обчислити відстань між двома об'єктами, і побудувати полігон, який охоплює всі точки в межах заданого списку об'єктів (атрибут convex_hull).

Зверніть увагу, що Shapely – це скоріше бібліотека для операцій з просторовими, а не геопросторовими об'єктами. Shapely не містить відомостей про поняття про географічні координати. Натомість перед маніпуляцією вона припускає, що геопросторові дані проєктуються на двовимірну декартову площину. Відтак результати операції потім можуть бути перетворені знову у географічні координати.

5.3.2. Приклад коду

Наступна програма (рис. 5.13) створює два геометричні об'єкти Shapely, коло і квадрат, та обчислює площу їхнього перетину (рис. 5.14).

```
Shapely example code.py - D:\Python\Shapely example code.py (2.7.10)
File  Edit  Format  Run  Options  Window  Help
import shapely.geometry
pt = shapely.geometry.Point(0, 0)
circle = pt.buffer(1.0)
square = shapely.geometry.Polygon([(0, 0), (1, 0),
                                   (1, 1), (0, 1),
                                   (0, 0)])
intersect = circle.intersection(square)
for x,y in intersect.exterior.coords:
    print x,y
                                                    Ln: 1  Col: 0
```

Рис. 5.13. Приклад коду програми для визначення перетину двох геометричних об’єктів та їх площі перетину

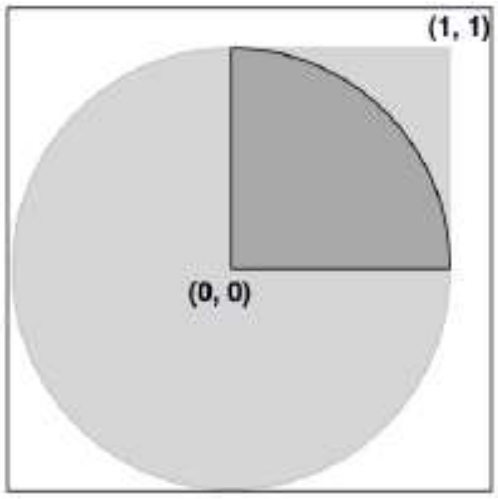

Рис. 5.14. Одиничне коло з вписаним квадратом

Перетин буде полігоном у формі чверті кола, як це показано на темно-сірій частині попереднього зображення:

Зверніть увагу, як коло будується шляхом отримання об'єкта Точка і за допомогою методу `buffer`(), що створює полігон, який являє собою контур кола.

5.3.3. Документація

Shapely постачається з якісно виконаною документацією, з докладними описами, розширеними зразками коду і великою кількістю ілюстрацій, які ясно показують, як працюють різні класи, методи й атрибути.

Документація Shapely цілком самостійна; немає потреби в документації GEOS або Java Topology Suite (якщо, звичайно, у вас немає особливого бажання побачити реалізацію у цих бібліотеках). Єдиним винятком коли, можливо, потрібно звернутися до документації GEOS, є компіляція GEOS з вихідних кодів і виникнуть проблеми в її роботі.

5.3.4. Доступність

Shapely може бути запущена на більшості операційних систем, таких як MS Windows, Mac OS X і Linux. Веб-сайт Shapely можна знайти за адресою: https://pypi.python.org/pypi/Shapely

На сайті розміщено все потрібне, зокрема документацію і файли для завантаження для бібліотеки Shapely як у вигляді початкового коду, так й у вигляді скомпільованих програм для MS Windows.

Якщо ви встановлюєте Shapely на комп'ютер з ОС Windows, скомпільовані програми вже містять вбудовану бібліотеку GEOS, інакше вам доведеться додатково встановити GEOS.

Зауваження: для роботи з навчальними прикладами потрібна Shapely версії 1.2 або новіша. Веб-сайт GEOS: http://trac.osgeo.org/geos/.

Для того щоби встановити GEOS на комп'ютери з Linux, треба завантажити початковий код з сайту GEOS і скомпілювати його самостійно або встановити відповідний RPM або APT пакет, що містить GEOS. Якщо ви працюєте у Mac OS X можна спробувати або завантажити і скомпілювати GEOS самостійно, або встановити скомпільований фреймворк GEOS, доступний на веб-сайті http://www.kyngchaos.com/software/frameworks.

Після встановлення GEOS в амнслідеобхідно завантажити, скомпілювати і встановити бібліотеку Shapely.

5.4. Візуалізація геопросторових даних

Дуже важко, якщо не неможливо, зрозуміти геопросторові дані, якщо вони не перетворені у візуальну форму, тобто поки вони не відображаються у вигляді якихось зображень. Перетворення геопросторових даних у зображення потребує відповідного інструментарію. У той час як існує кілька доступних інструментальних засобів, розглянемо один з них – Mapnik.

5.4.1. Mapnik

Mapnik – це вільний набір інструментів для побудови картографічних застосунків. Він отримує геопросторові дані з бази даних PostGIS, шейп-файлів, або будь-яких інших форматів, що підтримуються GDAL/OGR, і перетворює їх на чітко відрендерені, красиві зображення.

Є багато складних питань, пов'язаних з якісним рендерингом карт, і Mapnik виконує корисну роботу, даючи розробнику застосунків змогу контролювати процес рендеринга. Правила визначають, які об'єкти повинні з'явитися на карті, в той час як «символізатори» («symbolizers») контролюють зовнішній вигляд цих об'єктів.

Mapnik дає розробникам можливістть створювати XML-таблиці стилів, які контролюють процес створення карти. Так само, як і CSS-стилі, стилі Mapnik дають повний контроль над рендерингом геопросторових даних. Крім того, ви можете при бажанні створювати свої стилі вручну.

Mapnik написано на C ++, однак до нього включено прив'язки, які надають доступ до майже всієї функціональності Mapnik через Python. Оскільки ці прив'язки включені до основної бази коду, а не додаються сторонніми розробниками, підтримка Python вбудована прямо в Mapnik. Це роОтже, Python є чудовим інструментом для розробки застосунків на базі Mapnik.

Mapnik інтенсивно використовується проєктами OpenStreetMap (http://openstreetmap.org), Everyblock (http://everyblock.com). Оскільки результатом роботи Mapnik є лише зображення, це дає змогу легко виковистовувати Mapnik як частину веб-застосунків або додавати його безпосередньо у вікна програм для настільних комп'ютерів. Mapnik однаково добре працює на стаціонарному комп'ютері і в Інтернеті.

Архітектура. Головний об'єкт, з яким ви маєте справу, використовуючи Mapnik, – це Карта (Map). Склад об'єкта «Карта» відображено на рис. 5.15.

Створюючи об'єкт Карта, задають такі значення:

– загальна ширина і висота карти в пікселях;
– сторова прив'язка, використовувана для карти;
– колпроір фону, за вмістом карти.

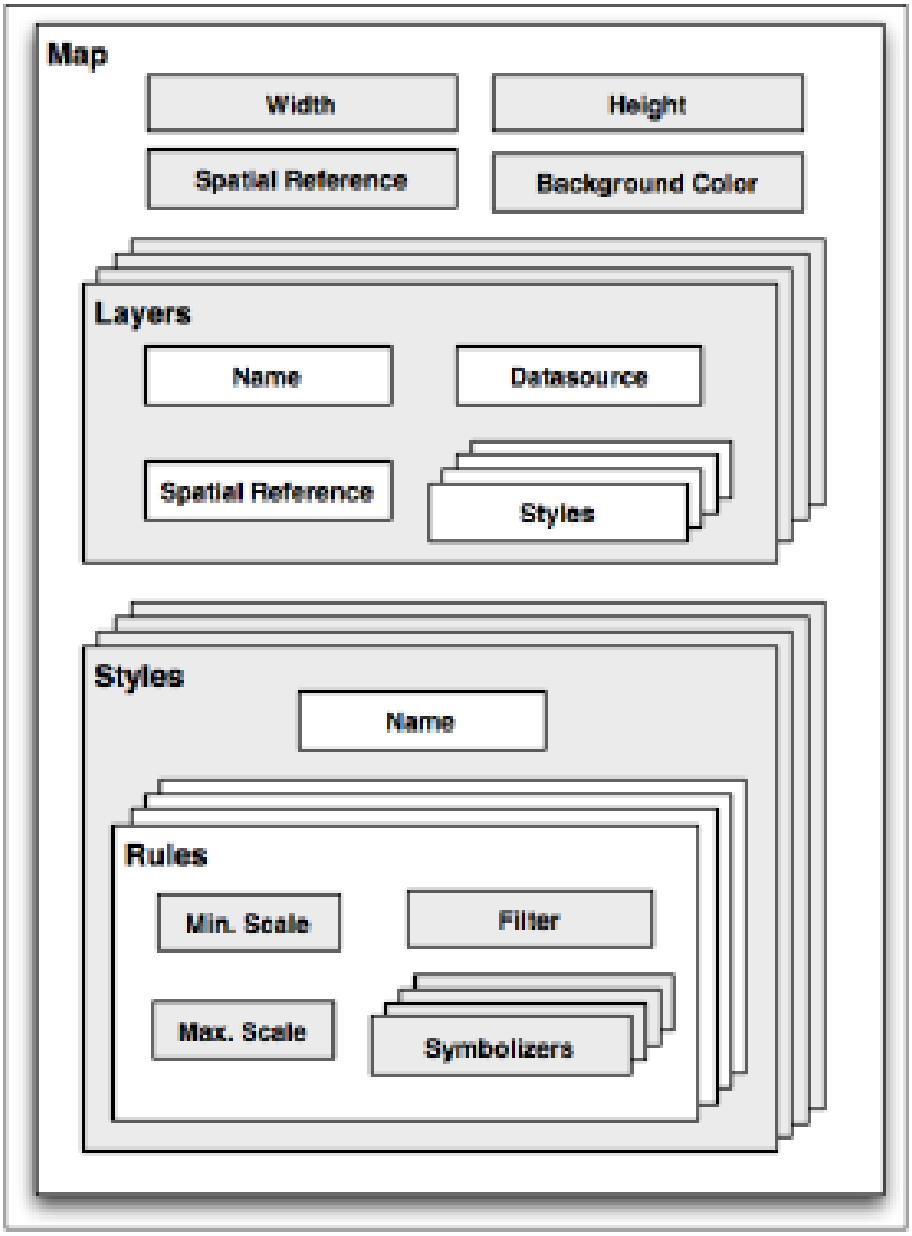

Рис. 5.15. Склад об'єкта Карта

Потім визначають один або кілька шарів, у яких є контент карт. У кожному шарі міститься така інформація:

1. Ім'я (`Name`).
2. Джерело даних (`Datasource`) об'єкта, звідки отримано дані для цього шару. Джерело даних може бути посиланням на базу даних, шейп-файл або інше джерело даних GDAL/OCR.
3. Просторова прив'язка (Spatial reference), використовувана для цього шару. Вона може відрізнятися від просторової прив'язки, що використовується для карти загалом, якщо це доцільно.

4. Список стилів (`Styles`), використовуваних у шарі. Кожен стиль називають по імені, так як насправді стилі визначаються в іншому місці (частіше за таблицею стилів XML).

Нарешті, ви обираєте один або кілька стилів (`Styles`), які визначають, як малюватиме різні шари в Mapnik. Крім імені, кожен стиль має список правил (Rules), які становлять основну частину визначення стилей. Кожне правило містить:

1. Мінімальне (`minimum scale`) і максимальне (`maximum scale`) значення масштабу (так званий «знаменник масштабу»). Правило буде застосоване лише за умови, що масштаб карти знаходиться в межах цього діапазону.

2. Фільтр (`filter`). Правило буде застосовуватися тільки до тих об'єктів, які відповідні цьому фільтру.

3. Список Символізаторів (`Symbolizers`). Вони визначають, як потрібні об'єкти будуть зображені за карті.

Є цілий ряд різних типів Символізаторів, реалізованих у Mapnik:

1) `LineSymbolizer` використовується, щоб зобразити «штрих» вздовж лінії, лінійного кільця, або іззовнішнього боку навколо полігону.

2) `LinePatternSymbolizer` використовує вміст файла зображення (вказаного по імені), щоб намалювати «штрих» вздовж лінії, лінійного кільця, або навколо іззовнішнього боку полігону.

3) `PolygonSymbolizer` використовується для зображення внутрішньої частини полігону.

4) `Po1ygonPatternSymbolizer` використовує вміст файла зображення (знову вказаного по імені), щоб намалювати внутрішню частину полігону.

5) PointSymbolizer використовує вміст файлу зображення (названого на і'мя), щоб намалювати зображення в точці.

6) `TextSymbolizer` малює текст об'єкта. Зображений текст береться з одного з атрибутів об'єкта, і є безліч варіантів контролю того, як текст буде зображено.

7) `RasterSymbolizer` використовується для малювання растрових даних, взятих з будь-якого набору даних GDAL.

8) `ShieldSymbolizer` малює одночасно текстову мітку і точку. Це схоже на використанні PointSymbolizer для малювання зображення і

TextSymbolizer для малювання мітки, крім того, він гарантує, що текст і зображення будуть поряд.

9) `BuildingSymbolizer` використовує псевдо-3D для малювання полігону, щоб створити враження, що полігон – це тривимірна будівля.

10) `MarkersSymbolizer` малює сині стрілки напрямку або SVG маркери напрямків полігонів і лінійних геометрій.

Коли ви ілюструєте прикладами `Symbolizer` і додаєте його в стилі (або безпосередньо в коді, або через таблицю стилів XML), ви передаєте ряд параметрів, які визначають, як повинен працювати Symbolizer. Наприклад, використовуючи `PolygonSymbolizer`, ви можете вказати колір заливки, прозорість, і значення «гамма», що дає змогу намалювати сусідні полігони тим самим кольором без зображення межі:

```
    p = mapnik.PolygonSymbolizer(mapnik.Color(127,
127, 0))
    p.fill_opacity = 0.8
    p.gamma = 0.65
```

Якщо Правило (`Rule`), яке використовує цей Symbolizer, відповідне одному або більше полігонам, ці полігони будуть намальовані за допомогою отриманого кольору, прозорості і значення «гамма».

Різні правила можуть, звичайно, мати різні `Symbolizers`, а також різні фільтри. Наприклад, можна встановити правила, за допомогою яких зобразити країни різними кольорами залежно від чисельності їхнього населення.

5.4.2. Приклад коду

У наступному прикладі програма виводить просту карту, використовуючи `Mapnik` (рис. 5.16).

```
Mapnik example code.py - D:\Python\Mapnik example code.py (2.7.10)
File  Edit  Format  Run  Options  Window  Help
import mapnik
symbolizer = mapnik.PolygonSymbolizer(
                        mapnik.Color("darkgreen"))
rule = mapnik.Rule()
rule.symbols.append(symbolizer)
style = mapnik.Style()
style.rules.append(rule)
layer = mapnik.Layer("mapLayer")
layer.datasource = mapnik.Shapefile(file="data/TM_WORLD_BORDERS-0.3.shp")
layer.styles.append("mapStyle")
map = mapnik.Map(800, 400)
map.background = mapnik.Color("steelblue")
map.append_style("mapStyle", style)
map.layers.append(layer)
map.zoom_all()
mapnik.render_to_file(map, "map.png", "png")
                                                          Ln: 17  Col: 0
```

Рис. 5.16. Код програми для виведення простої карти

Якщо ви працюєте в Mapnik версії 2.0, треба замінити вираз *import mapnik* у першому рядку цієї програми на *import mapnik2 as mapnik*.

Зверніть увагу, що ця програма створює `PolygonSymbolizer` для відображення полігонів країн, а потім прикріплює `symbolizer` до об'єкта Mapnik Rule. Правило (`Rule`) стає частиною об'єкта Mapnik Style. Потім ми створюємо об'єкт Mapnik Layer, зчитуючи картографічні дані з шейп-файлу, взятого з джерела даних. Нарешті об'єкт Mapnik Map створено, шар прикріплено, і підсумкова карта рендериться у файл зображення формату PNG (рис. 5.17).

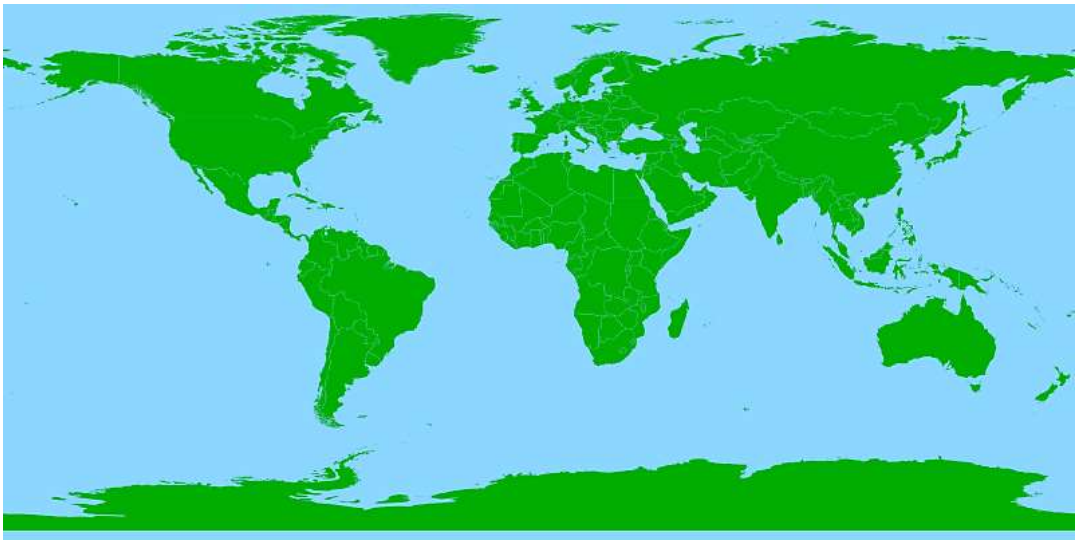

Рис. 5.17. Підсумкова карта

5.4.3. Документація

Mapnik має прийнятну документацію для проєкту з відкритим вихідним кодом: є зрозумілі інструкції щодо встановлення та деякі відмінні інструкції, хоча API-документація часто заплутана. Python-документація є похідною від документації для C++ і концентрується на описі реалізації прив'язок Python, а не на тому як кінцевий користувач буде працювати з Mapnik за допомогою Python – є багато технічних деталей, які не доступні Python-програмісту, і багатьох конкретних описів для Python немає.

Найкращий спосіб почати роботу з Mapnik – дотримуватися інструкціям з встановлення, а потім працювати за спеціальними інструкціями для Python. Можна також скористатися сторінкою *Learning Mapnik* у Mapnik Wiki: https://github.com/mapnik/mapnik/wiki/LearningMapnik.

Також корисно розглянути на Python API-документацію, незважаючи на її обмеженість. На головній сторінці розміщено перелік доступних класів і корисних функцій, багато з яких задокументовано. Класи містять інформацію про методи і властивості (атрибути), до яких можна отримати доступ, і, хоча багато з них не містять Python-документації, можна спробувати вгадати їхнє призначення.

5.4.4. Доступність

Mapnik працює на всіх основних операційних системах, зокрема MS Windows, Mac OS X та Linux. Основний сайт Mapnik має адресу http://mapnik.org.

За посиланнями можна завантажити вихідний код Mapnik, який може бути легко скомпільований, якщо ви працюєте на комп'ютері Unix, також можна скачати готові бінарні файли для Windows чи Mac OS X.

6. ДЖЕРЕЛА ГЕОПРОСТОРОВИХ ДАНИХ

Дані, які використовують у створенні геопросторових застосунків, не менш важливі, ніж код, написаний мовою програмування Python. Висока якість геопросторових даних, зокрема базових карт, ортофотокарт і ортофотопланів – це основа програмних продуктів. Якщо карти створенз неякісно, похідний від них програмний продукт (або застосунок) розглядатиметься як робота дилетанта, незалежно від того, наскільки добре написано решту коду програмного продукту.

Традиційно геопросторові дані розглядають як цінний ресурс, який продається на комерційній основі зі строгими ліцензійними обмеженнями. Проте одночасно трендом стає використання відкритих даних, так званих доступних геопросторових даних, які можна використовувати безоплатно і без обмежень. Ми використовуватимемо відкриті дані, доступні з відповідних серверів, які придатні для використання під час геопросторових розробок мовою програмування Python.

В Україні, як й у світі загалом, розробляється чимало геопорталів з відкритими даними. Геопортал – це комплекс програмно-технічних засобів, мережевих сервісів та сервісів геопросторових даних, що забезпечують відображення в мережі Інтернет геопросторових даних та метаданих, а також доступ користувачів до таких даних.

Постачальники географічної інформації використовують геопортали для публікації метаданих географічної інформації. Споживачі географічної інформації використовують геопортали для пошуку та доступу до потрібної інформації. Таким чином, геопортали відіграють все більш важливу роль в обміні географічною інформацією та дають змогу уникнути дублювання зусиль, невідповідностей, затримок, плутанини та витрачених ресурсів. Нижче подано опис деяких з них.

6.1. Джерела геопросторових даних у векторному форматі

6.1.1. OpenStreetMap

OpenStreetMap (http://openstreetmap.org) – це геопортал, в якому користувачі можуть співпрацювати з метою створення і редагування геопросторових даних. Його описано як «мапа світу, створена такими

самими людьми, як і ви, для вільного використання під відкритою ліцензією». На рис. 6.1 зображено частину карти м. Київ, а саме частину території Київського національного університету будівництва і архітектури (КНУБА), отриману з ресурсу OpenStreetMap.

OpenStreetMap не використовує для зберігання своїх даних стандартних форматів, таких як shapefile. Цьому ресурсу було розроблено власний формат на основі XML для подання геопросторових даних у вигляді вузлів (окремі точки), ребер (послідовності точок, що утворюють лінію), областей (замкнені ребра, що представляють полігони) та відношень (колекції інших елементів). Кожен елемент (вузол, ребро або відношення) може мати ряд пов'язаних з ним тегів, які несуть додаткову інформацію про нього.

Рис. 6.1. Фрагмент карти м. Київ – територія КНУБА з ресурсу OpenStreetMap

Приклад того, який вигляд мають дані у форматі OpenStreetMap XML:

```
<osm version=«0.6» generator=«CGImap 0.4.0 (21490 thorn-
02.openstreetmap.org)» copyright=«OpenStreetMap and
contributors»
attribution=«http://www.openstreetmap.org/copyright»
license=«http://opendatacommons.org/licenses/odbl/1-0/»>
   <way id=«99060145» visible=«true» version=«5»
   changeset=«26122461» timestamp=«2014-10-16T16:06:08Z»
   user=«Maturi0n» uid=«2052948»>
      <nd ref=«1145953745»/>
      <nd ref=«1145953599»/>
      <nd ref=«1145953865»/>
      <nd ref=«1145953843»/>
      <nd ref=«1145953780»/>
      <nd ref=«1145953786»/>
      <nd ref=«1145953618»/>
      <nd ref=«1145953956»/>
      <nd ref=«1145953851»/>
      <nd ref=«1145953838»/>
      <nd ref=«1145953748»/>
      <nd ref=«1145953691»/>
      <nd ref=«1145953924»/>
      <nd ref=«1145953947»/>
      <nd ref=«1145953841»/>
      <nd ref=«1145953829»/>
      <nd ref=«1145953736»/>
      <nd ref=«1145953679»/>
      <nd ref=«1145953917»/>
      <nd ref=«1145953868»/>
      <nd ref=«1145953794»/>
      <nd ref=«1145953797»/>
      <nd ref=«1145953683»/>
      <nd ref=«1145953666»/>
      <nd ref=«1145953873»/>
      <nd ref=«1145953858»/>
      <nd ref=«1145953788»/>
      <nd ref=«3006860700»/>
      <nd ref=«3006860699»/>
      <nd ref=«1145953751»/>
      <nd ref=«1145953745»/>
      <tag k=«addr:housenumber» v=«31»/>
      <tag k=«addr:street» v=«Повітрофлотський проспект»/>
```

```
    <tag k=«amenity» v=«university»/>
    <tag k=«building» v=«yes»/>
    <tag k=«name» v=«Київський національний університет
    будівництва і архітектури»/>
    <tag k=«name:de» v=«Kiewer Nationale Universität für
    Bauwesen und Architektur»/>
    <tag k=«name:en» v=«KNUCA»/>
    <tag k=«name:ru» v=«Киевский национальный университет
    строительства и архитектуры»/>
    <tag k=«name:uk» v=«Київський національний університет
    будівництва і архітектури»/>
    <tag k=«old_name:en» v=«Kyiv Civil Engineering
    Institute»/>
    <tag k=«short_name» v=«КНУБА»/>
    <tag k=«short_name:en» v=«KNUCA»/>
    <tag k=«website» v=«www.knuba.edu.ua»/>
  </way>
</osm>
```

Геопросторові дані з OpenStreetMap можна отримати одним з трьох способів:

1. Використовуючи OpenStreetMap API, завантажити потрібну підмножину даних.

2. Завантажити всю базу даних OpenStreetMap під назвою Planet.osm (близько 45 Гб) та обробляти її локально.

3. Скористатися одним з дзеркальних сайтів OpenStreetMap, які упаковують дані у менші блоки і конвертують в інші формати даних. Наприклад, дані щодо деяких українських міст можна отримати на сайті https://www.interline.io/osm/extracts/.

6.1.2. Natural Earth

Natural Earth (http://www.naturalearthdata.com) – це геопортал, який надає в загальне користування векторні і растрові картографічні дані високого, середнього та низького просторового розрізнення. Ідеться про два типи векторних картографічних даних:

– дані про забудовані території: полігони для країни, штату чи провінції, міської місцевості, меж парків, а також точкові та лінійні дані для населених територій, доріг і залізниць (рис. 6.2);

– дані про природні об’єкти: полігони і прямі лінії для континентів і великих островів (`land masses`), берегових ліній (`coastlines`), океанів, невеликих островів (`minor islands`), рифів (`reefs`), річок, озер тощо (рис. 6.3).

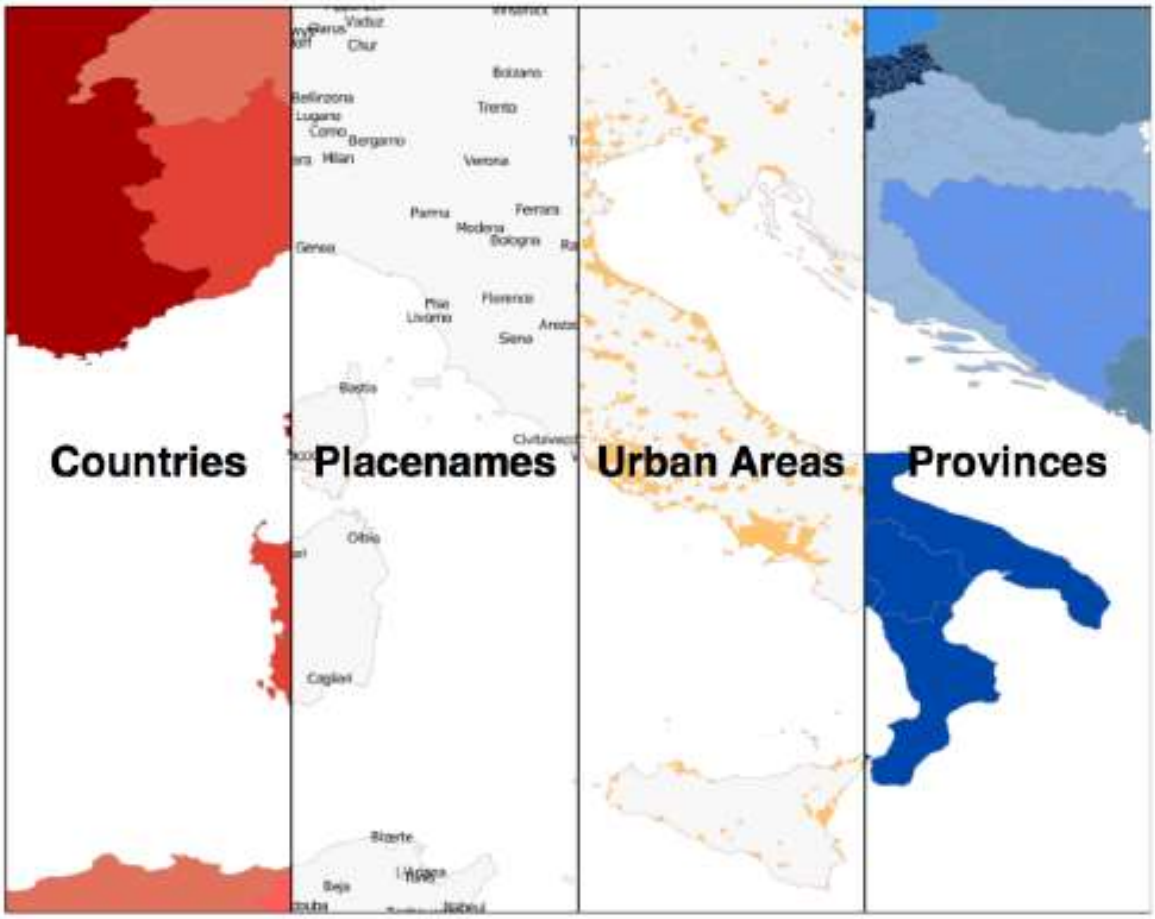

Рис. 6.2. Дані про забудовані території Natural Earth

Усе це можна завантажувати і вільно використовувати в ГІС, тому сайт Natural Earth є відмінним джерелом даних для розроблення застосунків.

Усі дані у векторному форматі на сайті Natural Earth постачаються у вигляді шейп-файлів. Дані подано у географічних координатах (широта, довгота), за допомогою стандартного датуму WGS 84, що значно полегщує використання файлів у застосунках.

Сайт Natural Earth дає змогу легко завантажити потрібні файли: слід лише натиснути на посилання *«Get the Data»* на головній сторінці. Можна також обрати просторове розрізнення і тип даних, які шукаєте. Крім того, можна завантажувати один або кілька шейп-файлів у

комплекті. Після того як вони будуть завантажені, можна використовувати спеціальні бібліотеки Python для роботи з вмістом цих шейп-файлів.

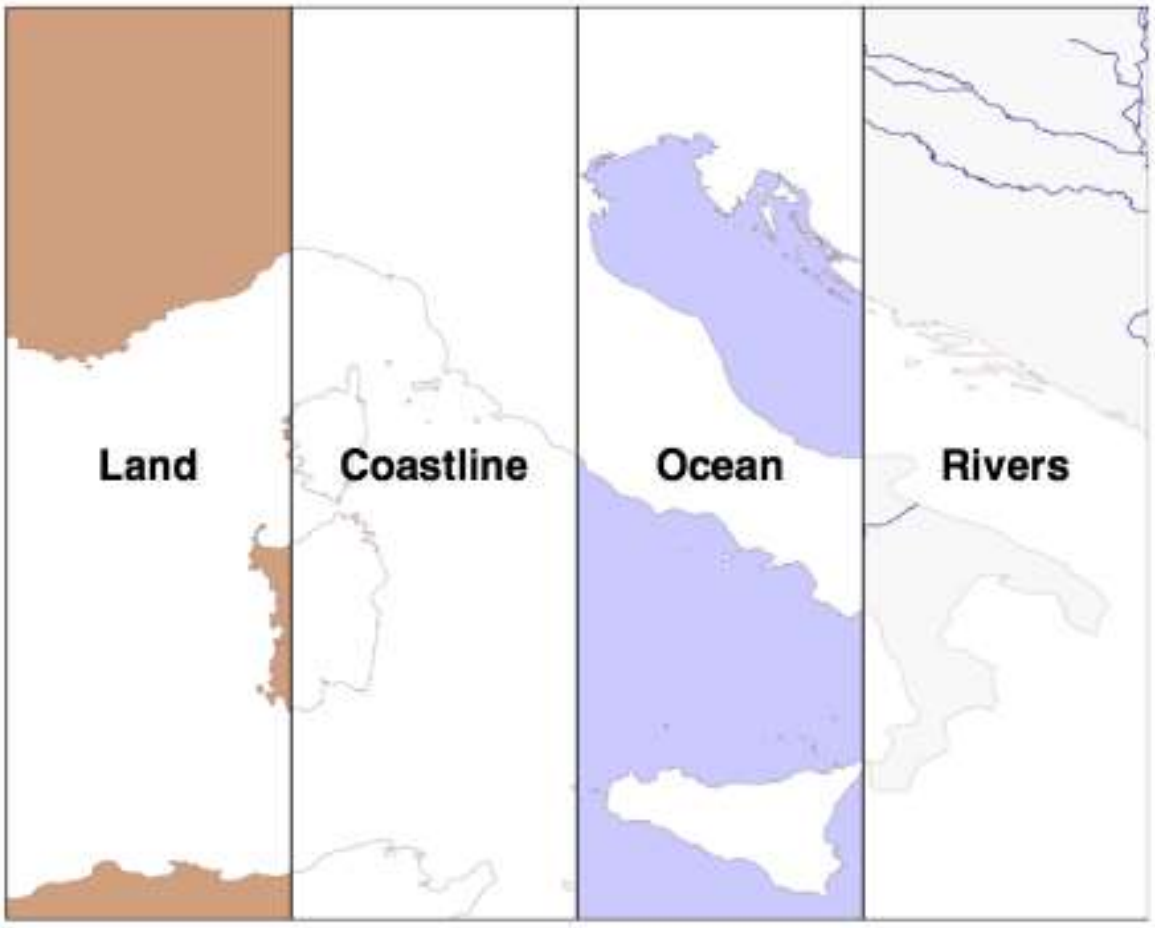

Рис. 6.3. Дані про природні об’єкти Natural Earth

Сайт Natural Earth є досить універсальним, оскільки містить докладну інформацію про геопросторові дані, які можна завантажити, і форум, на якому можна поставити запитання й обговорити будь-які проблеми.

6.1.3. Глобальна, самоузгоджена, ієрархічна, база даних берегової лінії з високим просторовим розрізненням (GSHHS)

Національний центр геофізичних даних США (частина NOAA) працює над проєктом зі створення векторних даних берегової лінії високої якості для всього світу. Результаційна база даних, яка називається Глобальною, самоузгодженою, ієрархічною базою даних берегової лінії з високим просторовим розрізненням (Global self-

consistent, hierarchical, high-resolution shoreline database – GSHHS), містить детальні векторні дані для берегових ліній, озер, і річок у п'яти різних просторових розрізненнях. Дані поділено на чотири різні «рівні»: межі океанів (ocean boundaries), межі озер (lake boundaries), межі островів в озерах (island-in-lake boundaries), межі ставків на островах в озерах (pond-on-island-in-lake boundaries).

Поданий скріншот показує берегову лінію України, озера півдня, і острови взяті з бази даних GSHHS (рис. 6.4).

Рис. 6.4. Берегова лінія України, озера півдня й острови,
взяті з бази даних GSHHS

GSHHS була побудована на основі двох відкритих баз геопросторових даних: Світового банку даних II (World Data Bank II), що містить дані про берегової лінії, озера і річки, та Світового векторного узбережжя (World Vector Shoreline), яка надає дані про берегову лінію. Оскільки база даних World Vector Shoreline містить точніші дані, але в ній немає інформації про річки і озера, дві бази даних були об'єднані з метою забезпечення користувачів по-можливості якнайточнішою інформацією. Після злиття баз даних автори вручну відредагували дані, щоб зробити їх послідовними й усунути ряд помилок. Результатом стала висока якість бази даних про межі між сушею і гідрографією для всього світу.

База даних GSHHS є доступною у двох різних форматах:

– формат бінарних даних, для роботи з Generic Mapping Tools (https://github.com/GenericMappingTools/gmt/issues);

– серія шейп-файлів.

GMT – це набір інструментів для роботи з геопросторовими даними. Ми не розглядатимемо ці інструменти, оскільки вони не реалізовані на Python.

Якщо завантажити дані у форматі шейп-файлів, отримаємо дванадцять окремих шейп-файлів, кожен з яких буде містити інформацію певного рівня і просторового розрізнення.

Залежно від просторового розрізнення мапам властива певна деталізація (табл. 6.1).

Таблиця 6.1

Деталізація карт залежно від просторового розрізнення

Код просторового розрізнення	Просторове розрізнення	Містить
c	грубе	Об'єкт, більший від 500 км^2
l	низьке	Об'єкт, більший від 100 км^2
i	середнє	Об'єкт, більший від 20 км^2
h	високе	Об'єкт, більший від 1 км^2
f	повне	Кожний об'єкт

Назва шейп-файла вказує на просторове розрізнення і рівень даних, що містяться в ньому. Рівень визначає тип меж, які включені до шейп-файлу. Наприклад, шейп-файл для меж океану в повному просторовому розрізненні називається GSHHS_f_L1.shp.

Кожний шейп-файл складається з одного шару, що містить різні полігональні об'єкти, які утворюють певний вид межі (табл. 6.2).

Таблиця 6.2

Типи меж залежно від рівня даних

Код рівня	Містить
1	Межі океану
2	Межі озера
3	Межі островів в озерах
4	Межі ставків на островах в озерах

Головний GSHHS веб-сайт можна знайти за посиланням: http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html

Файли доступні в обох форматах: GMT і шейп-файла. Після завантаження даних можна використовувати OGR, щоб прочитати файли й отримати з нього дані.

6.1.4. World Borders Dataset

Джерела описаних даних є досить комплексними. Якщо для роботи потрібні дані, які містяться в одному шейп-файлі і охоплюють весь світ, то таким набором даних може бути World Borders Dataset. У той час як деякі кордони країн змінюються, завдяки простоті World Borders Dataset є привабливим вибором для багатьох простих геопросторових застосунків.

Такий вигляд має карта «ідеального» світу, яку створено за допомогою World Borders Dataset (рис. 6.5).

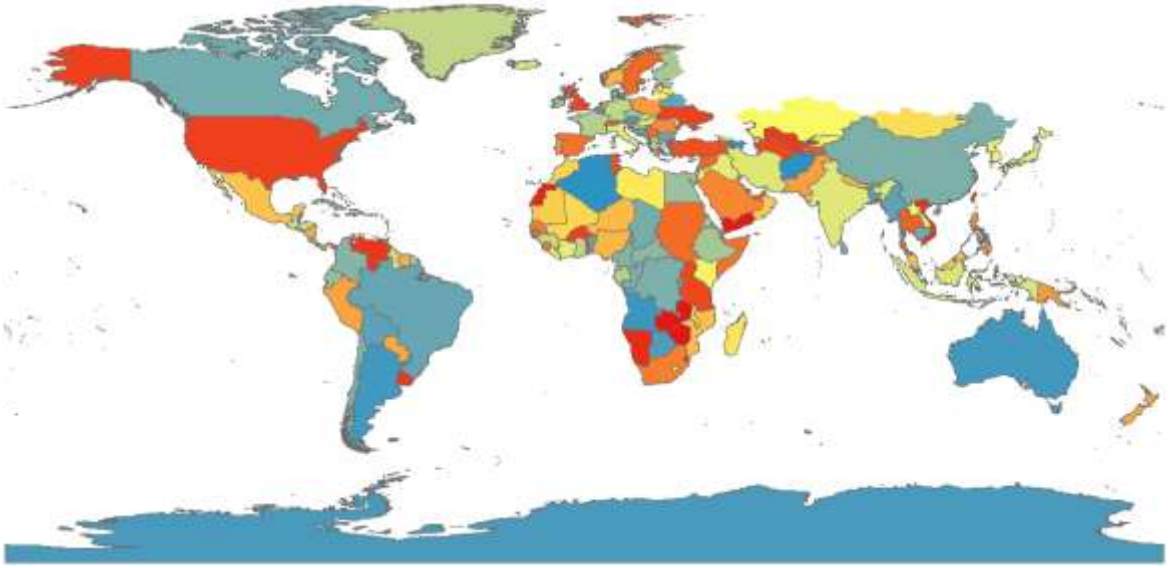

Рис. 6.5. Карта «ідеального» світу

World Borders Dataset доступний у форматі шейп-файла з одним шаром й одним об'єктом для кожної країни, який подано одним чи кількома полігонами, які визначають кордони країн разом з атрибутами, що містять назву країни чи області, різні ISO, FIRS та UN коди ідентифікації країн, класифікацію регіону і субрегіону, населення країни, площу території і широту/довготу.

Різні коди дають змогу легко добирати об’єкти за потрібними даними про країни. Також можна використовувати таку інформацію, як населення і площа, щоб виділити різні країни на карті. Наприклад, карта на попередньому рисунку використовує поле «FIPS_CNTRY», щоб розфарбувати країни різними кольорами.

World Borders Dataset можна завантажити за посиланням: http://thematicmapping.org/downloads/world_borders.php

Цуй веб-сайт також містить детальну інформацію про зміст набору даних, зокрема посилання на веб-сайт ООН, де зазначено коди регіонів та субрегіонів.

6.1.5. Пілотна версія національного геопорталу Національної інфраструктури геопросторових даних (НІГД) України

Геопортал прототипу НІГД створено у 2018 році як результат виконання українсько-японського проєкту «Створення національної інфраструктури геопросторових даних в Україні» (рис. 6.6).

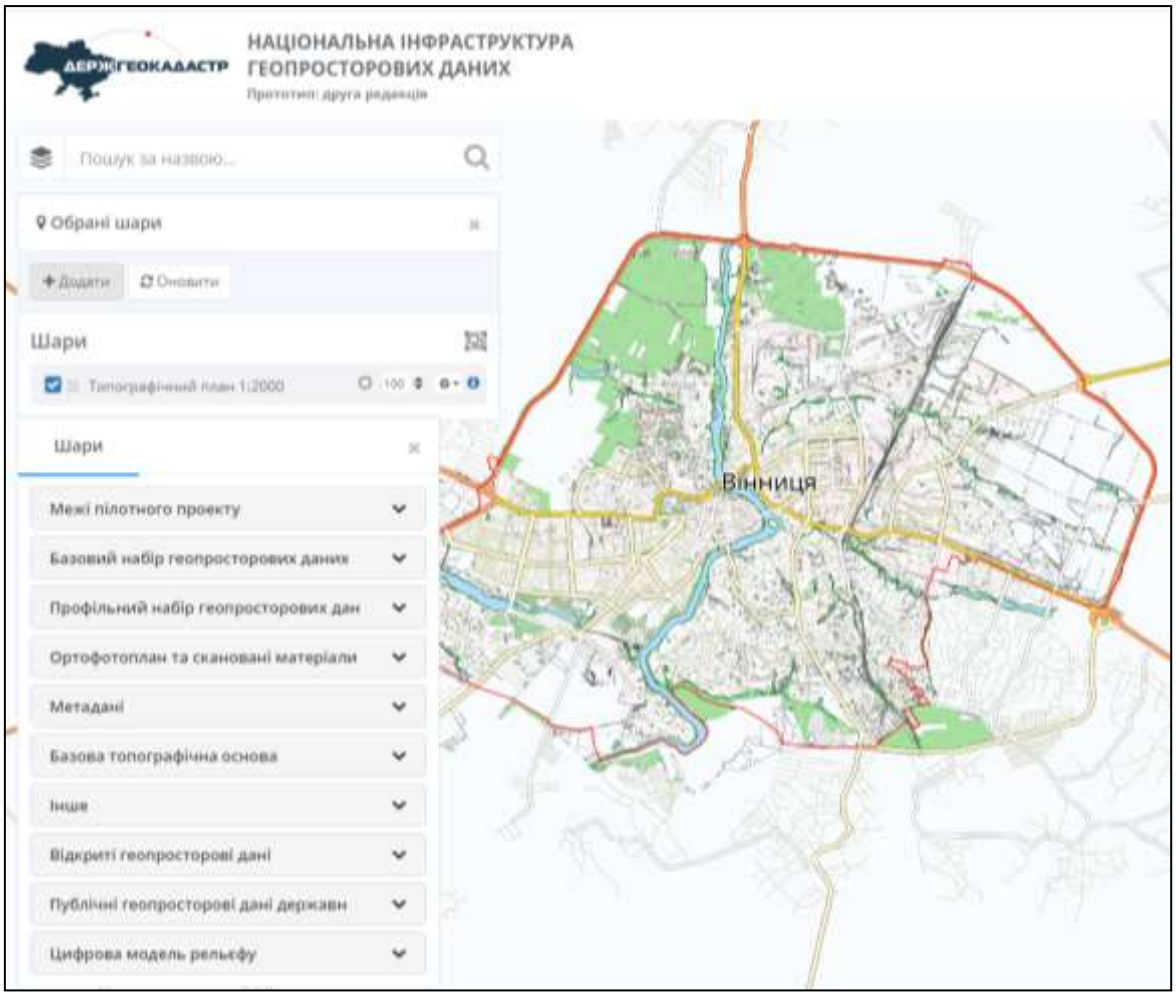

Рис. 6.6. Вікно карти геопорталу прототипу НІГД (https://nsdi2018.land.gov.ua/)

Цей пілотний проєкт зі створення національної інфраструктури геопросторових даних (НІГД) було реалізовано на території м. Вінниця площею приблизно 12 км^2. На експериментальній території виконано аерофотозйомку пілотної території, визначено базовий набір геопросторових даних, зібрано та систематизовано геопросторові дані

про пілотну територію, створено на їхній основі еталонну базу геопросторових даних відповідно до серії міжнародних стандартів ISO 19100 «Географічна інформація/Геоматика» та специфікацій INSPIRE.

Геопортал НІГД України відповідно до концепції розвитку земельного кадастру – Кадастр 2.0. Цей геопортал створено як логічне продовження українсько-японського проєкту для розвитку національної інфраструктури геопросторових даних в Україні. Архітектура НІГД зумовлює створення і розвиток мережі геопорталів на національному, регіональному та місцевому рівнях на основі серії міжнародних та гармонізованих національних стандартів ISO 19100 «Географічна інформація/Геоматика», OGC стандартів Відкритого геопросторового консорціуму та принципів INSPIRE для забезпечення інтероперабельності географічної інформації.

Геопортал НІГД України доступний за посиланням: http://nsdi.land.gov.ua/ після попередньої реєстрації (рис. 6.7).

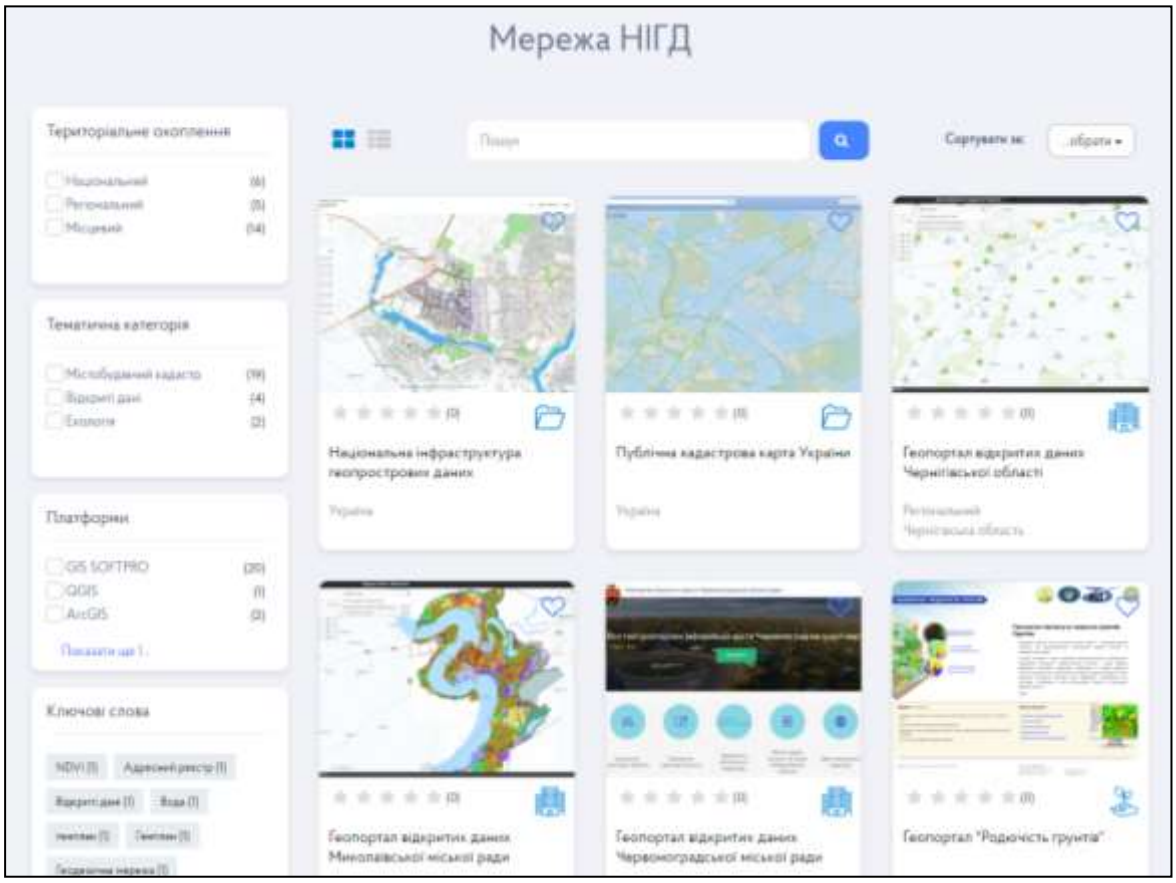

Рис. 6.7. Мережа геопорталів НІГД

Геопортал НІГД України містить такі розділи:

1. Нормативні документи.
2. Стандарти НІГД.
3. Глосарій НІГД.
4. Адресний реєстр.
5. Сервіси: валідації метаданих; перегляду наборів даних; доступу, редагування метаданих; перетворень.
6. Набори даних: інтегрована геопросторова інформаційна база; директива INSPIRE, Національна інфраструктура геопросторових даних.
7. Довідка: робота з сервісами, перегляд просторових даних, як користуватися геопорталом.
8. Реєстри: реєстр сертифікованих інженерів-геодезистів, мережа НІГД, адміністративно-територіальний устрій, адресний реєстр.
9. Каталог метаданих.
10. Карта.
11. Аналітика.

Загальний вигляд карти геопорталу НІГД подано на рис. 6.8.

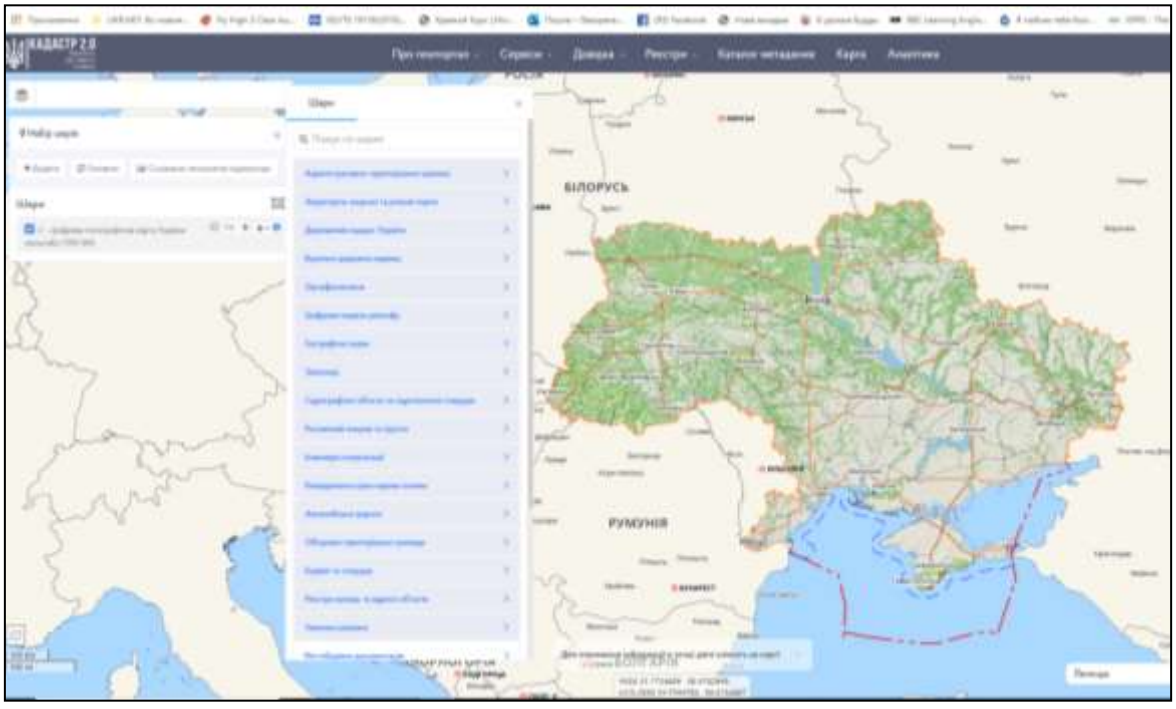

Рис. 6.8. Карта геопорталу НІГД

Створено пілот НІГД у вигляді геопорталу, який містить базові геопросторові дані, тематичні геопросторові дані, інформацію про адміністративно-територіальний устрій, Державну геодезичну мережу,

дані Державного земельного кадастру, сервіс каталогу метаданих, ортофотоплани та нормативно-правове забезпечення. З роботою геопорталу прототипу НІГД можна ознайомитися за посиланням: https://nsdi.gov.ua/.

На геопорталі доступним є сервіс каталогу метаданих, потрібний для виявлення геопросторових даних та геоінформаційних сервісів. Метадані містять упорядковані формалізовані набори спеціальних даних («даних про дані»), в яких описано структуру та властивості елементів географічної інформації, що зберігається і пропонується в цифровому і нецифровому вигляді. Метадані призначені для ведення каталогів геоінформаційних ресурсів та забезпечення процесів автоматизованого пошуку й оцінювання придатності геопросторових даних потенційними користувачами і системами. Наявність метаданих є обов'язковою умовою створення ринку геопросторових даних та сталого функціонування інфраструктури геопросторових даних. Організація формування, зберігання і доступу до метаданих є державним завданням. Ведення баз та каталогів метаданих, їхні розміщення в глобальних інформаційних мережах здійснюється уповноваженими центрами формування базових наборів геопросторових даних відповідно на загальнодержавному, регіональному та місцевому рівнях. Приклад метаданих, доступних на геопорталі, наведено на рис. 6.9.

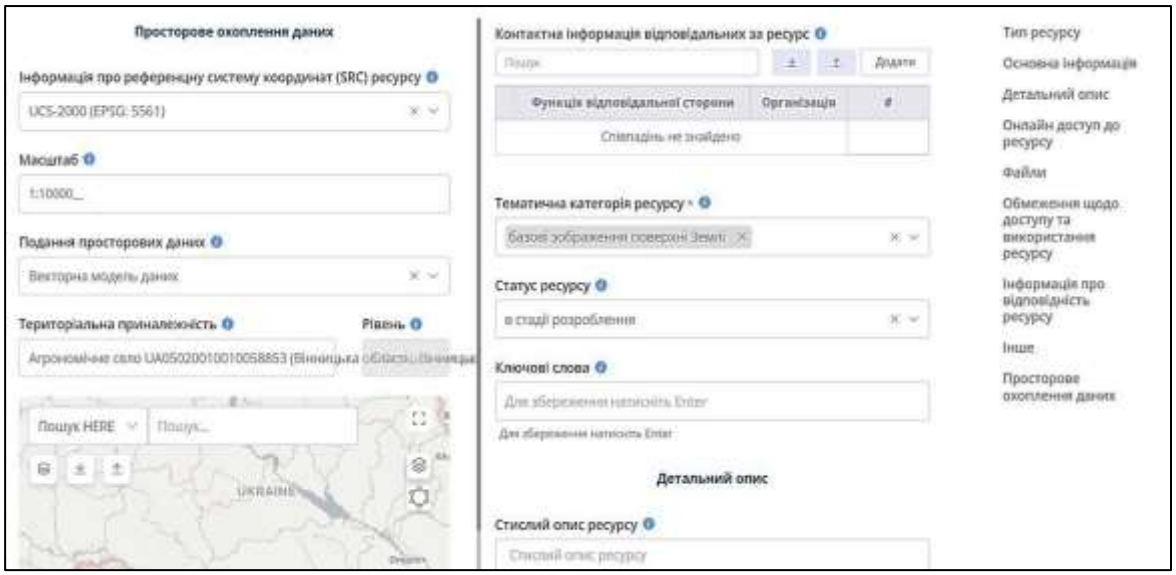

Рис. 6.9. Приклад метаданих, доступних на геопорталі прототипу НІГД

6.1.6. Публічна кадастрова карта України

Це геопортал, на якому оприлюднюються відомості Державного земельного кадастру, який доступний за посиланням: https://map.land.gov.ua/?cc=3461340.1719504707,6177585.367221659&z=6.5&l=kadastr&bl=ortho10k_all (рис. 6.10).

Рис. 6.10. Геопортал зі Публічної кадастрової карти України

На геопорталі Публічної кадастрової карти можна здійснювати навігацію між картографічними матеріалами та виводити на екран комбіновану інформацію з різних інформаційних шарів.

Для цього потрібно використовувати панель «Шари»  , яка розміщена праворуч у робочому вікні (рис. 6.11).

Розділ «Базові шари» містить матеріали, які слугують картографічною основою. В один і той самий момент може бути під’єдн.аний лише один базовий шар.

Примітка: для різних базових шарів доступні різні масштаби відображення Найбільший масштаб, доступний для шару «Ортофотоплани».

Розділ «Шари» містить додаткові інформаційні шари, які можна накладати на обрану картографічну основу (базовий шар) та комбінувати між собою. Одночасно може бути під’єднано декілька додаткових інформаційних шарів.

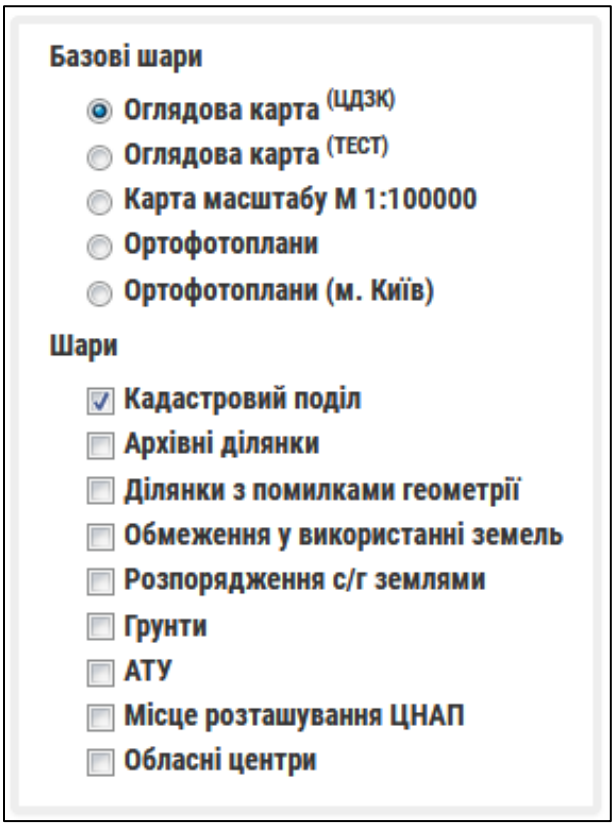

Рис. 6.11. Панель шарів Публічної кадастрової карти

На геопорталі Публічної кадастрової карти доступний картографічний веб-сервіс WMS, призначений для керування просторовими об'єктами. Щоб додати WMS-шар до проекту QGIS з Публічної кадастрової карти, треба у QGIS попередньо завантажити у проєкт будь-який картографічний шар на територію України для коректного відображення WMS шару з Публічної кадастрової карти (наприклад, OpenStreetMap).

У відкритому вікні у полі «Название» треба вказати назву для нового з'єднання WMS (наприклад, Публічна кадастрова карта) (рис. 6.12). У полі «Адрес» слід вказати: https://m1.land.gov.ua/geowebcache/service/wms. Натиснути кнопку «ОК». Нове з'єднання WMS створене.

Далі потрібно виділити рядок «kadastr» та натиснути кнопку «Добавить» (рис. 6.13). До та проекту QGIS буде доданий новий шар, що відображає кадастровий поділ межі земельних ділянок. Аналогічно можна додавати інші шари. Натисніть кнопку «Завершить».

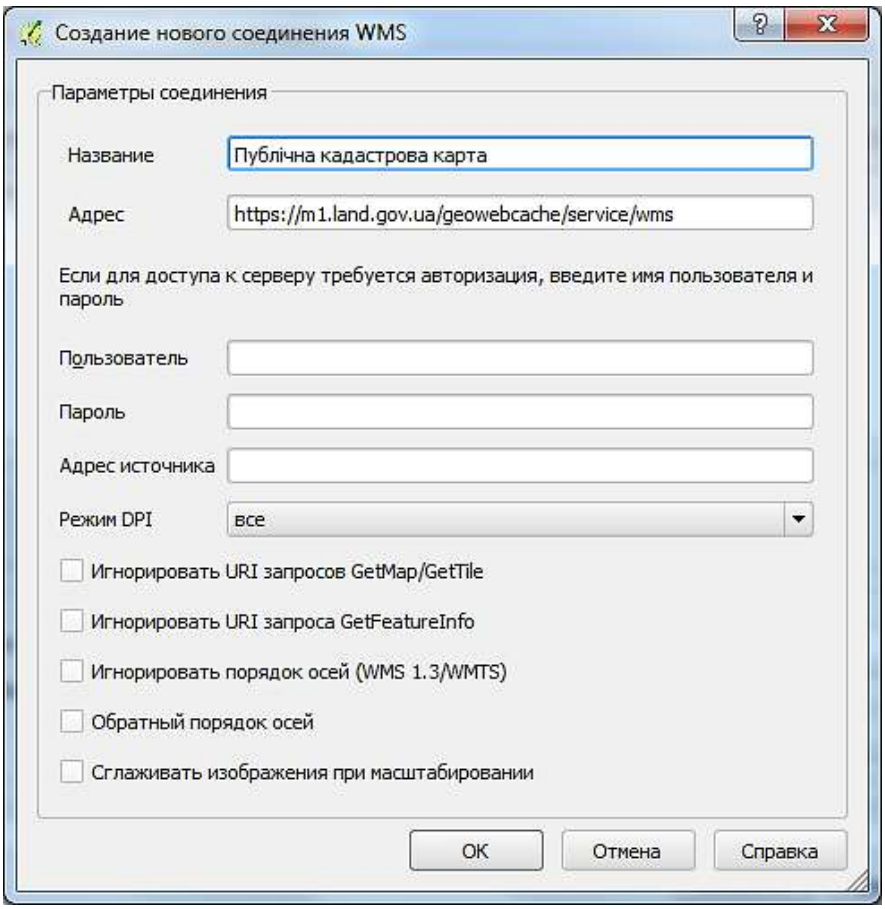

Рис. 6.12. Створення нового з’єднання WMS в QGIS

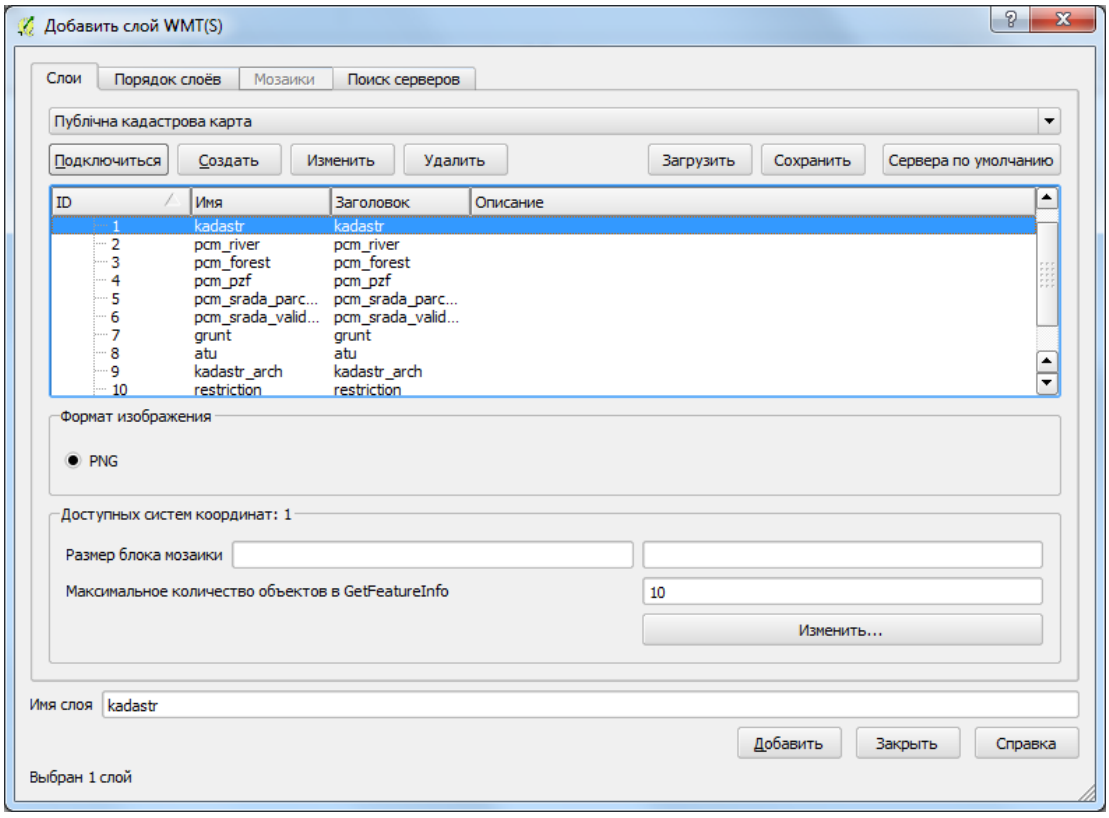

Рис. 6.13. Вікно додавання шарів WMT(S) у QGIS з доданими шарами з Публічної кадастрової карти

Для коректного відображення на карті встановіть для доданих шарів систему координат WGS 84 / Pseudo Mercator EPSG:3857.

Примітка: зверніть увагу, що функція GetFeatureInfo в цій реалізації не підтримується.

Крім того можна під'єднатити Публічну кадастрову карту в середовищі ArcGIS. Доступні також на геопорталі API Е-сервіси, використання яких є можливим після реєстрації та авторизації: e.land.gov.ua. Електронний кабінет «Е-сервіси» Державного земельного кадастру дає громадянам можливість отримувати послуги, які надає орган виконавчої влади, що провадить державну політику у сфері земельних відносин, в електронному вигляді з використанням Інтернету: витяг з державного земельного кадастру, нормативна грошова оцінка, інформація про права.

6.1.7 Геопортал Адміністративно-територіального устрою України (АТУ)

Геопортал «Адміністративно-територіальний устрій України» призначений для створення, наповнення, підтримання в актуальному стані бази даних і метаданих про адміністративно-територіальні одиниці, топографію, населені пункти, автошляхи та залізниці, законодавчі дані про зміни адміністративно-територіального устрою, перспективні плани формування територій громад, бюджетні паспорти громад, дані про об'єднані територіальні громади, кількісні показники стосовно навчальних закладів чи інших соціальних об'єктів, бюджету, розміру дотацій або субвенції тощо. Геопортал розроблений ДП «Науково-дослідний інститут геодезії і картографії» за дорученням віце-прем'єр-міністра – міністра регіонального розвитку, будівництва та житлово-комунального господарства України Геннадія Зубка, за участі Мінрегіону та за сприяння швейцарсько-українського проекту «Підтримка децентралізації в Україні» DESPRO.

Геопортал АТУ доступний за посиланням: https://atu.gki.com.ua/ (рис. 6.14).

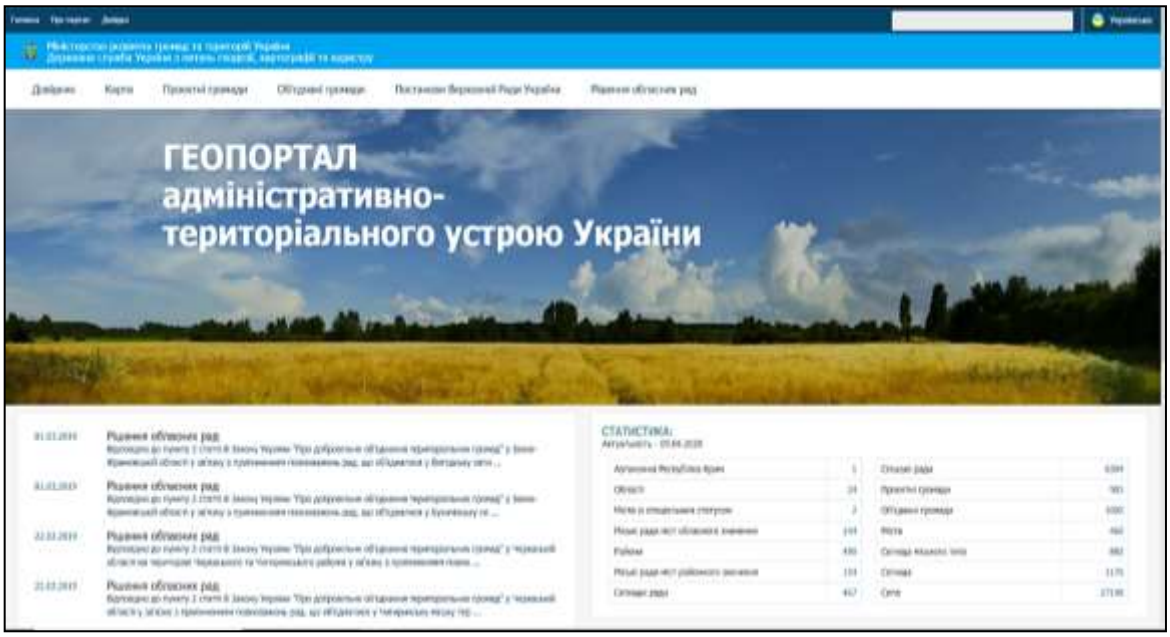

Рис. 6.14. Головна сторінка геопорталу АТУ

На геопорталі АТУ доступні: довідник АТУ, карта АТУ України, проєктні громади, об’єднані громади, постанови Верховної Ради України та рішення обласних рад. Топографічною основою геопорталу є цифрова топографічна карта масштабу 1:100 000, яка призначена для задоволення потреб органів державної влади, економіки, оборони, науки, освіти і громадян країни, а також є основою для створення геоінформаційних систем, спеціальних, тематичних та інших карт і планів, розвитку Національної інфраструктури геопросторових даних загальнодержавного рівня. Перелік шарів, які доступні на геопорталі: математичні елементи, гідрографія та гідротехнічні споруди, населені пункти, дороги та дорожні споруди, рельєф, рослинний покрив та ґрунти, межі адміністративно-територіальних утворень (державний кордон України, межі автономної Республіки Крим, областей, районів, міських рад загальнодержавного та обласного значення, межі міських, селищних та сільських рад, межі об’єднаних територіальних громад) (рис. 6.15, 6.16).

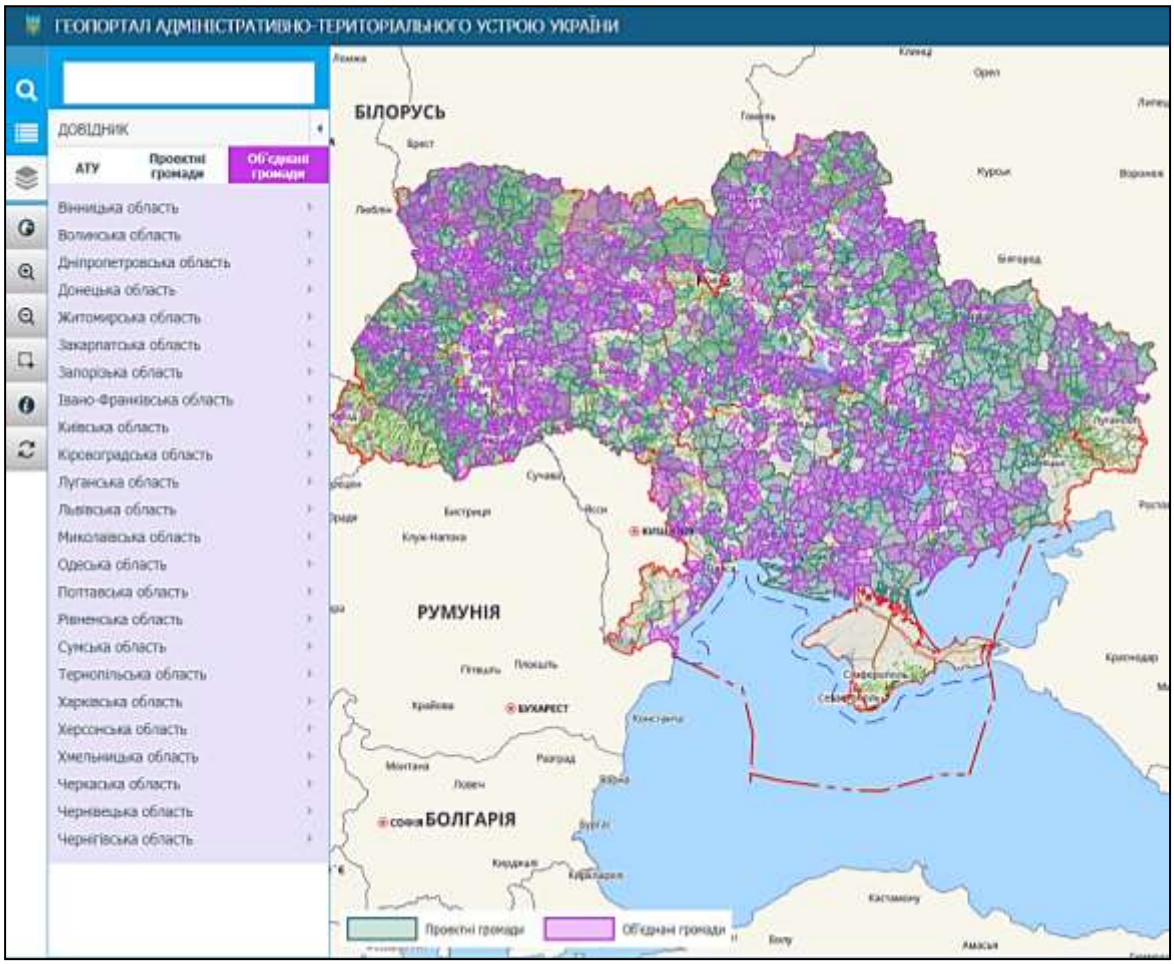

Рис. 6.15. Інтерфейс веб-карти геопорталу АТУ

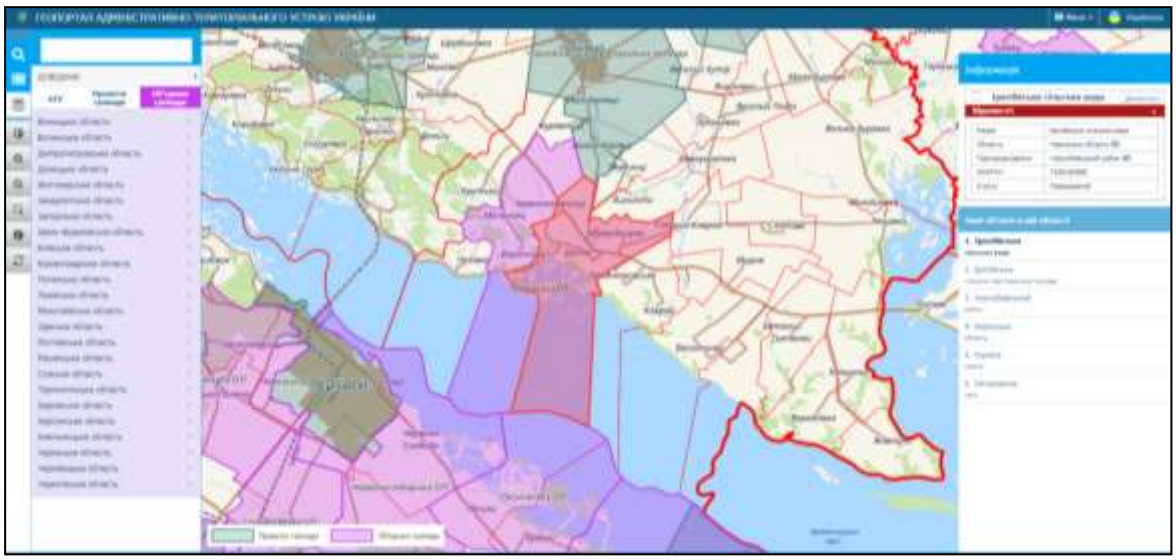

Рис. 6.16. Інтерфейс картки про об’єкт адміністративно-територіального устрою на геопорталі АТУ

6.1.8. Геопортал Державної геодезичної мережі

Цей геопортал створено Державним підприємством «Науково-дослідний інститут геодезії і картографії» з метою: ознайомлення користувачів з державною геодезичною мережею України; вибірки геодезичних та нівелірних пунктів на район робіт з метою їхнього обстеження та майбутнього оформлення заявки на отримання точних координат; отримання довідкової інформації про розміщення, щільність та характеристики окремих геодезичних пунктів на район робіт; надання консультацій користувачам щодо використання геодезичних пунктів та систем координат; забезпечення зворотного зв'язку з користувачами щодо надання ними додаткової інформації про окремі пункти (про стан пункту, стан зовнішнього знаку, доїзд до нього, фотографії тощо).

Геопортал ДГМ доступний за посиланням: https://dgm.gki.com.ua/home (рис. 6.17).

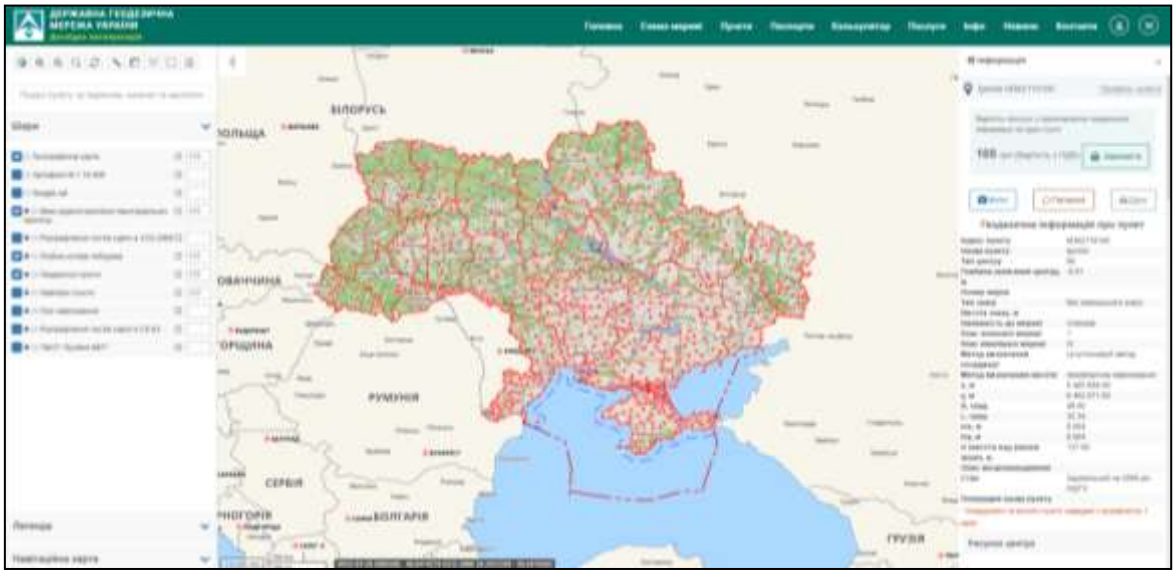

Рис. 6.17. Геопортал ДГМ

Координати геодезичних пунктів на Геопорталі подано з точністю 20 метрів; висоти пунктів – з точністю 10 м.

Для нормальної роботи з Геопорталом потрібен веб-браузер однієї з версій: Google Chrome 24 і вище; Internet Explorer 9 і вище; Opera 12 і вище; Mozilla FireFox 17 і вище.

Інтерфейс програми є відповідним еталонній архітектурі геопорталу консорціуму OGC. Еталонна архітектура геопорталу ґрунтується на загальній сервіс орієнтованій архітектурі (SOA – Service

Oriented Architecture). Головною ознакою SOA є доступність IT-функціональності різним мережним пошуковим механізмам. По суті, SOA надає користувачам різноманітні високорівневі послуги, які можуть бути викликані і застосовані у бізнес-процесі.

Ресурс геопорталу складається з таких наборів даних:

1. Топографічна карта – топографічна карта України масштабу 1:200 000;
2. Адміністративно-територіальні межі – адміністративно-територіальні межі територій АР Криму, областей та районів України;
3. Пункти геодезичної мережі – база даних геодезичних пунктів;
4. Лінійно-кутова побудова – база даних лінійно-кутових вимірів, задіяних в ДГМ;
5. Пункти нівелірної мережі – база даних пунктів нівелірної мережі I та II;
6. Лінії нівелювання – база даних нівелірних перевищень ліній нівелювання I та II класів;
7. Розграфлення аркушів топографічної карти – набір номенклатурних аркушів карт в державній розграфці масштабів 1:10 000 – 1:200 000.

Банк геодезичних даних державної геодезичної мережі та геодезичних мереж згущення містить:

1. Відомості про УСК-2000.
2. Схему геодезичної мережі.
3. Геодезичну інформацію про пункти.
4. Замовлення геодезичної інформації про пункти.
5. Операції з перетворення та трансформування координат.

Геопортал забезпечує виконання таких сервісів:

– перегляд на топографічній карті місцезнаходження геодезичних та нівелірних пунктів, схем геодезичної та нівелірної мереж;

– отримання інформації про окремі пункти;

– виконання вибірок інформації за традиційними методами (за назвою чи наближеними координатами геодезичних пунктів, за адміністративним районом, населентм пунктом, за номенклатурою аркуша карти, контуру на карті);

– формування заявок на отримання точних координат та висот пунктів;

– друк фрагментів карт та схем геодезичної та нівелірної мереж;

– реєстрація користувачів;

– додавання до ресурсів Геопорталу додаткової інформації від користувачів про стан пунктів, доїзд до них, докладний опис, фотографії їхнього місцезнаходження тощо.

6.1.9. Додаткові та довідкові дані і матеріали

Під час автоматизації процесів геопросторового аналізу і моделювання об'єктів і явищ за допомогою прикладного програмування у ГІС слід використовувати такі додаткові вихідні дані та матеріали:

1) Державний реєстр географічних назв (https://land.gov.ua/info/informatsiia-pro-derzhavnyi-reiestr-heohrafichnykh-nazv/);

2) Довідник «Чисельність наявного населення України на 01.01.2019» видання Держкомстату України – для визначення кількості жителів у містах і селищах міського типу: (http://database.ukrcensus.gov.ua/PXWEB2007/ukr/publ_new1/2019/zb_chnn2019.pdf);

3) Банк даних Всеукраїнського перепису населення Державної служби статистики України станом на 2001 рік: (http://database.ukrcensus.gov.ua/_MULT/Database/Census/databasetree_uk.asp) – для подання інформації про кількість жителів в селах;

4) «Тарифное руководство №4 железных дорог Украины» вид. Укрзалізниця, 2001 – для перевірки власних назв залізничних станцій, платформ, зупинних пунктів, роз'їздів (https://www.uz.gov.ua/cargo_transportation/legal_documents/tk4/);

5) Карти автомобільних доріг України загального користування (міжнародні, національні, регіональні та територіальні дороги) Державної служби автодоріг України за даними сайту https://kmplus.ukravtodor.gov.ua/ – для перевірки номерів автодоріг;

6) Постанова Кабінету Міністрів України «Про затвердження переліку автомобільних доріг загального користування державного значення» від 30 січня 2019 року № 55 (https://zakon.rada.gov.ua/laws/show/55-2019-%D0%BF#n13);

7) Перелік внутрішніх водних шляхів, що належать до категорії судноплавних, затверджений постановою № 640 Кабінету Міністрів України від 12 червня 1996 р. Зміни та доповнення до переліку, затверджені постановою № 1688 від 29 жовтня 2003 р. (https://zakon.rada.gov.ua/laws/show/ru/640-96-%D0%BF);

8) ГІС Акціонерне товариство Житомирobленерго: https://www.ztoe.com.ua/gisenergy/;

9) ГІС ПАТ Запоріжжяобленерго: https://www.zoe.com.ua/gis/;

10) ГІС АТ Миколаївобленерго: https://www.energy.mk.ua/gis/#9&47.2000&32.2000;

11) ГІС ПрАТ ДТЕК Київські електромережі: https://kem.energymap.com.ua/#10/50.4920/30.4788;

12) ГІС ПАТ ДТЕК Дніпровські електромережі: https://dnem.energymap.com.ua/#9/48.4738/35.3210;

13) Геодезична системи Кіровоградобленерго: http://kiroe.com.ua:8088/;

14) ГІС АТ ПОЛТАВАОБЛЕНЕРГО: https://www.poe.pl.ua/gis/;

15) ГІС ПрАТ Волиньобленерго: https://www.energosoftcom.lviv.ua/GeoSystem?TypeLoad=VOE;

16) ГІС ВАТ Тернопільобленерго: https://www.energosoftcom.lviv.ua/GeoSystem?TypeLoad=TOE;

17) ГІС АТ Прикарпаттяобленерго: https://www.energosoftcom.lviv.ua/GeoSystem?TypeLoad=PROE;

18) ГІС ПрАТ Львівобленерго: https://tu.loe.lviv.ua/GeoSystem?TypeLoad=LOE;

19) ГІС ПрАТ Рівнеобленерго: https://gisenergy.roe.vsei.ua/;

20) ГІС ЧЕРКАСИОБЛЕНЕРГО: http://www.cherkasyoblenergo.com/gis.html;

21) ГІС АТ Хмельницькобленерго: https://gis.hoe.com.ua/;

22) ГІС АТ Сумиобленерго: http://gis.soe.com.ua/;

23) ГІС Луганське Енергетичне Об`єднання: http://gis.en.lg.ua:8889/;

24) ГІС АТ ДТЕК Одеські електромережі: https://oem.energymap.com.ua/gis.php;

25) ГІС Чернігівобленерго: http://gis.chernihivoblenergo. com.ua:8889/;

26) Реєстр сертифікованих аеродромів (злітно-посадкових майданчиків). Державна авіаційна служба України: https://data.gov.ua/dataset/3af98777-673b-4a23-b5b4-537436735bc2/resource/9b8d7a40-b87e-4ef5-9534-34744b09fe09;

27) Геопортал UA. Аеродроми: https://map.geoportalua.com/airports/;

28) Реєстр гідроспоруд. Річкова інформаційна служба України: https://ukrris.com.ua/hydraulics/;

29) Реєстр морських портів. Адміністрація морських портів України (АМПУ): http://www.uspa.gov.ua/reestr-morskikh-portiv;

30) Кодифікатор адміністративно-територіальних одиниць та територій територіальних громад (версія на останню дату) https://www.minregion.gov.ua/napryamki-diyalnosti/rozvytok-mistsevoho-samovryaduvannya/administratyvno/kodyfikator-administratyvno-terytorialnyh-odynycz-ta-terytorij-terytorialnyh-gromad/;

31) Геопортал Державного кадастру територій та об'єктів природно-заповідного фонду: https://pzf.mepr.gov.ua/

6.2. Джерела геопросторових даних у растровому форматі

Одним з найбільш захопливих аспектів таких програм, як Google Earth, є здатність «бачити» Землю, так ніби ви летите над нею. Це досягнуто шляхом відображення ретельно зшитих разом супутникових і аерофотознімків, що створює ілюзію неначе користувач дивиться на поверхню Землі згори.

Програмне забезпечення Google Earth – це безплатна, вільно-завантажувана програма компанії Google, яка відображає віртуальний глобус. В межах цього проєкту в мережу Інтернет викладено аерофотознімки та супутникові знімки більшої частини Землі. Для деяких регіонів ці знімки сягають дуже високого просторового розрізнення. Програма поширюється під двома різними ліцензіями: Google Earth, безплатна версія та Google Earth Pro, що пропонується для комерційного використання.

Написання власної версії Google Earth – це майже неможливе завданням, однак можна отримати безплатні супутникові знімки у растровому форматі геопросторових даних, щоб використовувати їх надалі у власних геопросторових застосунках.

Растрові дані не обмежується лише зображенням поверхні Землі; з даних у растровому форматі можна отримати й іншу корисну інформацію – наприклад, цифрові моделі рельєфу (digital elevation models – DEM), які містять висоту кожної точки на поверхні Землі, що може бути використане для розрахунку висоти будь-якої потрібної точки. DEM-дані можуть бути використані також для створення двомірних зображень, що відображають різні висоти, використовуючи різні відтінки кольору, або щоб імітувати ефект затінення пагорбів з використанням методу затінення зображення рельєфу (shaded relief imagery).

Далі розглянемо джерела геопросторових даних у растровому форматі.

6.2.1. Landsat та Sentinel-2

Програма Landsat – найтриваліший проєкт з отримання супутникових фотознімків планети Земля. Назва походить від англійських слів «land» – земля і «satellite» – супутник. Перший з супутників в межах цієї космічної програми був запущений в 1972 році; останній, на цей момент, Landsat 8 – 11 лютого 2013 року (рис. 6.18).

Рис. 6.18. Територія КНУБА на супутниковому знімку Landsat 8

Обладнання, встановлене на супутниках Landsat, зробило мільярди знімків. Знімки Landsat охоплюють чорний-білі, традиційні червоно-зелено-сині (RGB), а також інфрачервоні і тепловізійні зображення. Знімки, отримані в США і на станціях отримання даних із супутників по всьому світу, є унікальним ресурсом для виконання безлічі наукових досліджень у галузі сільського господарства, картографії, геології, лісівництва, розвідки, освіти і національної безпеки. Наприклад, супутник Landsat 7 постачає знімки у 8 спектральних діапазонах з просторовою роздільною здатністю, як правило, 30 метрів на піксель, панхроматичні

зображення з роздільною здатністю – 15 метрів на піксель; періодичність збору даних для всієї планети спочатку становила 16-18 діб.

Sentinel-2 – космічна місія дистанційного зондування Землі, запущена Європейським космічним агентством (ESA) відповідно до програми «Copernicus» для дистанційного спостереження і підтримки таких сервісів, як моніторинг лісів, фіксування змін покриву Землі, відстеженням наслідків стихійних лих. Ця місія складається з двох однакових супутників – Sentinel-2A і Sentinel-2B. Місія Sentinel-2 має такі можливості:

– мультиспектральні дані в 13 діапазонах: видимому, близькому інфрачервоному та інфрачервоному короткохвильовому спектрах;

– систематичне покриття поверхні Землі від 56° S до 84° N, прибережних вод, і всього Середземного моря;

– проходить ті самі зони кожні п’ять днів під однаковими кутами зору. Над високими широтами проходи Sentinel-2 перекриваються, а деякі регіони спостерігаються двічі або більше разів що п’ять днів, але під різними кутами огляду;

– роздільна здатність становить 10 м, 20 м і 60 м;

– 290 км поле зору;

– безкоштовне та відкрите поширення даних.

Два ідентичні супутники Sentinel-2 (Sentinel-2A і Sentinel-2B) працюють одночасно. Орбіта є сонячно-синхронною на висоті 786 км, 14,3 обертань на день, із низхідним вузлом орбіти о 10:30 ранку. Цей місцевий час було обрано як компроміс, мінімізуючи покриття хмарами і забезпечуючи достатнє освітлення Сонцем. Цей час близький до місцевого часу, використаний в Landsat і відповідний супутникам SPOT, що дає змогу поєднувати дані Sentinel-2 з історичними знімками для ретроспективного аналізу.

Формат даних. Космічні знімки, як правило, доступні у вигляді файлів GeoTIFF. GeoTIFF – це файл зображення TIFF з геопросторовими тегами, що дає змогу прив'язувати зображення до поверхні Землі. Чимало програмного забезпечення для ГІС, зокрема GDAL, здатне читати файли у форматі GeoTIFF. Оскільки зображення надходять безпосередньо з супутників, файли, які ви можете завантажити звичайно зберігаються в окремих групах даних в окремих файлах. Залежно від супутника, що надсилає зображення, там може бути до восьми різних груп даних, наприклад, Landsat 7 створює окремо червоний, зелений і

синій канали, а також три різних інфрачервоних канали, тепловий канал і «панхроматичний» (чорно-білий) канал високої роздільної здатності. Необроблені супутникові дані складаються з восьми окремих файлів GeoTIFF, по одному для кожного каналу. Група 1 містить дані синього, група 2 – дані зеленого та група 3 – дані червоного кольору. Це окремі файли, то можна об'єднуються для отримання єдиного кольорового зображення (рис. 6.19).

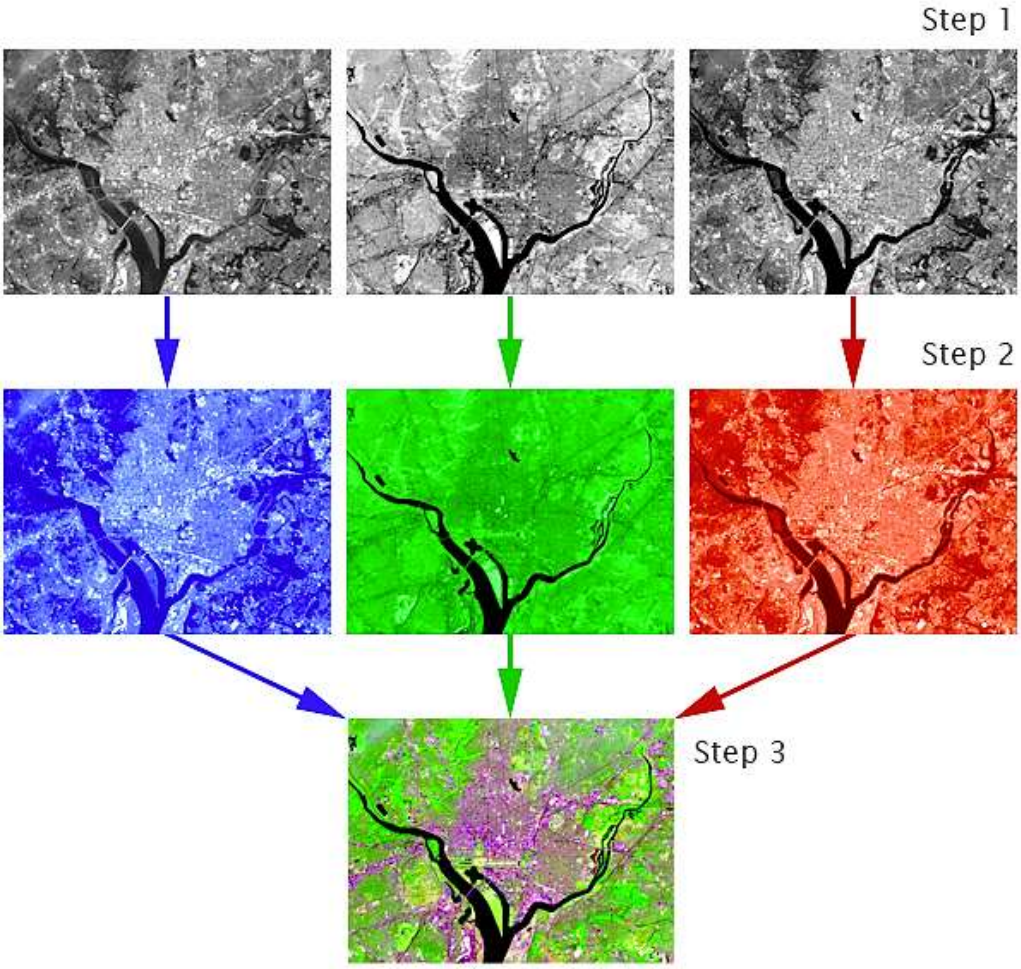

Рис. 6.19. Отримання єдиного кольорового зображення

Ще однією проблемою у роботі з даними Landsat є те, що зображення, отримані за допомогою супутників спотворюються через різноманітні фактори, як-от форма еліпсоїда Землі, висота фотографування місцевості, орієнтація супутника під час отримання зображення. Необроблені дані, таким чином, не відображають сфотографованих об'єктів, однак спотворення можуть бути виправлені за допомогою орторектифікації (orthorectification). У більшості випадків ортотрансформовані версії супутникових знімків можна відразу завантажити з відповідного сайта.

Завантажувати космічні знімки можна з сайта: https://earthexplorer.usgs.gov/, попередньо зареєструвавшись (рис. 20).

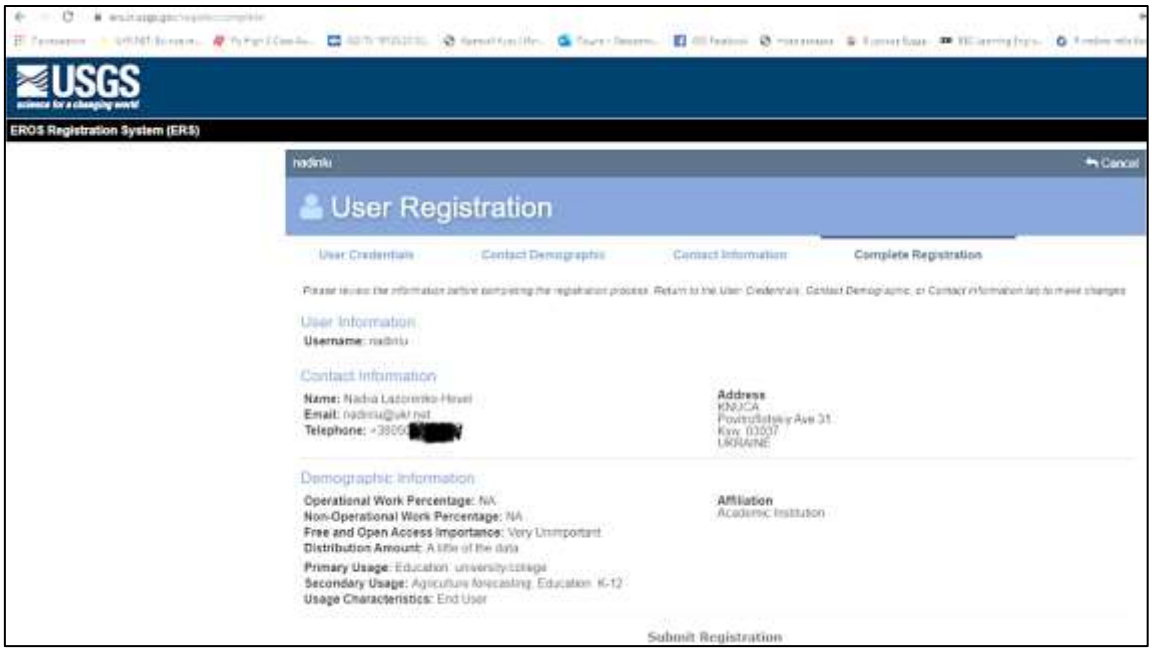

Рис. 6.20. Вікно завершення реєстрації
на геопорталі Геологічної служби США

Після реєстрації і її підтвердження можна завантажувати космічні знімки, для цього потрібно ліворуч на панелі обрати конкретну місцевість або, знаючи координати місцевості, ввести ці дані, й у випадному вікні будуть відфільтровані дані про територію, що вас цікавіть, відтак їх можна буде завантажити (рис. 6.21).

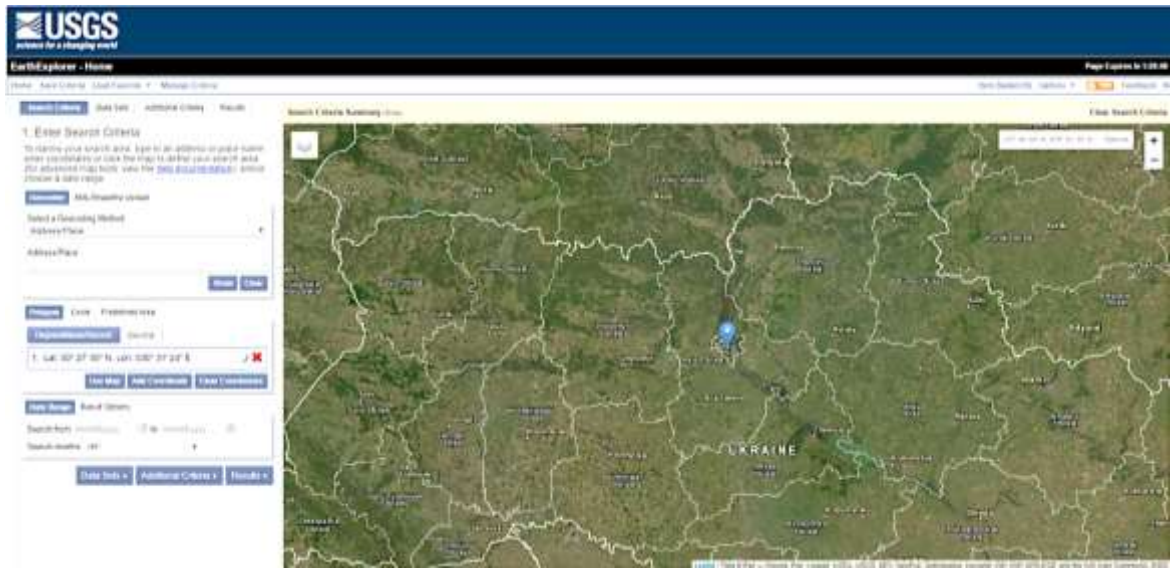

Рис. 6.21. Вікно вибору знімків на геопорталі Геологічної служби США

У закладці «Результати» будуть представлені набори даних супутникових знімків, наведеному прикладі, на територію міста Києва, які можна завантажити, попередньо ознайомившись з метаданими на ці знімки (рис. 6.22).

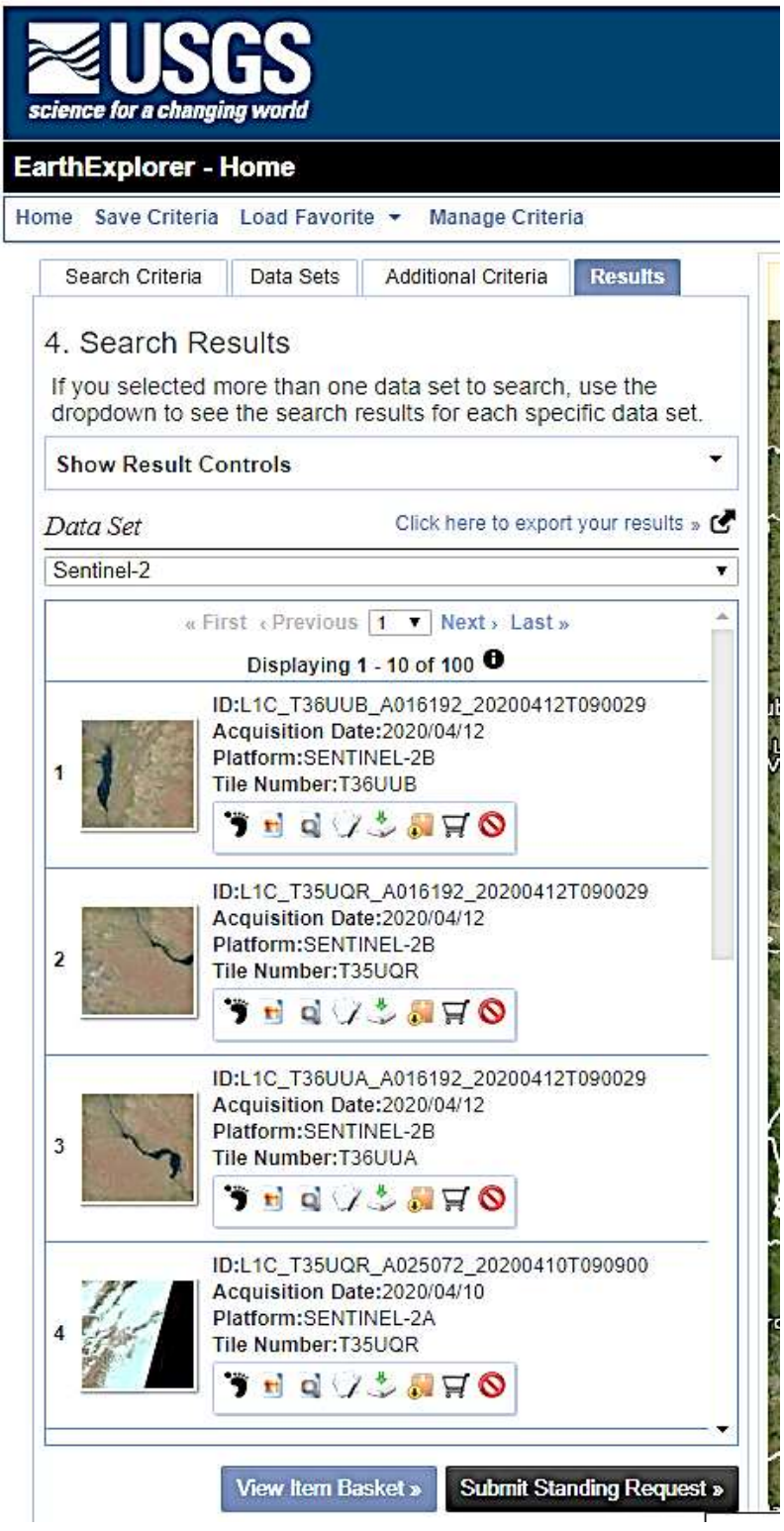

Рис. 6.22. Результати запиту щодо вибору космічних знімків на території міста Києва

Так само можна завантажити космічні знімки Landsat і на їхній основі створювати свої геоінформаційні продукти.

6.2.2. Natural Earth

Веб-сайт Natural Earth (www.naturalearthdata.com) крім векторних даних надає доступ до растрових даних, а саме робить доступними п'ять типів растрових карт двох масштабів: 1:10 000 000 та 1:50 000 000 (рис. 6.23):

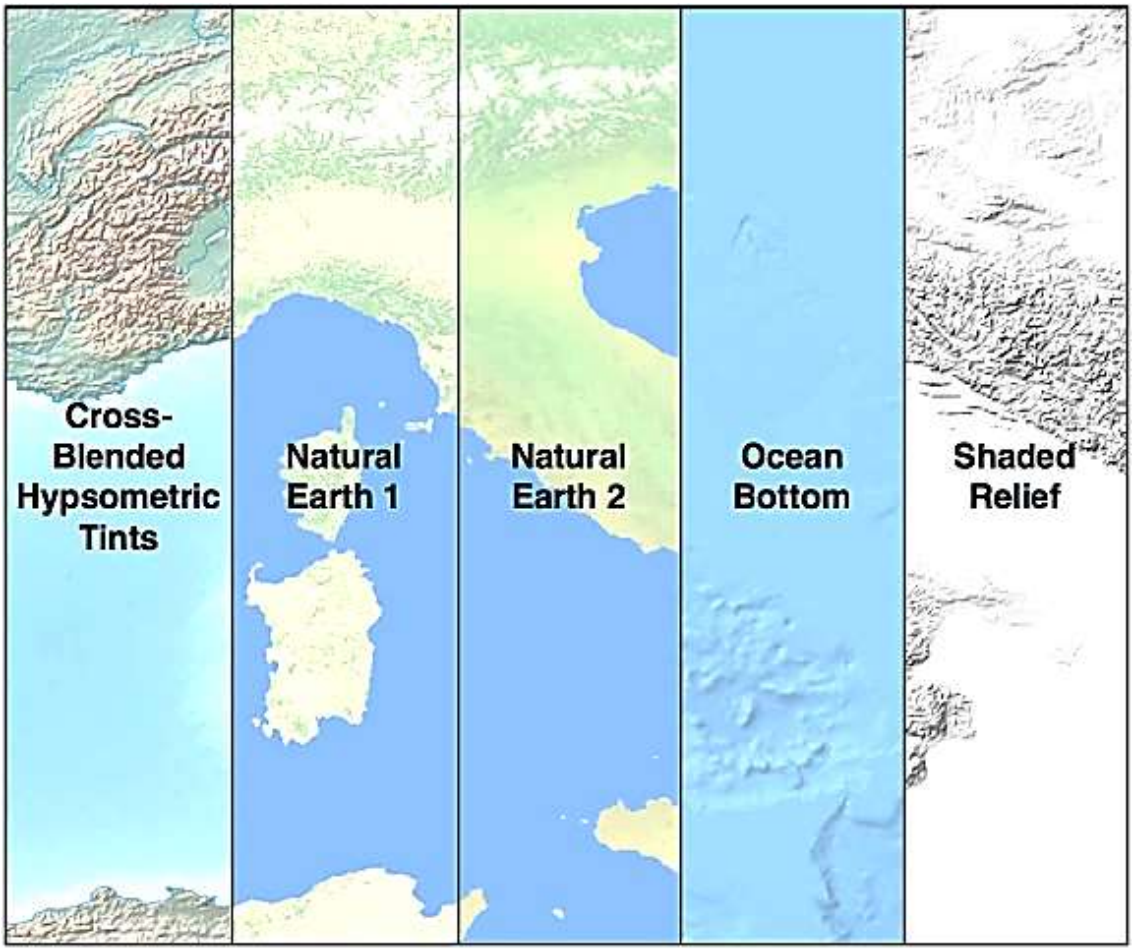

Рис. 6.23. Растрові дані від Natural Earth

1. Растр з назвою «Перехресно -змішані гіпсометричні кольори» (**Cross-Blended Hypsometric Tints**) слугує візуалізації, для якої колір обирається відповідно до висоти і клімату. Ці зображення потім часто поєднують із затіненими зображеннями рельєфу для створення реалістичного зображення поверхні Землі.

2. **Natural Earth 1** та **Natural Earth 2** – більш ідеалізований погляд на поверхню Землі з використанням світлої палітри і м'яко

змішаних кольорів, які становлять відмінний фон для розміщення [ваших власних] геопросторових даних.

3. Океанське дно (**Ocean Bottom**) – це набір даних, що використовує комбінацію затінених зображень рельєфу та кольорів глибин для візуалізації океанського дна.

4. Затонований рельєф (**Shaded Relief**) зображує поверхню Землі з використанням сірих «тіней» на основі даних з високою роздільною здатністю стосовно висот.

Доступним є також один додатковий набір растрових даних, який надає батиметрію (дані про глибини) в масштабі 1:50 000 000. На наступному скріншоті – приклад даних батиметрії для океану, що оточує Нову Зеландію (рис. 6.24).

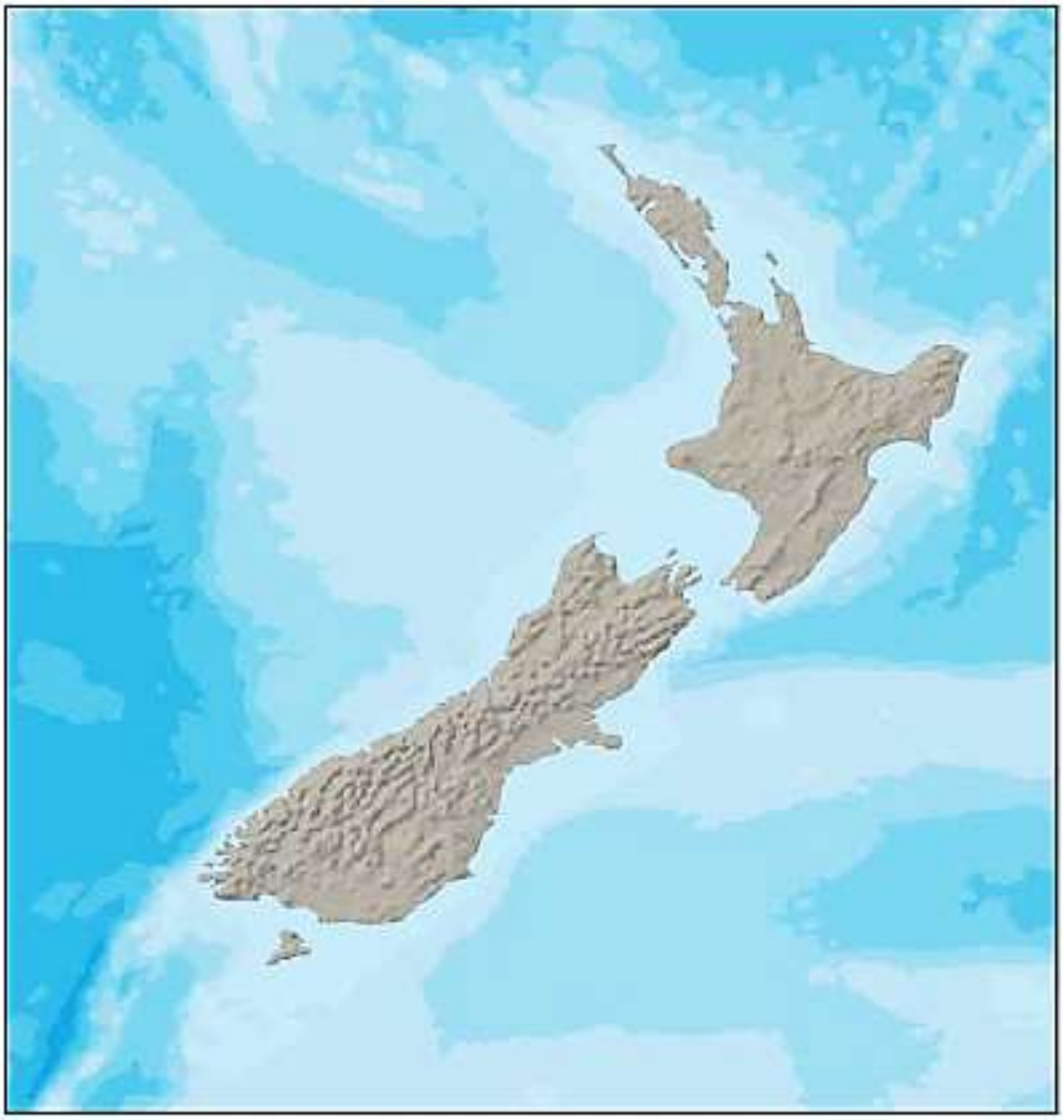

Рис. 6.24. Шар батиметрії на сайті Natural Earth

Більшу частину даних растрового формату на сайті Natural Earth наведено в стандартному форматі зображень TIFF. Єдиним винятком є батиметричні дані, які надаються у формі файла з шарами (layer) Adobe Photoshop з різними відтінками синього кольору, що асоціюються з групами глибин.

У всіх випадках растрові дані подаються у географічній (довгота/широта) проєкції і використовують датум WGS84 для простого переходу між координатами довгота/широта та піксельними координатами растрового зображення.

Растрові дані, як і векторні, легко завантажити з сайту Natural Earth; просто зайдіть на сайт і пройдіть за посиланням Get the Data, щоб завантажити дані у растровому форматі. Ви можете обрати, завантажити дані в масштабі 1:10 000 000 чи 1:50 000 000, а також великий або малий розмір кожного файла.

Після завантаження даних у форматі TIFF, ви можете відкрити файл у графічному редакторі, або скористатися стандартною утилітою командного рядка, такою як gdal_translate для операцій із зображенням. Дані батиметрії можна відкрити або безпосередньо в Adobe Photoshop або в подібному іншому програмному забезпеченні. Кожну смугу, що позначає глибину, представлено окремим шаром у файлі, за замовчуванням вона пов'язана з певним відтінком синього. Ви можете обрати будь-який бажаний колір, а таеож те, які шари показувати, а які приховати. Коли ви закінчите працювати з картою, можна згладити зображення і зберегти його у вигляді файла TIFF для використання у своїх програмних продуктах.

6.2.3. Global Land One-kilometer Base Elevation (GLOBE)

GLOBE – це міжнародний проєкт зі створення високоякісних цифрових моделей рельєфу (DEM) для всього світу. Це набір DEM файлів у вільному доступі, які можуть бути використані для геопросторового аналізу та розроблення програмних продуктів.

На скріншоті (рис. 6.25) відтворено дані цифрової моделі рельєфу GLOBE DEM північної частини Чилі у вигляді зображення в градаціях сірого.

Як і всі ЦМР-дані, GLOBE використовує растрові значення, що відображають висоти у конкретній точці на поверхні Землі. Ці дані складаються з 32-бітних цілих чисел зі знаком «+» або «–», що означають висоту в метрах вище (або нижче) рівня моря.

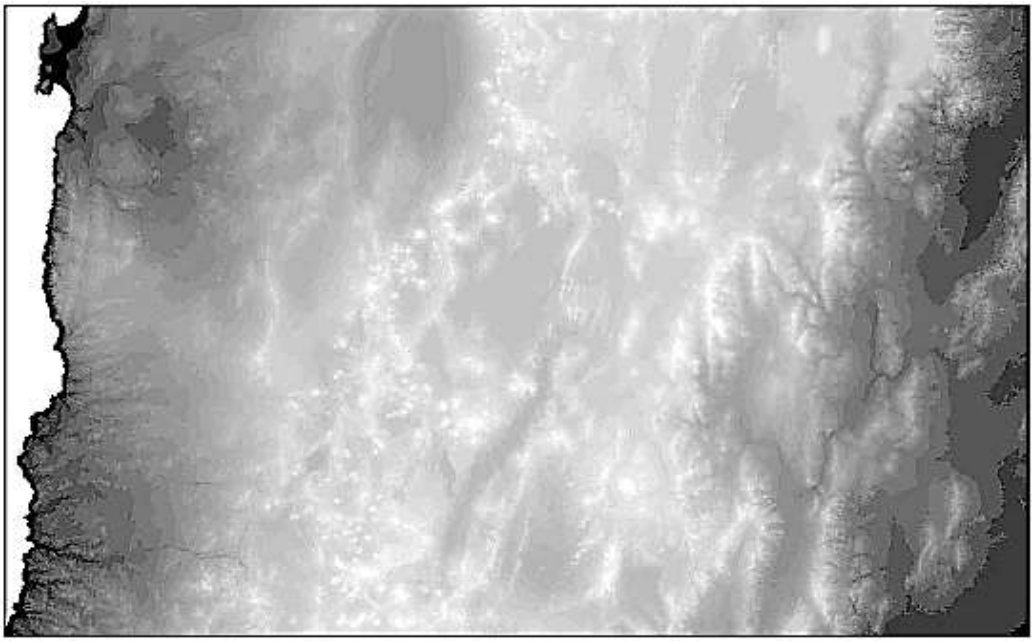

Рис. 6.25. ЦМР північної частини Чилі

Кожна комірка, або «піксель», у растрових даних – це висота квадрата на поверхні Землі, завширшки 30 кутових секунд довготи і заввишки 30 кутових секунд дуги широти (рис. 6.26).

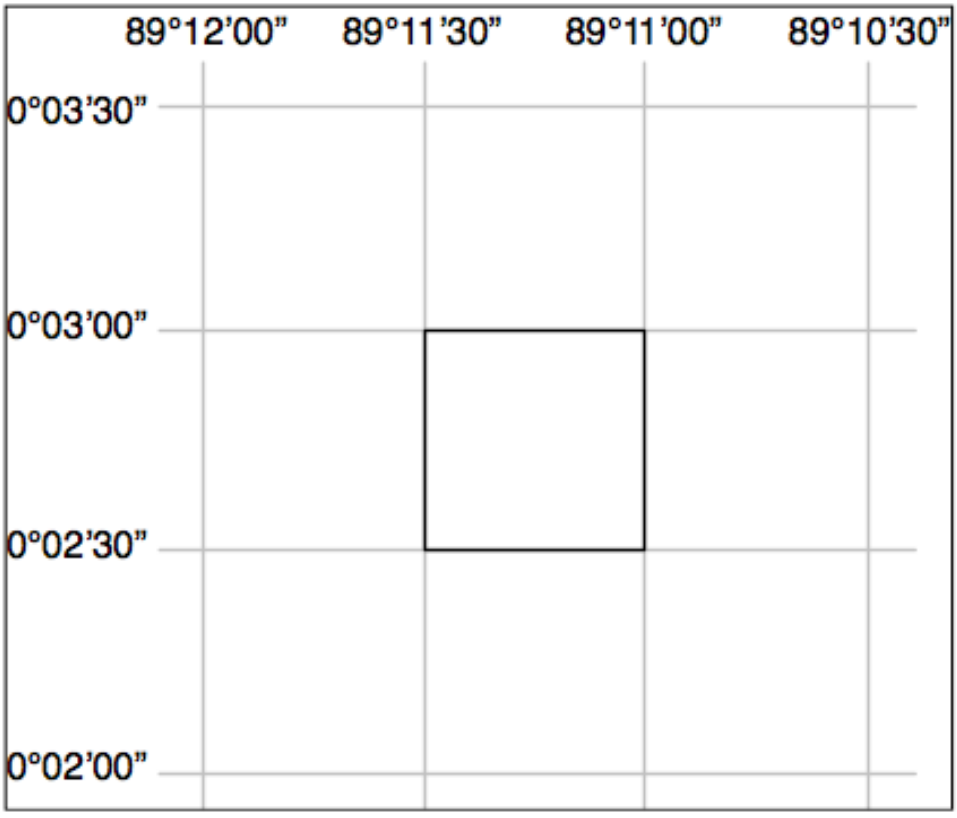

Рис. 6.26. Комірка розміром 30 кутових секунд довготи
та 30 кутових секунд дуги широти

Зверніть увагу, що 30 кутових секунд приблизно дорівнюють 0,00833 градусам широти або довготи, що приблизно є відповідним квадрату завширшки і заввишки в один кілометр.

Сирі дані GLOBE – це довгий список 32-розрядних цілих чисел у форматі зворотного порядку байтів, де клітини зчитуються зліва направо, а потім зверху знизу (табл. 6.3).

Таблиця 6.3

Дані GLOBE

x = 0, y = 0	x = 1, y = 0	…	x = 10800, y = 0
x = 0, y = 1	x = 1, y = 1	…	x = 10800, y = 1
…	…	…	…
x = 0, y = 6000	x = 1, y = 6000	…	x = 10800, y = 6000

Файли (*.hdr) містять докладнішу інформацію про дані ЦМР, зокрема про ширину, висоту та прив'язку області. Інструменти, такі як GDAL, дають можливість читати сирі дані з *.hdr файла.

Отримати дані ЦМР можна для будь-якої ГІС з основного веб-сайта GLOBE за посиланням: http://ngdc.noaa.gov/mgg/topo/globe.html.

Для отримання докладнішої документації даних GLOBE, ви можете скористатися посиланням Get Data Online, щоб завантажити попередньо обчислені набори даних або вибрати певну область для якої необхідно завантажити DEM дані.

Якщо ви завантажуєте одну з попередньо створених областей, вам потрібно буде також завантажити відповідний *.hdr файл так, що дані можуть мати прив'язку і оброблятися за допомогою GDAL. Якщо ви обираєте довільну область для завантаження, відповідний *.hdr файл буде створений для вас, тільки-но ви оберете тип експорту ESRI ArcView так, що файл буде створений у форматі, придатному для GDAL.

Якщо ви завантажуєте попередньо створену область, заголовний *.hdr файл може бути досить важко знайти. Відповідні *.hdr файли у форматі ESRI можуть бути завантажені з сайта: http://www.ngdc.noaa.gov/mgg/topo/elev/esri/hdr/.

Після того як дані завантажено, помістіть файл «сирої» DEM в ту саму директорію, що й файл *.hdr. Ви можете відкрити дані, використовуючи файл безпосередньо у GDAL, наприклад, як цей:

```
import osgeo.gdal
dataset = osgeo.gdal.Open(«j10g.bil»)
```

Набір даних буде складатися з однієї смуги растрових даних, які потім можна читати або обробляти, використовуючи бібліотеку GDAL і пов'язані інструменти.

6.2.4. National Elevation Dataset (NED)

Національна база даних висот (National Elevation Dataset – NED) є цифровим набором даних висот високої роздільної здатності, створеним Геологічною службою США (US Geological Survey). Вона охоплює континентальну частину США, Аляску, Гаваї та інші території США. Більшу частину Сполучених Штатів покрито даними висот з роздільною здатністю 30 м/піксель або 10 м/піксель, з деякими областями, з роздільною здатністю у 3 м/піксель, тоді як Аляска – переважно тільки у 60 м/піксель.

Дані NED можуть бути завантажені у кількох форматах, зокрема GeoTIFF та ArcGRID, які можуть бути оброблені за допомогою GDAL.

Як і в інших DEM-даних, кожен «піксель» растрового зображення представляє висоту певної території на поверхні Землі. У NED дані подано як висоту в метрах вище або нижче від еталонної висоти, відомої як Північно-Американського відлік висоти 1988 (North American Vertical Datum of 1988). Це приблизно дорівнює висоті вище або нижче від рівня моря, що дає змогу враховувати припливи чи інші зміни.

Веб сайт Національної бази даних висот: http://nationalmap.gov/elevation.html. Цей сайт описує набір даних NED. Для завантаження потрібно скористатися засобом перегляду National Map Viewer, доступним за посиланням http://viewer.nationalmap.gov/viewer/.

6.3. Джерела інших типів геопросторових даних

Крім векторних і растрових джерел геопросторових даних є інші джерела, наприклад, реєстри та статистичні джерела геопросторових даних. Тому розглянемо деякі з них.

6.3.1. Банк даних Всеукраїнського перепису населення

Проведення чергового Всеукраїнського перепису населення визначено розпорядженням Кабінету Міністрів України від 9 грудня 2020 року № 1542-р «Про проведення у 2023 році Всеукраїнського перепису населення».

Перепис проводиться шляхом обходу та опитування населення на дату перепису. Перепис населення визначається як захід, що дозволяє здійснювати через регулярні інтервали часу офіційний облік чисельності населення, яке проживає на території країни загалом та на всіх рівнях її адміністративно-територіального устрою. Дані перепису потрібні для прогнозування й управління соціально-економічним розвитком країни, реалізації демографічної політики, бюджетного планування та інших цілей. Дані перепису було зібрано в Банку даних Всеукраїнського перепису населення, який розміщено на сайті Державної служби статистики України: http://database.ukrcensus.gov.ua/MULT/Database/Census/databasetree_uk.asp. Ці дані можна використовувати для створення своїх програмних продуктів.

Крім того, є Довідник «Чисельність наявного населення України на 01.01.2019» видання Держкомстату України – для визначення кількості жителів у містах і селищах міського типу, який доступний за посиланням: http://database.ukrcensus.gov.ua/PXWEB2007/ukr/publ_new1/2019/zb_chnn2019.pdf

6.3.2. Державний реєстр географічних назв

Створення та ведення Державного реєстру географічних назв відбувається на підставі Закону України «Про географічні назви» та постанови Кабінету Міністрів України від 11 травня 2006 р. № 622 «Про затвердження Положення про Державний реєстр географічних назв», наказу Мінагрополітики від 11.06.2014 № 219 «Про затвердження обліково-реєстраційних форм Державного реєстру географічних назв», зареєстрованого у Мін'юсті 24.06.2014 за № 683/25460.

Географічні назви, включені до Державного реєстру географічних назв, призначені для застосування органами державної влади та органами місцевого самоврядування, установами, організаціями, підприємствами, засобами масової інформації та громадянами.

Однозначні географічні назви мають важливе значення для ухвалення управлінських рішень державними органами, економічного розвитку, екології, торгівлі, збереження культурної спадщини, міжнародної співпраці, координації служб екстреної допомоги, економії часу та коштів у разі надзвичайних ситуаціях тощо.

Державний реєстр географічних назв створено у вигляді автоматизованої системи обліку унормованих географічних назв, яка потребує постійної актуалізації за офіційними даними відомостей про найменування та перейменування географічних об’єктів, зміну їхнього статусу, категорії, виду тощо.

Наповнення електронної бази даних назв географічних об’єктів України Державного реєстру географічних назв виконано за адміністративно-територіальними одиницями (АР Крим, області, райони, міста, сільські та селищні ради) на основі топографічної карти масштабу 1:100 000.

База даних Державного реєстру географічних назв містить понад 100 000 назв населених пунктів та одиниць адміністративно-територіального устрою, фізико-географічних та соціально-економічних об’єктів України станом на 1 грудня 2017 року.

Сформовані на територію Автономної Республіки Крим та областей переліки назв адміністративно-територіальних одиниць та населених пунктів, фізико-географічних і соціально-економічних об’єктів України, зареєстрованих у Державному реєстрі географічних назв, містять відомості про реєстраційний номер, унормовану назву географічного об’єкта, унормовану назву латиницею, вид географічного об’єкта, адміністративний статус (для населених пунктів), адміністративно-територіальну прив’язку, географічні координати (широта та довгота).

Доступ до Реєстру географічних назв за посиланням: https://land.gov.ua/info/informatsiia-pro-derzhavnyi-reiestr-heohrafichnykh-nazv/.

Джерел геопросторових даних є безліч і їхня кількість зростає щодня. В розділі наведено одні з найчастіше використовуваних.

7. ЗАСТОСУВАННЯ PYTHON У QGIS

QGIS (раніше відомий як «Quantum GIS») – вільна крос-платформова геоінформаційна система (ГІС). QGIS є однією з найбільш функціональних і зручних настільних геоінформаційних систем, які динамічно розвиваються. QGIS – це зручна ГІС з відкритим кодом, що розповсюджується на умовах GNU General Public License. QGIS є проєктом Open Source Geospatial Foundation (OSGeo). Вона працює на Linux, Unix, Mac OSX, Windows та Android, підтримує безліч растрових та векторних форматів, бази даних та має багаті можливості.

Основним призначенням системи є оброблення й аналіз геопросторових даних, підготовка різної картографічної продукції. QGIS дає користувачам змогу створювати карти з безліччю шарів, використовуючи різні картографічні проєкції. Мапи можуть бути зібрані в різні формати і використовуватися для різних цілей. Інтерфейс QGIS побудований на базі бібліотеки Qt. Пакет має гнучку систему розширень, які можна створювати мовами C++ і Python. QGIS забезпечує інтеграцію з іншими відкритими ГІС-пакетами, зокрема PostGIS, GRASS і MapServer, щоб дати користувачам широкі функціональні можливості. Плагіни, написані на Python, C++, розширюють можливості QGIS. Є плагіни для геокодування, виконання геооброблення, схожі на стандартні інструменти ArcGIS, інтерфейс PostgreSQL/PostGIS, SpatiaLite і MySQL баз даних і використання Mapnik як карти візуалізації.

Починаючи з версії 9.0, в QGIS реалізовано підтримку сценаріїв мовою Python. Прив’язки (bindings) PyQGIS залежать від SIP та PyQt4. Основною причиною використання SIP замість більш поширеного SWIG є те, що код QGIS залежить від бібліотек Qt. А прив’язки для Qt (PyQt) також генеруються за допомогою SIP, це дає змогу досягти прозорої інтеграції PyQGIS та PyQt.

Застосовують декілька способів прив’язки QGIS і Python:

- автоматичне виконання Python коду в разі запуску QGIS;
- виконання команд у консолі Python QGIS;
- створення та використання плагінів на Python;
- створення власних програм за допомогою API QGIS.

Повний опис доступний за посиланням: https://qgis.org/api/. В ньому зібрано інформацію про всі класи бібліотек QGIS. QGIS Python API практично ідентичне C++ API.

7.1. Створення сценаріїв у консолі Python

QGIS надає інтегровану консоль Python для створення сценаріїв. Його можна відкрити з меню Plugins ► Python Console (рис. 7.1).

Рис. 7.1. Консоль QGIS Python

На знімку екрана показано, як отримати поточний обраний шар у списку шарів, показати його ідентифікатор і, за бажанням, якщо це векторний шар, показати кількість функцій. Для взаємодії із середовищем QGIS є змінна **`iface`**, яка являє собою екземпляр **`QgisInterface`**. Цей інтерфейс дає змогу отримати доступ до полотна карти, меню, панелей інструментів та інших частин програми QGIS.

Для зручності користувача наступні оператори виконуються під час запуску консолі (у подальшому можна буде встановити додаткові початкові команди)

```
from qgis.core import *
import qgis.utils
```

Для тих, хто часто користується консоллю, може бути корисним встановити ярлик для запуску консолі (у «Налаштуваннях» ► «Комбінації клавіш»…)

7.2. Розширення на Python

Функціональність QGIS можна розширити за допомогою плагінів. Плагіни можна писати на Python. Основна перевага над плагінами C++ – це простота розповсюдження (немає компіляції для кожної платформи) і простіша розробка.

Після появи підтримки Python було написано багато плагінів, що охоплюють різні функції. Інсталятор плагіна дає користувачам змогу легко отримувати, оновлювати та видаляти плагіни Python.

Плагіни Python також доступні для сервера QGIS.

Плагіни обробки можна використовувати для обробки даних. Їх легше розробити, вони більш специфічні та легші, ніж плагіни Python.

7.3. Запуск коду Python під час запуску QGIS

Застосовують різні методи запуску коду Python під час кожного запуску QGIS:

– створення скрипту startup.py

– встановлення PYQGIS_STARTUP змінної середовища для наявного файлу Python

– вказівка сценарію запуску за допомогою параметра.--code init_qgis.py

Щоразу, коли запускається QGIS, у домашньому каталозі Python користувача та списку системних шляхів слід шукати файл із назвою «startup.py». Якщо цей файл існує, він виконується вбудованим інтерпретатором Python.

Шлях у домашньому каталозі користувача зазвичай знаходять за адресою:

```
Linux:.local/share/QGIS/QGIS3 ;
Windows:AppData\Roaming\QGIS\QGIS3 ;
macOS:Library/Application Support/QGIS/QGIS3 ;
```

Системні шляхи за замовчуванням залежать від операційної системи. Щоб знайти шляхи, які вам підходять, відкрийте консоль Python і запустіть, `QStandardPaths.standardLocations (QStandardPaths.AppDataLocation)`, щоби переглянути список каталогів за замовчуванням.

Сценарій startup.py виконується одразу після ініціалізації python у QGIS на початку запуску програми.

Ви можете запустити код Python безпосередньо перед завершенням ініціалізації QGIS, встановивши для PYQGIS_STARTUP змінної середовища шлях до інаявного файла Python.

Потрібний код буде запущений до завершення ініціалізації QGIS. Цей метод дуже корисний для очищення sys.path, який може мати небажані шляхи, або для ізоляції/завантаження початкового середовища без використання віртуального середовища, наприклад, встановлення homebrew або MacPorts на Mac.

Ви можете надати спеціальний код для виконання як параметр запуску QGIS. Для цього створіть файл Python, наприклад qgis_init.py, щоб виконати та запустити QGIS із командного рядка за допомогою .qgis --code qgis_init.py

Код, наданий через --code виконується пізніше, на етапі ініціалізації QGIS, після завантаження компонентів програми.

Для того щоби надати додаткові аргументи для вашого --codeсценарію або для іншого коду Python, який виконується, ви можете використовувати --py-args аргумент. Будь-який аргумент після --py-args або перед --аргументом (якщо він є) буде переданий Python, але проігнорований самою програмою QGIS.

У наведеному далі прикладі myfile.tif буде доступним через sys.argvPython, але не буде завантажений QGIS, тоді як otherfile.tif буде завантажене QGIS, але його немає в sys.argv.

qgis --code qgis_init.py --py-args myfile.tif -- otherfile.tif

Якщо потрібен доступ до кожного параметра командного рядка з Python, можна використовувати QCoreApplication.arguments()

QgsApplication.instance().arguments()

7.4. Програми Python

Для автоматизації процесів зручно створювати сценарії. З PyQGIS це цілком можливо – імпортуйте модуль qgis.core , ініціалізуйте його, і ви готові до обробки.

Або можна створити інтерактивну програму, яка використовує функції ГІС – виконувати вимірювання, експортувати карту як PDF тощо. Модуль qgis.gui надає різні компоненти графічного інтерфейсу користувача, зокрема віджет полотна карти, який можна включити в програму з підтримкою масштабування, панорамування та/або інші користувацькі інструменти карти.

Користувальницькі програми PyQGIS або автономні сценарії повинні бути налаштовані для пошуку ресурсів QGIS, таких як інформація про проєкцію та постачальників для читання векторних і растрових шарів. Ресурси QGIS ініціалізуються додаванням кількох рядків на початку вашої програми або сценарію. Код для ініціалізації QGIS для користувальницьких програм і автономних сценаріїв схожий. Не використовуйте як qgis.py назву для свого сценарію, оскільки Python не зможе імпортувати прив'язки.

7.4.1. Використання PyQGIS в автономних сценаріях

Для того щоби запустити окремий сценарій, ініціалізуйте ресурси QGIS на початку сценарію:

```
from qgis.core import *
# Supply path to qgis install location
QgsApplication.setPrefixPath(«/path/to/qgis/installation»,
True)
# Create a reference to the QgsApplication.  Setting the
# second argument to False disables the GUI.
qgs = QgsApplication([], False)
# Load providers
qgs.initQgis()
# Write your code here to load some layers, use processing
# algorithms, etc.
# Finally, exitQgis() is called to remove the
# provider and layer registries from memory
qgs.exitQgis()
```

Спочатку імпортуємо модуль qgis.core і налаштовуємо префіксний шлях. Префіксний шлях – це розміщення QGIS у вашій системі. Він налаштовується в скрипті шляхом виклику `setPrefixPath()` методу. Другий аргумент `setPrefixPath()` має значення **`True`**, вказуючи, що слід використовувати шляхи за замовчуванням.

Шлях встановлення QGIS залежить від платформи; найпростіший спосіб знайти його для вашої системи – скористатися сценарієм у консолі Python з QGIS і переглянути результат виконання:

```
QgsApplication.prefixPath()
```

Після того як шлях префікса налаштовано, зберігаємо посилання на `QgsApplication` в змінній qgs. Другий аргумент має значення False, вказуючи, що ми не плануємо використовувати GUI, оскільки пишемо автономний сценарій. Якщо QgsApplication налаштовано, завантажуємо постачальників даних QGIS і реєстр шарів, викликаючи `initQgis()` метод.

```
qgs.initQgis()
```

Після ініціалізації QGIS ми готові написати решту сценарію. Нарешті закінчуємо викликом `exitQgis()` видалення постачальників даних і реєстру шарів із пам'яті.

```
qgs.exitQgis()
```

7.4.2. Використання PyQGIS у спеціальних програмах

Єдина відмінність між використанням PyQGIS в автономних сценаріях і власною програмою PyQGIS полягає в другому аргументі під час створення екземпляра `QgsApplication`. Передайте **`True`** замість **`False`**, щоб вказати, що плануєте використовувати GUI.

```
from qgis.core import *
# Supply the path to the qgis install location
QgsApplication.setPrefixPath(«/path/to/qgis/installation»,
True)
# Create a reference to the QgsApplication.
# Setting the second argument to True enables the GUI.  We
need
# this since this is a custom application.
qgs = QgsApplication([], True)
# load providers
qgs.initQgis()
# Write your code here to load some layers, use processing
# algorithms, etc.
# Finally, exitQgis() is called to remove the
# provider and layer registries from memory
qgs.exitQgis()
```

Тепер можна працювати з API QGIS – завантажувати шари та виконувати певну обробку або запускати графічний інтерфейс з полотном карти.

7.5. Запуск програм

Системі потрібно вказати, де шукати бібліотеки QGIS і відповідні модулі Python, якщо вони не зберігаються у добре відомому місці – інакше Python скаржитиметься:

```
>>> import qgis.core
ImportError: No module named qgis.core
```

Це можна виправити, встановивши `PYTHONPATH`змінну середовища. У наступних командах `<qgispath>` потрібно замінити на ваш фактичний шлях встановлення QGIS:

- у Linux: **експорт `PYTHONPATH=/<qgispath>/share/qgis/python`**
- у Windows: **установити `PYTHONPATH=c:\<qgispath>\python`**
- на macOS: **експорт `PYTHONPATH=/<qgispath>/Contents/Resources/python`**

Тепер шлях до модулів PyQGIS відомий, але він залежить від бібліотек `qgis_core`і `qgis_gui`(модулі Python служать лише оболонками). Шлях до цих бібліотек може бути невідомий операційній системі, і тоді ви знову отримаєте помилку імпорту (повідомлення може відрізнятися залежно від системи):

```
>>> import qgis.core
ImportError: libqgis_core.so.3.2.0: cannot open shared
object file:
  No such file or directory
```

Виправте це, додавши каталоги, де зберігаються бібліотеки QGIS, до шляху пошуку динамічного компонувальника:

- у Linux: **експорт `LD_LIBRARY_PATH=/<qgispath>/lib`**
- у Windows: **установить `PATH=C:\<qgispath>\bin;C:\<qgispath>\apps\<qgisrelease>\bin;%PATH%`,**

де `<qgisrelease>` має бути замінене типом випуску, який ви націлюєте (наприклад, `qgis-ltr`, `qgis`, `qgis-dev`)

Ці команди можна вписати в завантажувальний скрипт, який налаштовуватиме систему перед запуском програми. При розгортанні

програм, що використовують PyQGIS, можна використовувати один з двох способів:

– вимагати від користувача встановлення QGIS перед встановленням вашої програми. Інсталятор програми повинен шукати типові розміщення бібліотек QGIS і дозволити користувачеві встановити шлях, якщо його не знайдено. Цей підхід має перевагу в тому, що він простіший, однак він потребує від користувача виконання додаткових кроків;

– постачати QGIS разом зі своїм застосунком. Підготовка у випуску стане більш складною і розмір програми зросте, але користувачі будуть позбавлені необхідності завантажувати і встановлювати додаткове програмне забезпечення.

Ці дві моделі розгортання можна змішувати. Ви можете надати автономні програми для Windows і macOS, але для Linux залиште встановлення GIS користувачеві та його менеджеру пакетів.

7.6. Завантаження проектів

Для того щоби завантажити проект у поточну програму QGIS, вам потрібно створити екземпляр класу `QgsProject`. Це одиночний клас, тому для `instance()` цього потрібно використовувати його метод. Ви можете викликати його `read()` метод, передаючи шлях проєкту, який потрібно завантажити:

```
 # If you are not inside a QGIS console you first need to
import
 # qgis and PyQt classes you will use in this script as
shown below:
 from qgis.core import QgsProject
 # Get the project instance
 project = QgsProject.instance()
 # Print the current project file name (might be empty in
case no projects have been loaded)
 # print(project.fileName())
 # Load another project
project.read('testdata/01_project.qgs')
print(project.fileName())
testdata/01_project.qgs
```

Якщо потрібно внести зміни в проєкт (наприклад, додати або видалити деякі шари) і зберегти зміни, викличте метод `write()` свого екземпляра проєкту. Метод `write()` також приймає додатковий шлях для збереження проєкту в новому місці:

```
# Save the project to the same
project.write()
# ... or to a new file
project.write('testdata/my_new_qgis_project.qgs')
```

Обидві функції `read()` і `write()` повертають логічне значення, за допомогою якого можна перевірити, чи була операція успішною.

Якщо ви пишете окрему програму QGIS, щоб синхронізувати завантажений проєкт з полотном, вам потрібно створити екземпляр `QgsLayerTreeMapCanvasBridge`, як у прикладі нижче:

```
bridge = QgsLayerTreeMapCanvasBridge( \
         QgsProject.instance().layerTreeRoot(), canvas)
# Now you can safely load your project and see it in the
canvas
project.read('testdata/my_new_qgis_project.qgs')
```

7.7. Вирішення проблеми поганих шляхів

Може статися так, що завантажені в проєкт шари переміщуються в інше місце. Коли проєкт завантажується знову, усі шляхи шару розриваються. Клас `QgsPathResolver` допоможе переписати шлях шарів у проєкті.

Цей `setPathPreprocessor()` метод дає змогу встановлювати спеціальну функцію попередньої обробки шляхів для маніпулювання шляхами та джерелами даних перед їхнім розв’язанням у посиланнях на файли або джерел шарів.

Функція процесора повинна приймати єдиний рядковий аргумент (що являє собою вихідний шлях до файла або джерело даних) і повертати оброблену версію цього шляху. Функція препроцесора шляху викликається **перед** будь-яким обробником поганого шару. Якщо встановлено декілька препроцесорів, вони викликатимуться послідовно відповідно до порядку, у якому вони були встановлені спочатку.

Деякі випадки використання:

1. Замінити застарілий шлях:

```
    def my_processor(path):
        return
path.replace('c:/Users/ClintBarton/Documents/Projects',
'x:/Projects/')

    QgsPathResolver.setPathPreprocessor(my_processor)
```

2. Замінити адресу хоста бази даних на нову:

```
    def my_processor(path):
        return path.replace('host=10.1.1.115',
'host=10.1.1.116')

QgsPathResolver.setPathPreprocessor(my_processor)
```

3. Замінити збережені облікові дані бази даних на нові:

```
    def my_processor(path):
        path= path.replace(«user='gis_team'«,
«user='team_awesome'«)
        path = path.replace(«password='cats'«,
«password='g7as!m*'«)
        return path

QgsPathResolver.setPathPreprocessor(my_processor)
```

Подібним чином `setPathWriter()` – доступний метод для функції запису шляху.

Приклад заміни шляху на змінну:

```
def my_processor(path):
return
path.replace('c:/Users/ClintBarton/Documents/Projects',
'$projectdir$')

QgsPathResolver.setPathWriter(my_processor)
```

Обидва методи повертають `id`, який можна використати для видалення попереднього процесора або запису, який вони додали. Дивіться `removePathPreprocessor()` і `removePathWriter()`.

7.8. Використання `Flag` для прискорення

У деяких випадках, коли може не знадобитися використовувати повністю функціональний проект, а ви хочете отримати до нього доступ лише з певної причини, позначки можуть бути корисними. Повний список прапорців доступний у розділі `ProjectReadFlag`. Кілька прапорців можна додати разом.

Наприклад, якщо ми не дбаємо про фактичні шари та дані і лише хочемо отримати доступ до проєкту (наприклад, для налаштувань макета або 3D-виду), можена використати прапорець, щоб обійти `DontResolveLayers` етап перевірки даних і запобігти появі діалогового вікна поганого шару. Можна вчинити так:

```
readflags = Qgis.ProjectReadFlags()
readflags |= Qgis.ProjectReadFlag.DontResolveLayers
project = QgsProject()
project.read('C:/Users/ClintBarton/Documents/Projects/mysweetproject.qgs', readflags)
```

Щоб додати більше `flags`, необхідно використати оператор Python Побітове АБО ( ).

7.9. Завантаження шарів

7.9.1. Векторні шари

Для того щоби створити та додати екземпляр векторного шару до проєкту, вкажіть ідентифікатор джерела даних шару, ім’я шару та ім’я постачальника:

```
# get the path to the shapefile e.g. /home/project/data/ports.shp
path_to_airports_layer = «testdata/airports.shp»

# The format is:
# vlayer = QgsVectorLayer(data_source, layer_name, provider_name)

vlayer = QgsVectorLayer(path_to_airports_layer, «Airports layer», «ogr»)
```

```
 if not vlayer.isValid():
     print(«Layer failed to load!»)
else:
    QgsProject.instance().addMapLayer(vlayer)
```

Ідентифікатор джерела даних – це рядок, специфічний для кожного провайдера векторних даних. Назва шару використовується у віджеті списку шарів. Слід перевіряти успішно завершилося завантаження шару чи ні. У разі помилок повертається неправильний об'єкт.

Для векторного шару геопакетів:

```
# get the path to a geopackage  e.g.
/usr/share/qgis/resources/data/world_map.gpkg
path_to_gpkg = os.path.join(QgsApplication.pkgDataPath(),
«resources», «data», «world_map.gpkg»)
# append the layername part
gpkg_countries_layer = path_to_gpkg +
«|layername=countries»
# e.g. gpkg_places_layer =
«/usr/share/qgis/resources/data/world_map.gpkg|layername=co
untries»
vlayer = QgsVectorLayer(gpkg_countries_layer, «Countries
layer», «ogr»)
if not vlayer.isValid():
print(«Layer failed to load!»)
else:
QgsProject.instance().addMapLayer(vlayer)
```

Найшвидший спосіб відкрити та відобразити векторний шар у QGIS – це `addVectorLayer()` метод `QgisInterface`:

```
vlayer = iface.addVectorLayer(path_to_airports_layer, «Airports layer», «ogr»)
                    if not vlayer:
                      print(«Layer failed to load!»)
```

Таким чином буде створений новий шар і додантй до поточного проєкту QGIS (відобразиться у списку шарів) за один крок. Функція повертає екземпляр шару або `None` якщо шар не вдалося завантажити.

Далі показано, як отримати доступ до різних джерел даних, використовуючи векторні дані провайдерів:

– бібліотека GDAL (Shapefile та багато інших форматів файлів) – джерелом даних є шлях до файла:

– для Shapefile:

```
vlayer = QgsVectorLayer(«testdata/airports.shp»,
«layer_name_you_like», «ogr»)
QgsProject.instance().addMapLayer(vlayer)
```

– для dxf (зверніть увагу на внутрішні параметри в *uri* джерела даних):

```
uri =
«testdata/sample.dxf|layername=entities|geometrytype=Poly
gon»
vlayer = QgsVectorLayer(uri, «layer_name_you_like»,
«ogr»)
QgsProject.instance().addMapLayer(vlayer)
```

– база даних PostGIS – це рядок з усією інформацією, потрібною для [створення] під'єднання до бази даних PostgreSQL.

`QgsDataSourceUri` клас може створити цей рядок для вас. Зауважте, що QGIS має бути скомпільований з підтримкою Postgres, інакше цей провайдер буде недоступний:

```
uri = QgsDataSourceUri()
# set host name, port, database name, username and
password
uri.setConnection(«localhost», «5432», «dbname», «johny»,
«xxx»)
# set database schema, table name, geometry column and
optionally
# subset (WHERE clause)
uri.setDataSource(«public», «roads», «the_geom», «cityid
= 2643», «primary_key_field»)

vlayer = QgsVectorLayer(uri.uri(False), «layer name you like», «postgres»)
```

Аргумент *False*, переданий у *uri.uri(False)* запобігає розширенню параметрів конфігурації автентифікації, якщо ви не використовуєте жодної конфігурації автентифікації, цей аргумент не має значення.

- CSV або інші текстові файли з роздільниками – щоб відкрити файл із крапкою з комою як роздільник, із полем «x» для координати X і полем «y» для координати Y, скористайтеся приблизно таким:

```
uri
«file://{}/testdata/delimited_xy.csv?delimiter={}&xField=
{}&yField={}».format(os.getcwd(), «;», «x», «y»)
vlayer = QgsVectorLayer(uri, «layer name you like»,
«delimitedtext»)
QgsProject.instance().addMapLayer(vlayer)
```

Рядок постачальника структурований як URL-адреса, тому шлях має мати префікс ***file://***. Крім того, він дає змогу використовувати геометрії у форматі WKT (добре відомий текст) як альтернативу полям ***x*** і ***y*** та вказувати систему відліку координат. Наприклад:

```
uri =
«file:///some/path/file.csv?delimiter={}&crs=epsg:4723&wk
tField={}».format(«;», «shape»)
```

GPX – постачальник даних «gpx» зчитує треки, маршрути та шляхові точки з файлів gpx. Для того щоби відкрити файл, у URL-адресі потрібно вказати тип (доріжка/маршрут/маршрутна точка):

```
uri = «testdata/layers.gpx?type=track»
vlayer = QgsVectorLayer(uri, «layer name you like», «gpx»)
QgsProject.instance().addMapLayer(vlayer)
```

База даних SpatiaLite – подібно до баз даних PostGIS, `QgsDataSourceUri` може використовуватися для створення ідентифікатора джерела даних:

```
uri = QgsDataSourceUri()
uri.setDatabase('/home/martin/test-2.3.sqlite')
schema = ''
table = 'Towns'
geom_column = 'Geometry'
uri.setDataSource(schema, table, geom_column)
```

```
display_name = 'Towns'
vlayer = QgsVectorLayer(uri.uri(), display_name,
'spatialite')
QgsProject.instance().addMapLayer(vlayer)
```

Геометрія на базі MySQL WKB через GDAL – джерелом даних є рядок під'єднання до таблиці:

```
uri =
«MySQL:dbname,host=localhost,port=3306,user=root,password
=xxx|layername=my_table»
vlayer = QgsVectorLayer( uri, «my table», «ogr» )
QgsProject.instance().addMapLayer(vlayer)
```

З'єднання WFS: з'єднання визначається за допомогою URI та за допомогою `WFS` постачальника:

```
uri = «https://demo.mapserver.org/cgi-
bin/wfs?service=WFS&version=2.0.0
&request=GetFeature&typename=ms:cities«
vlayer = QgsVectorLayer(uri, «my wfs layer», «WFS»)
```

URI можна створити за допомогою стандартної `urllib` бібліотеки:

```
import urllib

3params = {
    'service': 'WFS',
    'version': '2.0.0',
    'request': 'GetFeature',
    'typename': 'ms:cities',
    'srsname': «EPSG:4326»
}
uri2 = 'https://demo.mapserver.org/cgi-bin/wfs?' +
urllib.parse.unquote(urllib.parse.urlencode(params))
```

Ви можете змінити джерело даних наявного шару, викликавши `setDataSource()` екземпляр `QgsVectorLayer`, як у наступному прикладі:

```
uri = «https://demo.mapserver.org/cgi-
bin/wfs?service=WFS&version=2.0.0&request=GetFeature&type
name=ms:cities»
provider_options = QgsDataProvider.ProviderOptions()
```

```
# Use project's transform context
provider_options.transformContext =
QgsProject.instance().transformContext()
vlayer.setDataSource(uri, «layer name you like», «WFS»,
provider_options)

del(vlayer)
```

7.9.2. Растрові шари

Для доступу до растрових файлів використовують бібліотеку GDAL, якмй підтримує широкий спектр форматів файлів. Якщо виникають проблеми з відкриттям деяких файлів, перевірте, чи підтримує ваш GDAL певний формат (не всі формати доступні за замовчуванням). Щоби завантажити растр з файла, вкажіть його ім'я та відображуване ім'я:

```
# get the path to a tif file  e.g.
/home/project/data/srtm.tif
path_to_tif = «qgis-
projects/python_cookbook/data/srtm.tif»
rlayer = QgsRasterLayer(path_to_tif, «SRTM layer name»)
if not rlayer.isValid():
    print(«Layer failed to load!»)
```

Для того щоби завантажити растр з геопакета, треба:

```
# get the path to a geopackage  e.g.
/home/project/data/data.gpkg
path_to_gpkg = os.path.join(os.getcwd(), «testdata»,
«sublayers.gpkg»)
# gpkg_raster_layer =
«GPKG:/home/project/data/data.gpkg:srtm»
gpkg_raster_layer = «GPKG:» + path_to_gpkg + «:srtm»

rlayer = QgsRasterLayer(gpkg_raster_layer, «layer name
you like», «gdal»)

if not rlayer.isValid():
    print(«Layer failed to load!»)
```

Подібно до векторних шарів растрові шари можна завантажити за допомогою функції addRasterLayer об'єкта `QgisInterface`:

```
iface.addRasterLayer(path_to_tif, «layer name you like»)
```

Це створює новий шар і додає його до поточного проєкту (відображаючи [його] у списку шарів) за один крок.

Щоб завантажити растр PostGIS

Растри PostGIS, подібні до векторів PostGIS, можна додати до проєкту за допомогою рядка URI. Ефективним є зберігати багаторазовий словник рядків для параметрів під'єднання до бази даних. Це полегшує редагування словника для відповідного під'єднання. Потім словник кодують в URI за допомогою об'єкта метаданих постачальника «postgresraster». Після цього растр можна додати до проєкту.

```
uri_config = {
     # database parameters
     'dbname':'gis_db',      # The PostgreSQL database to
connect to.
     'host':'localhost',     # The host IP address or
localhost.
     'port':'5432',          # The port to connect on.
     'sslmode':QgsDataSourceUri.SslDisable, # SslAllow,
SslPrefer, SslRequire, SslVerifyCa, SslVerifyFull
     # user and password are not needed if stored in the
authcfg or service
     'authcfg':'QconfigId',  # The QGIS athentication
database ID holding connection details.
     'service': None,          # The PostgreSQL service to
be used for connection to the database.
    'username':None,         # The PostgreSQL user name.
    'password':None,         # The PostgreSQL password for
the user.
    # table and raster column details
    'schema':'public',       # The database schema that the
table is located in.
    'table':'my_rasters',    # The database table to be
loaded.
    'geometrycolumn':'rast',# raster column in PostGIS
table
    'sql':None,              # An SQL WHERE clause. It
should be placed at the end of the string.
    'key':None,              # A key column from the table.
    'srid':None,             # A string designating the SRID
of the coordinate reference system.
    'estimatedmetadata':'False', # A boolean value telling
if the metadata is estimated.
```

```
    'type':None,            # A WKT string designating the
WKB Type.
    'selectatid':None,      # Set to True to disable
selection by feature ID.
    'options':None,         # other PostgreSQL connection
options not in this list.
    'enableTime': None,
    'temporalDefaultTime': None,
    'temporalFieldIndex': None,
    'mode':'2',             # GDAL 'mode' parameter, 2
unions raster tiles, 1 adds tiles separately (may require
user input)
}
# remove any NULL parameters
uri_config = {key:val for key, val in uri_config.items() if
val is not None}
# get the metadata for the raster provider and configure
the URI
md =
QgsProviderRegistry.instance().providerMetadata('postgresra
ster')
uri = QgsDataSourceUri(md.encodeUri(uri_config))

# the raster can then be loaded into the project
rlayer = iface.addRasterLayer(uri.uri(False), «raster layer
name», «postgresraster»)
```

Растрові шари також можна створювати за допомогою служби WCS:

```
layer_name = 'modis'
url = «https://demo.mapserver.org/cgi-
bin/wcs?identifier={}».format(layer_name)
rlayer = QgsRasterLayer(uri, 'my wcs layer', 'wcs')
```

Ось опис параметрів, які може містити WCS URI:

WCS URI складається з пар **ключ=значення,** розділених символом `&`. Це той самий формат, що й рядок запиту в URL-адресі, закодований так само. `QgsDataSourceUri` слід використовувати для побудови URI, щоб переконатися, що спеціальні символи закодовані належним чином.

- **url** (обов’язково): URL-адреса сервера WCS. Не використовуйте VERSION в URL-адресі, оскільки кожна версія WCS використовує різні назви параметрів для версії **GetCapabilities**, дивіться param version;
- **ідентифікатор** (обов’язково): назва покриття;
- **час** (необов’язково): позиція або період часу (beginPosition/endPosition[/timeResolution]);
- **формат** (необов’язково): назва підтримуваного формату. За замовчуванням – це перший підтримуваний формат із назвою tif або перший підтримуваний формат;
- **crs** (необов’язково): CRS у формі AUTHORITY:ID, наприклад, EPSG:4326. За замовчуванням EPSG:4326, якщо підтримується, або перша підтримувана CRS;
- **ім'я користувача** (необов'язково): ім'я користувача для базової автентифікації;
- **пароль** (необов'язково): пароль для базової автентифікації;
- **IgnoreGetMapUrl** (необов’язково, хак): якщо вказано (встановлено значення 1), ігнорувати URL-адресу GetCoverage, рекламовану GetCapabilities. Може знадобитися, якщо сервер налаштовано неправильно;
- **InvertAxisOrientation** (необов’язковий, хак): якщо вказано (встановлено значення 1), змініть вісь у запиті GetCoverage. Може знадобитися для географічної CRS, якщо сервер використовує неправильний порядок осей;
- **IgnoreAxisOrientation** (необов’язковий, хак): якщо вказано (встановлено на 1), не інвертуйте орієнтацію осі відповідно до стандарту WCS для географічної CRS;
- **cache** (необов’язково): контроль завантаження кешу, як описано в QNetworkRequest::CacheLoadControl, але запит повторно надсилається як PreferCache, якщо не вдається виконати AlwaysCache. Дозволені значення: AlwaysCache, PreferCache, PreferNetwork, AlwaysNetwork. Типовим є AlwaysCache.

Крім того, ви можете завантажити растровий шар із сервера WMS. Однак наразі неможливо отримати доступ до відповіді GetCapabilities через API – ви повинні знати, яких шарів ви потребуєте:

```
urlWithParams =
«crs=EPSG:4326&format=image/png&layers=continents&styles&
url=https://demo.mapserver.org/cgi-bin/wms»
rlayer = QgsRasterLayer(urlWithParams, 'some layer name',
'wms')
if not rlayer.isValid():
  print(«Layer failed to load!»)
```

7.9.3. Екземпляр QgsProject

Якщо ви бажаєте використовувати відкриті шари для візуалізації, не забудьте додати їх до екземпляра `QgsProject`. Екземпляр `QgsProject` бере на себе права власності на шари, і пізніше до них можна отримати доступ з будь-якої частини програми за їхнім унікальним ідентифікатором. Коли шар видаляється з проєкту, екземпляр також видаляється. Шари можуть бути видалені користувачем в інтерфейсі QGIS або через Python за допомогою методу `removeMapLayer()`.

Додавання шару до поточного проєкту здійснюється за допомогою `addMapLayer()` методу:

```
QgsProject.instance().addMapLayer(rlayer)
Щоб додати шар в абсолютну позицію:
# first add the layer without showing it
QgsProject.instance().addMapLayer(rlayer, False)
# obtain the layer tree of the top-level group in the
project
layerTree = iface.layerTreeCanvasBridge().rootGroup()
# the position is a number starting from 0, with -1 an
alias for the end
layerTree.insertChildNode(-1, QgsLayerTreeLayer(rlayer))
```

Якщо потрібно видалити шар, використовуйте `removeMapLayer()` метод:

```
# QgsProject.instance().removeMapLayer(layer_id)
QgsProject.instance().removeMapLayer(rlayer.id())
```

У наведеному коді передається ідентифікатор шару (його можна отримати, викликавши метод `id()` шару), але ви також можете передати шару сам об’єкт.

Для списку завантажених шарів і ідентифікаторів шарів скористайтеся таким `mapLayers()` методом:

```
QgsProject.instance().mapLayers()
```

7.10. Робота з растровими шарами

Фрагменти коду на цій сторінці потребують наступного імпорту, якщо ви перебуваєте поза консоллю pyqgis:

```
from qgis.core import (
    QgsRasterLayer,
    QgsProject,
    QgsPointXY,
    QgsRaster,
    QgsRasterShader,
    QgsColorRampShader,
    QgsSingleBandPseudoColorRenderer,
    QgsSingleBandColorDataRenderer,
    QgsSingleBandGrayRenderer,
)

from qgis.PyQt.QtGui import (
    QColor,
)
```

7.10.1. Інформація про шар

Растровий шар складається з однієї або кількох растрових смуг, які називаються одноканальними та багатоканальними растрами. Одна смуга представляє матрицю значень. Кольорове зображення (наприклад, аерофотознімок) – це растр, що складається з червоних, синіх і зелених смуг. Одноканальні растри зазвичай представляють або безперервні змінні (наприклад, висоту), або дискретні змінні (наприклад, землекористування). У деяких випадках растровий шар постачається з палітрою, а растрові значення стосуються кольорів, що зберігаються в палітрі.

Наступний код означає, що *rlayer* – це `QgsRasterLayer` об’єкт.

```
rlayer = QgsProject.instance().mapLayersByName('srtm')[0]
# get the resolution of the raster in layer unit
print(rlayer.width(), rlayer.height())
```

```
919 619
# get the extent of the layer as QgsRectangle
print(rlayer.extent())
<QgsRectangle: 20.06856808199999875 -
34.27001076999999896, 20.83945284300000012 -
33.75077500700000144>
# get the extent of the layer as Strings
print(rlayer.extent().toString())
20.0685680819999988,-34.2700107699999990 :
20.8394528430000001,-33.7507750070000014
# get the raster type: 0 = GrayOrUndefined (single band),
1 = Palette (single band), 2 = Multiband
print(rlayer.rasterType())
0
# get the total band count of the raster
print(rlayer.bandCount())
1
# get the first band name of the raster
print(rlayer.bandName(1))
Band 1: Height
# get all the available metadata as a QgsLayerMetadata
object
print(rlayer.metadata())
<qgis._core.QgsLayerMetadata object at 0x13711d558>
```

7.10.2. Рендеринг

Коли растровий шар завантажується, він отримує стандартний рендер на основі свого типу. Його можна змінити або у властивостях шару, або програмно.

Для того щоби запитати поточний рендер, слід виконати:

```
print(rlayer.renderer())
<qgis._core.QgsSingleBandGrayRenderer object at
0x7f471c1da8a0>
print(rlayer.renderer().type())
singlebandgray
```

Щоб установити рендер, скористайтеся `setRenderer()` методом `QgsRasterLayer`. Розрізняють кілька класів рендера (похідних від `QgsRasterRenderer`):

- `QgsHillshadeRenderer;`

- QgsMultiBandColorRenderer;
- QgsPalettedRasterRenderer;
- QgsRasterContourRenderer;
- QgsSingleBandColorDataRenderer;
- QgsSingleBandGrayRenderer;
- QgsSingleBandPseudoColorRenderer.

Односмугові растрові шари можна намалювати сірими кольорами (низькі значення = чорний, високі значення = білий) або за допомогою псевдокольорового алгоритму, який призначає кольори значенням. Одноканальні растри з палітрою також можна малювати за допомогою палітри. Багатосмугові шари зазвичай створюються шляхом відображення смуг у кольори RGB. Інша можливість – використовувати для малювання лише одну смугу.

7.10.3. Одноканальні растри

Припустімо, нам потрібен односмуговий растровий шар візуалізації з кольорами від зеленого до жовтого (що відповідає значенням пікселів від 0 до 255). На першому етапі ми підготуємо об'єкт `QgsRasterShader` і налаштуємо його функцію шейдера:

```
fcn = QgsColorRampShader()
fcn.setColorRampType(QgsColorRampShader.Interpolated)
lst = [ QgsColorRampShader.ColorRampItem(0,
QColor(0,255,0)),
      QgsColorRampShader.ColorRampItem(255,
QColor(255,255,0)) ]
fcn.setColorRampItemList(lst)
shader = QgsRasterShader()
shader.setRasterShaderFunction(fcn)
```

Шейдер відображає кольори відповідно до його карти кольорів. Карта кольорів надається як список значень пікселів із пов'язаними кольорами. Розрізняють три режими інтерполяції:

- лінійний (`Interpolated`): колір лінійно інтерполюється із записів карти кольорів над і під значенням пікселя;
- дискретний (`Discrete`): колір береться з найближчого запису карти кольорів з рівним або вищим значенням;

- точний (`Exact`): колір не інтерполюється, будуть намальовані лише пікселі зі значеннями, що дорівнюють записам карти кольорів.

На другому кроці пов'яжемо цей шейдер із растровим шаром:

```
renderer =
QgsSingleBandPseudoColorRenderer(rlayer.dataProvider(),
1, shader)
rlayer.setRenderer(renderer)
```

Число *1* в коді вище є номером смуги (растрові смуги індексуються з одиниці). Нарешті слід скористатися `triggerRepaint()` методом, щоб побачити результати:

```
rlayer.triggerRepaint()
```

7.10.4. Багатоканальні растри

За замовчуванням QGIS відображає три смуги – червону, зелену та синю для створення кольорового зображення (це `MultiBandColor` стиль малювання). У деяких випадках ви можете змінити ці параметри. Наступний код замінює червону смугу (1) і зелену смугу (2):

```
rlayer_multi =
QgsProject.instance().mapLayersByName('multiband')[0]
rlayer_multi.renderer().setGreenBand(1)
rlayer_multi.renderer().setRedBand(2)
```

Якщо для візуалізації растру необхідна лише одна смуга, можна обрати односмуговий малюнок, або рівні сірого, або псевдокольори.

Потрібно скористатися `triggerRepaint()`, щоб оновити карту та побачити результат:

```
rlayer_multi.triggerRepaint()
```

7.10.5. Присвоєння значень

Растрові значення можна запитувати за допомогою `sample()`методу класу `QgsRasterDataProvider`. Треба вказати а `QgsPointXY` та номер смуги растрового шару, який ви хочете запитати. Метод повертає кортеж із значенням і `True` або `False` залежно від результатів:

```
val, res =
rlayer.dataProvider().sample(QgsPointXY(20.50, -34), 1)
```

Іншим методом запиту растрових значень є використання `identify()` методу, який повертає `QgsRasterIdentifyResult` об'єкт.

```
ident = rlayer.dataProvider().identify(QgsPointXY(20.5, -
34), QgsRaster.IdentifyFormatValue)

if ident.isValid():
  print(ident.results())
{1: 323.0}
```

У цьому випадку `results()` метод повертає словник з індексами смуг як ключі та значення смуг як значення, наприклад: {1: 323.0}

7.10.6. Редагування растрових даних

За допомогою класу можна створити растровий шар `QgsRasterBlock`. Наприклад, щоб створити растровий блок 2×2 з одним байтом на піксель, потрібно:

```
block = QgsRasterBlock(Qgis.Byte, 2, 2)
block.setData(b'\xaa\xbb\xcc\xdd')
```

Растрові пікселі можна перезаписати завдяки методу **`writeBlock()`**, зокрема, Щоб перезаписати растрові дані в позиції 0,0 блоком 2×2, слід виконати:

```
provider = rlayer.dataProvider()
provider.setEditable(True)
provider.writeBlock(block, 1, 0, 0)
provider.setEditable(False)
```

7.11. Робота з векторними шарами

Фрагменти коду на цій сторінці потребують імпорту, якщо ви перебуваєте поза консоллю pyqgis:

```
from qgis.core import (
      QgsApplication,
      QgsDataSourceUri,
      QgsCategorizedSymbolRenderer,
      QgsClassificationRange,
      QgsPointXY,
      QgsProject,
      QgsExpression,
      QgsField,
     QgsFields,
     QgsFeature,
     QgsFeatureRequest,
     QgsFeatureRenderer,
     QgsGeometry,
     QgsGraduatedSymbolRenderer,
     QgsMarkerSymbol,
     QgsMessageLog,
     QgsRectangle,
     QgsRendererCategory,
     QgsRendererRange,
     QgsSymbol,
     QgsVectorDataProvider,
     QgsVectorLayer,
     QgsVectorFileWriter,
     QgsWkbTypes,
     QgsSpatialIndex,
     QgsVectorLayerUtils
   )

from qgis.core.additions.edit import edit

from qgis.PyQt.QtGui import (
       QColor,
   )
```

У цьому пункті описано різні дії, які можна виконувати із векторними шарами, робота яких ґрунтується на методах класу `QgsVectorLayer`.

7.11.1. Отримання інформації про атрибути

Ви можете отримати інформацію про поля, пов'язані з векторним шаром, викликавши `fields()` об'єкт `QgsVectorLayer`:

```
vlayer = QgsVectorLayer(«testdata/airports.shp»,
«airports», «ogr»)
for field in vlayer.fields():
    print(field.name(), field.typeName())
ID Integer64
fk_region Integer64
ELEV Real
NAME String
USE String
```

За допомогою методів класу `displayField()` отримують інформацію про поле та шаблон, які використовуються на вкладці `Властивості дисплея`.`mapTipTemplate()``QgsVectorLayer`.

Коли ви завантажуєте векторний шар, поле завжди обирається QGIS як , тоді як за замовчуванням поле порожнє. За допомогою цих методів ви можете легко отримати обидва: `Display NameHTML Map Tip`

```
vlayer = QgsVectorLayer(«testdata/airports.shp»,
«airports», «ogr»)
print(vlayer.displayField())
NAME
```

7.11.2. Вибір властивостей

У робочому столі QGIS об’єкти можна обирати різними способами: користувач може клацнути об’єкт, намалювати прямокутник на полотні карти або скористатися фільтром виразів. Вибрані функції зазвичай виділяють іншим кольором (за замовчуванням – жовтим), щоб привернути увагу користувача до вибору.

Іноді може бути корисним програмний вибір функцій або зміна кольору за замовчуванням.

Для вибору всіх функцій `selectAll()` можна використовувати метод:

```
# Get the active layer (must be a vector layer)
layer = iface.activeLayer()
layer.selectAll()
```

Щоб обрати за допомогою виразу, використовуйте `selectByExpression()` метод:

```
# Assumes that the active layer is points.shp file from the
QGIS test suite
# (Class (string) and Heading (number) are attributes in
points.shp)
layer = iface.activeLayer()
layer.selectByExpression('«Class»=\'B52\' and «Heading» > 10
and «Heading» <70', QgsVectorLayer.SetSelection)
```

Для того щоби змінити колір виділення, ви можете скористатися `setSelectionColor()` методом `QgsMapCanvas`, як показано в наступному прикладі:

```
iface.mapCanvas().setSelectionColor( QColor(«red») )
```

Щоб додати об’єкти до вибраного списку об’єктів для заданого шару, можна викликати `select()` передачу до нього списку ідентифікаторів об’єктів:

```
selected_fid = []

# Get the first feature id from the layer
feature = next(layer.getFeatures())
if feature:
    selected_fid.append(feature.id())

# Add that features to the selected list
layer.select(selected_fid)
```

Щоб очистити вибір:

```
layer.removeSelection()
```

7.11.3. Доступ до атрибутів

На атрибути можна посилатися за їхніми іменами:

```
print(feature['name'])
First feature
```

Крім того, на атрибути можна посилатися за індексом. Це трохи швидше, ніж використання імені. Наприклад, щоб отримати другий атрибут:

```
print(feature[1])
First feature
```

7.11.4. Перегляд вибраних функцій

Якщо вам потрібні лише вибрані об'єкти, ви можете скористатися `selectedFeatures()` методом з векторного шару:

```
selection = layer.selectedFeatures()
for feature in selection:
    # do whatever you need with the feature
    pass
```

7.11.5. Перегляд підмножини функцій

Якщо ви хочете повторити задану підмножину об'єктів у шарі, наприклад ті, що знаходяться в заданій області, ви повинні додати об'єкт `QgsFeatureRequest` до `getFeatures()` виклику. Ось приклад:

```
1areaOfInterest = QgsRectangle(450290,400520,
450750,400780)
2
3request =
QgsFeatureRequest().setFilterRect(areaOfInterest)
4
5for feature in layer.getFeatures(request):
6    # do whatever you need with the feature
7    pass
```

Заради швидкості перетин часто виконують лише за допомогою обмежувальної рамки об'єкта. Однак є прапорець `ExactIntersect`, який гарантує, що будуть повернуті лише пересічні функції:

```
request = QgsFeatureRequest().setFilterRect(areaOfInterest) \
.setFlags(QgsFeatureRequest.ExactIntersect)
```

За допомогою `setLimit()` можна обмежити кількість запитуваних функцій. Ось приклад:

```
request = QgsFeatureRequest()
request.setLimit(2)
for feature in layer.getFeatures(request):
print(feature)
<qgis._core.QgsFeature object at 0x7f9b78590948>
<qgis._core.QgsFeature object at 0x7faef5881670>
```

Якщо потрібен фільтр на основі атрибутів замість (або на додаток) до просторового, як показано в наведених прикладах, ви можете створити об'єкт `QgsExpression`і передати його конструктору `QgsFeatureRequest`. Ось приклад:

```
# The expression will filter the features where the
field «location_name»
# contains the word «Lake» (case insensitive)
exp = QgsExpression('location_name ILIKE \'%Lake%\'')
request = QgsFeatureRequest(exp)
```

See Вирази, фільтрація та обчислення значень для details про syntax supported by `QgsExpression`.

Запит можна використовувати для визначення даних, отриманих для кожної функції, тому ітератор повертає всі функції, але повертає часткові дані для кожної з них.

```
1# Only return selected fields to increase the «speed» of
the request
2request.setSubsetOfAttributes([0,2])
3
4# More user friendly version

5request.setSubsetOfAttributes(['name','id'],layer.fields()
)
6
7# Don't return geometry objects to increase the «speed»
of the request
8request.setFlags(QgsFeatureRequest.NoGeometry)
9
10# Fetch only the feature with id 45
11request.setFilterFid(45)
12
13# The options may be chained
```

```
14request.setFilterRect(areaOfInterest).setFlags(QgsFeature
Request.NoGeometry).setFilterFid(45).setSubsetOfAttributes(
[0,2])
```

7.11.6. Редагування векторних шарів

Більшість постачальників векторних даних підтримують редагування даних шару. Іноді вони підтримують лише підмножину можливих дій редагування. Скористайтеся цією `capabilities()` функцією, щоб дізнатися, який набір функцій підтримується.

```
caps = layer.dataProvider().capabilities()
# Check if a particular capability is supported:
if caps & QgsVectorDataProvider.DeleteFeatures:
    print('The layer supports DeleteFeatures')
The layer supports DeleteFeatures
```

Щоб отримати список усіх доступних можливостей, зверніться до `API Documentation of QgsVectorDataProvider`

Для того щоби надрукувати текстовий опис можливостей шару у списку, розділеному комами, ви можете скористатися `capabilitiesString()` за таким прикладом:

```
caps_string = layer.dataProvider().capabilitiesString()
# Print:
# 'Add Features, Delete Features, Change Attribute
Values, Add Attributes,
# Delete Attributes, Rename Attributes, Fast Access to
Features at ID,
# Presimplify Geometries, Presimplify Geometries with
Validity Check,
# Transactions, Curved Geometries'
```

Використовуючи будь-який із наведених далі методів для редагування векторного шару, зміни вносять безпосередньо до основного сховища даних (файла, бази даних тощо).

```
# If caching is enabled, a simple canvas refresh might
not be sufficient
# to trigger a redraw and you must clear the cached
image for the layer
if iface.mapCanvas().isCachingEnabled():
```

```
        layer.triggerRepaint()
    else:
        iface.mapCanvas().refresh()
```

7.11.8. Додавання об'єктів

Створіть кілька `QgsFeature` екземплярів і передайте їхній список методу провайдера `addFeatures()`. Він поверне два значення: результат (`True` або `False`) і список доданих функцій (їхній ідентифікатор встановлюється сховищем даних).

Для того щоби налаштувати атрибути функції, ви можете або ініціалізувати функцію, передаючи об’єкт `QgsFields` (це можна отримати за допомогою `fields()` методу векторного шару), або викликати `initAttributes()` передачу кількості полів, які ви хочете додати.

```
if caps & QgsVectorDataProvider.AddFeatures:
    feat = QgsFeature(layer.fields())
    feat.setAttributes([0, 'hello'])
    # Or set a single attribute by key or by index:
    feat.setAttribute('name', 'hello')
    feat.setAttribute(0, 'hello')

feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(123,
456)))
    (res, outFeats) =
layer.dataProvider().addFeatures([feat])
```

7.11.9. Видалення об'єктів

Для того щоби видалити деякі функції, лише надайте список їхніх ідентифікаторів.

```
if caps & QgsVectorDataProvider.DeleteFeatures:
res = layer.dataProvider().deleteFeatures([5, 10])
```

7.11.10. Зміна об'єктів

Можна або змінити геометрію об'єкта, або змінити деякі атрибути. У наступному прикладі спочатку змінюються значення атрибутів з індексами 0 і 1, а потім змінюється геометрія об’єкта.

```
fid = 100   # ID of the feature we will modify

if caps & QgsVectorDataProvider.ChangeAttributeValues:
    attrs = { 0 : «hello», 1 : 123 }
    layer.dataProvider().changeAttributeValues({ fid :
attrs })

if caps & QgsVectorDataProvider.ChangeGeometries:
    geom = QgsGeometry.fromPointXY(QgsPointXY(111,222))
    layer.dataProvider().changeGeometryValues({ fid : geom
})
```

7.11.11. Редагування векторних шарів за допомогою буфера змін

Під час редагування векторів у програмі QGIS потрібно спочатку запустити режим редагування для певного шару, потім внести деякі зміни та, нарешті, зафіксувати (або скасувати) зміни. Усі зміни, які ви вносите, не будуть записані, доки ви їх не зафіксуєте – вони залишаються в буфері редагування шару в пам'яті. Цю функцію можна використовувати також програмно – це тільки ще один метод редагування векторного шару, який доповнює пряме використання постачальників даних. Використовуйте цей параметр, коли надаєте деякі інструменти графічного інтерфейсу для редагування векторного шару, оскільки це дасть користувачеві змогу вирішувати, чи фіксувати/скасовувати, і використовувати скасування/повторення. Після внесення змін усі зміни з буфера редагування зберігаються у постачальника даних.

Методи подібні до тих, які ми бачили в провайдері, але `QgsVectorLayer` замість цього вони викликаються в об'єкті.

Для того щоби ці методи працювали, шар має бути в режимі редагування. Для запуску режиму редагування скористайтеся `startEditing()` методом. Щоб припинити редагування, скористайтеся методами `commitChanges()` або `rollBack()`. Перший зафіксує всі ваші зміни до джерела даних, а другий відкине їх і взагалі не змінить джерело даних.

Щоб дізнатися, чи знаходиться шар у режимі редагування, скористайтеся `isEditable()` методом.

Ось кілька прикладів, які демонструють, як використовувати ці методи редагування.

```
from qgis.PyQt.QtCore import QVariant

feat1 = feat2 = QgsFeature(layer.fields())
fid = 99
feat1.setId(fid)

# add two features (QgsFeature instances)
layer.addFeatures([feat1,feat2])
# delete a feature with specified ID
layer.deleteFeature(fid)

# set new geometry (QgsGeometry instance) for a feature
geometry = QgsGeometry.fromWkt(«POINT(7 45)»)
layer.changeGeometry(fid, geometry)
# update an attribute with given field index (int) to a
given value
fieldIndex =1
value ='My new name'
layer.changeAttributeValue(fid, fieldIndex, value)

# add new field
layer.addAttribute(QgsField(«mytext», QVariant.String))
# remove a field
layer.deleteAttribute(fieldIndex)
```

Для того щоби скасування/повторення працювало належним чином, згадані виклики мають бути загорнуті в команди скасування. (Якщо ви не дбаєте про скасування/повторення і хочете, щоб зміни були збережені негайно, тоді вам буде легше працювати, редагуючи за допомогою постачальника даних.)

Ось як можна використовувати функцію скасування:

```
layer.beginEditCommand(«Feature triangulation»)

# ... call layer's editing methods ...

if problem_occurred:
  layer.destroyEditCommand()
  # ... tell the user that there was a problem
  # and return
# ... more editing ...
layer.endEditCommand()
```

Метод `beginEditCommand()` створить внутрішню «активну» команду та зафіксує подальші зміни у векторному шарі. Виклик `endEditCommand()` команди надсилається до стеку скасування, і користувач зможе скасувати/повторити її з графічного інтерфейсу. Якщо під час внесення змін щось пішло не так, `destroyEditCommand()` метод видалить команду та скасувати всі зміни, внесені, поки ця команда була активною.

Ви також можете використовувати -оператор, щоб обернути фіксацію та скасування у більш семантичний блок коду, як показано в прикладі `with edit(layer):`

```
with edit(layer):
  feat = next(layer.getFeatures())
  feat[0] = 5
  layer.updateFeature(feat)
```

Це буде автоматично викликане `commitChanges()` в кінці. Якщо станеться будь-який виняток, це призведе до `rollBack()` всіх змін. У разі виявлення проблеми всередині `commitChanges()` (коли метод повертає False) `QgsEditError`буде викликаний виняток.

7.11.12. Додавання та видалення полів

Для додавання полів (атрибутів) треба вказати список визначень полів. Для видалення полів досить надати список індексів полів.

```
from qgis.PyQt.QtCore import QVariant
if caps & QgsVectorDataProvider.AddAttributes:
    res = layer.dataProvider().addAttributes(
        [QgsField(«mytext», QVariant.String),
        QgsField(«myint», QVariant.Int)])

if caps & QgsVectorDataProvider.DeleteAttributes:
    res =
layer.dataProvider().deleteAttributes([0])
 # Alternate methods for removing fields
 # first create temporary fields to be removed
(f1-3)

layer.dataProvider().addAttributes([QgsField(«f1»,
```

```
QVariant.Int),QgsField(«f2»,QVariant.Int),QgsField
(«f3»,QVariant.Int)])
 layer.updateFields()
 count=layer.fields().count() # count of layer
fields
 ind_list=list((count-3, count-2)) # create list

 # remove a single field with an index
 layer.dataProvider().deleteAttributes([count-1])

# remove multiple fields with a list of indices
layer.dataProvider().deleteAttributes(ind_list)
```

Після додавання або видалення полів у постачальники даних потрібно поля шару оновити, оскільки зміни не поширюються автоматично.

7.11.13. Використання просторового індексу

Просторові індекси можуть значно підвищити продуктивність вашого коду, якщо вам потрібно надсилат часті запити до векторного шару. Уявіть, наприклад, що ви пишете алгоритм інтерполяції, і що для заданого місця вам потрібно знати 10 найближчих точок із шару точок, щоб використовувати ці точки для обчислення інтерпольованого значення. Без просторового індексу єдиний спосіб для QGIS знайти ці 10 точок – це обчислити відстань від кожної точки до вказаного місця, а потім порівняти ці відстані. Це потребує багато часу, особливо якщо запит треба повторити для кількох місць. Якщо для шару є просторовий індекс, операція відбуватиметься набагато ефективніше.

Уявить, що шар без просторового індексу як телефонна книга, в якій телефонні номери не впорядковані та не проіндексовані. Єдиний спосіб дознатися номер телефону певної людини – це переглядати книжку від початку, поки не натримати на нього.

Просторові індекси не створюються за замовчуванням для векторного шару QGIS, але їх можна легко створити:

- створити просторовий індекс за допомогою `QgsSpatialIndex` класу:

```
index = QgsSpatialIndex()
```

- add features to index – індекс бере `QgsFeature` об’єкт і додає його до внутрішньої структури даних. Ви можете створити

об’єкт вручну або використати його з попереднього виклику методу постачальника `getFeatures()`:

```
index.addFeature(feat)
```

- крім того, ви можете завантажити всі функції шару одночасно за допомогою масового завантаження:

```
index = QgsSpatialIndex(layer.getFeatures())
```

- щойно просторовий індекс буде заповнений деякими значеннями, ви зможете виконувати деякі запити:

```
# returns array of feature IDs of five nearest features
nearest = index.nearestNeighbor(QgsPointXY(25.4, 12.7), 5)
# returns array of IDs of features which intersect the rectangle
intersect = index.intersects(QgsRectangle(22.5, 15.3, 23.1, 17.2))
```

Також можна використовувати `QgsSpatialIndexKDBush` просторовий індекс. Цей індекс схожий на *стандартний* `QgsSpatialIndex`, але:

- підтримує **лише** функції однієї точки
- є **статичним** (жодні додаткові функції не можуть бути додані до індексу після побудови)
- мети досягнуто набагато **швидше!**
- Дає змогу досягти прямого отримання точок оригінальної функції, уникнувши додаткових запитів функції
- підтримує справжній пошук *на основі відстані*, тобто повертає всі точки в радіусі від точки пошуку

Клас `QgsVectorLayerUtils` містить кілька дуже корисних методів, які можна використовувати з векторними шарами.

Наприклад, `createFeature()` метод готує для `QgsFeature` додавання до векторного шару, зберігаючи всі можливі обмеження та значення за замовчуванням кожного поля:

```
vlayer = QgsVectorLayer(«testdata/airports.shp», «airports», «ogr»)
feat = QgsVectorLayerUtils.createFeature(vlayer)
```

Метод `getValues()` дає змогу швидко отримати значення поля або виразу:

```
vlayer = QgsVectorLayer(«testdata/airports.shp», «airports», «ogr»)
# select only the first feature to make the output shorter
```

```
vlayer.selectByIds([1])
val = QgsVectorLayerUtils.getValues(vlayer, «NAME»,
selectedOnly=True)
print(val)
(['AMBLER'], True)
```

7.11.14. Створення векторних шарів

Є кілька способів створення набору даних векторного шару:

- клас `QgsVectorFileWriter`: зручний клас для запису векторних файлів на диск, використовуючи або статичний виклик, `writeAsVectorFormatV3()`, який зберігає весь векторний шар, або створення екземпляра класу та виклики до `addFeature()`. Цей клас підтримує всі векторні формати, які підтримує GDAL (GeoPackage, Shapefile, GeoJSON, KML та інші);
- клас `QgsVectorLayer`: створює екземпляр постачальника даних, який інтерпретує наданий шлях (url) джерела даних для під'єднання та доступу до даних. Його можна використовувати для створення тимчасових шарів на основі пам'яті (`memory`) і під'єднаня до векторних наборів даних GDAL (`ogr`), баз даних (`postgres`, `spatialite`, `mysql`, `mssql`) тощо (`wfs`, `gpx`, `delimitedtext`...).

```
# SaveVectorOptions contains many settings for the writer
process
save_options = QgsVectorFileWriter.SaveVectorOptions()
transform_context =
QgsProject.instance().transformContext()
# Write to a GeoPackage (default)
error = QgsVectorFileWriter.writeAsVectorFormatV3(layer,
                        «testdata/my_new_file.gpkg»,
                        transform_context,
                        save_options)
if error[0] == QgsVectorFileWriter.NoError:
    print(«success!»)
else:
  print(error)
# Write to an ESRI Shapefile format dataset using UTF-8
text encoding
save_options = QgsVectorFileWriter.SaveVectorOptions()
save_options.driverName = «ESRI Shapefile»
```

```
save_options.fileEncoding = «UTF-8»
transform_context =
QgsProject.instance().transformContext()
error = QgsVectorFileWriter.writeAsVectorFormatV3(layer,
                        «testdata/my_new_shapefile»,
                        transform_context,
                        save_options)
if error[0] == QgsVectorFileWriter.NoError:
    print(«success again!»)
else:
  print(error)
# Write to an ESRI GDB file
save_options = QgsVectorFileWriter.SaveVectorOptions()
save_options.driverName = «FileGDB»
# if no geometry
save_options.overrideGeometryType = QgsWkbTypes.Unknown
save_options.actionOnExistingFile =
QgsVectorFileWriter.CreateOrOverwriteLayer
save_options.layerName = 'my_new_layer_name'
transform_context =
QgsProject.instance().transformContext()
gdb_path = «testdata/my_example.gdb»
error = QgsVectorFileWriter.writeAsVectorFormatV3(layer,
                                    gdb_path,
                                    transform_context,
                                    save_options)
if error[0] == QgsVectorFileWriter.NoError:
  print(«success!»)
else:
  print(error)
```

Ви також можете конвертувати поля, щоб зробити їх сумісними з різними форматами за допомогою `FieldValueConverter`. Наприклад, щоб перетворити типи змінних масиву (наприклад, у Postgres) у текстовий тип, можна виконате таке:

```
 LIST_FIELD_NAME = 'xxxx'

 class
ESRIValueConverter(QgsVectorFileWriter.FieldValueConverter):

   def __init__(self, layer, list_field):

QgsVectorFileWriter.FieldValueConverter.__init__(self)
```

```
        self.layer = layer
        self.list_field_idx =
    self.layer.fields().indexFromName(list_field)

      def convert(self, fieldIdxInLayer, value):
       if fieldIdxInLayer == self.list_field_idx:
          return
    QgsListFieldFormatter().representValue(layer=vlayer,
                            fieldIndex=self.list_field_idx,
                            config={},
                            cache=None,
                            value=value)
        else:
          return value

      def fieldDefinition(self, field):
        idx = self.layer.fields().indexFromName(field.name())
        if idx == self.list_field_idx:
         return QgsField(LIST_FIELD_NAME, QVariant.String)
        else:
          return self.layer.fields()[idx]

    converter = ESRIValueConverter(vlayer, LIST_FIELD_NAME)
    opts = QgsVectorFileWriter.SaveVectorOptions()
    opts.fieldValueConverter = converter
```

Можна також вказати CRS призначення – якщо дійсний екземпляр `QgsCoordinateReferenceSystem` передається як четвертий параметр, рівень перетворюється на цей CRS.

Для того щоби отримати дійсні назви драйверів, викличте `supportedFiltersAndFormats()` метод або зверніться до підтримуваних форматів OGR – вам слід передати значення в стовпці «Код» як назву драйвера.

За бажанням ви можете вказати, чи потрібно експортувати лише обрані функції, передавати додаткові параметри драйвера для створення або вказувати автору не створювати атрибути.

```
from qgis.PyQt.QtCore import QVariant

# define fields for feature attributes. A QgsFields
object is needed
```

```
fields = QgsFields()
fields.append(QgsField(«first», QVariant.Int))
fields.append(QgsField(«second», QVariant.String))

«««« create an instance of vector file writer, which will
create the vector file.
Arguments:
1. path to new file (will fail if exists already)
2. field map
3. geometry type - from WKBTYPE enum
4. layer's spatial reference (instance of
   QgsCoordinateReferenceSystem)
5. coordinate transform context
6. save options (driver name for the output file,
encoding etc.)
««««

crs = QgsProject.instance().crs()
transform_context =
QgsProject.instance().transformContext()
save_options = QgsVectorFileWriter.SaveVectorOptions()
save_options.driverName = «ESRI Shapefile»
save_options.fileEncoding = «UTF-8»

writer = QgsVectorFileWriter.create(
  «testdata/my_new_shapefile.shp»,
  fields,
  QgsWkbTypes.Point,
  crs,
  transform_context,
  save_options
)

if writer.hasError() != QgsVectorFileWriter.NoError:
    print(«Error when creating shapefile: «,
writer.errorMessage())

# add a feature
fet = QgsFeature()

fet.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(10,10)
))
fet.setAttributes([1, «text»])
```

```
writer.addFeature(fet)

# delete the writer to flush features to disk
del writer
```

Серед усіх постачальників даних, які підтримуються класом QgsVectorLayer, зосередьмося на рівнях на основі пам’яті. Постачальник пам’яті призначений для використання переважно розробниками плагінів або сторонніх програм. Він не зберігає даних на диску, що дає розробникам змогу використовувати його як швидкий сервер для деяких тимчасових шарів.

Провайдер підтримує рядкові та цілочисельні поля, а також поля з рухомою комою.

Провайдер пам'яті також підтримує просторове індексування, яке вмикається викликом функції провайдера createSpatialIndex(). Після створення просторового індексу ви зможете швидше переглядати об’єкти в менших регіонах (оскільки необов’язково обходити всі об’єкти, досить лише тих, що знаходяться у вказаному прямокутнику).

Постачальник пам’яті створюється шляхом передавання «memory»рядка провайдера конструктору QgsVectorLayer.

Конструктор також приймає URI, що визначає тип геометрії шару, один з-поміж: «Point», «LineString», «Polygon», «MultiPoint» або «MultiLineString»«MultiPolygon»«None»

URI також може вказувати систему координат, поля та індексацію постачальника пам’яті в URI. Синтаксис:

crs=визначення

Визначає систему відліку координат, де визначення може бути будь-якою з прийнятих форм QgsCoordinateReferenceSystem.createFromString()

індекс=так

Визначає також, чи буде провайдер використовувати просторовий індекс:

поле=назва:тип(довжина,точність)

Задає атрибути шару. Кожен атрибут має ім'я та, опціонально, тип (ціле число, дійсне число або рядок), довжину та точність. Таких описів може бути кілька.

Наступний приклад URI містить усі ці параметри:

```
«Point?crs=epsg:4326&field=id:integer&field=name:str
ing(20)&index=yes»
```

Ще один приклад коду ілюструє створення та заповнення постачальника пам’яті:

```
from qgis.PyQt.QtCore import QVariant

# create layer
vl = QgsVectorLayer(«Point», «temporary_points», «memory»)
pr = vl.dataProvider()

# add fields
pr.addAttributes([QgsField(«name», QVariant.String),
                  QgsField(«age»,  QVariant.Int),
                 QgsField(«size», QVariant.Double)])
vl.updateFields() # tell the vector layer to fetch changes
from the provider

# add a feature
fet = QgsFeature()
fet.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(10,10)))
fet.setAttributes([«Johny», 2, 0.3])
pr.addFeatures([fet])

# update layer's extent when new features have been added
# because change of extent in provider is not propagated to
the layer
vl.updateExtents()
```

Нарешті перевірмо, чи всі дії були успішними:

```
 # show some stats
 print(«fields:», len(pr.fields()))
 print(«features:», pr.featureCount())
 e = vl.extent()
 print(«extent:», e.xMinimum(), e.yMinimum(),
e.xMaximum(), e.yMaximum())

 # iterate over features
 features = vl.getFeatures()
 for fet in features:
```

```
    print(«F:», fet.id(), fet.attributes(),
fet.geometry().asPoint())
fields: 3
features: 1
extent: 10.0 10.0 10.0 10.0
F: 1 ['Johny', 2, 0.3] <QgsPointXY: POINT(10 10)>
```

7.11.15. Зовнішній вигляд (символіка) векторних шарів

У малюванні векторного шару зовнішній вигляд даних визначається **рендером** і **символами** , асоційованими з шаром. Символи – це класи, призначені для відмальовування візуального подання об'єктів, а рендер визначає який символ буде використаний для окремого об'єкта.

Рендерер для певного шару можна отримати так:

renderer = layer.renderer()

І з цим посиланням дослідимо це

print(«Type:», renderer.type())

Type: singleSymbol

У базовій бібліотеці QGIS є кілька відомих типів рендерів (табл. 7.1).

Таблиця 7.1

Рендери у бібліотеці QGIS

Тип	Клас	Опис
singleSymbol	`QgsSingleSymbolRenderer`	Малює всі об'єкти одним і тим самим символом
категоризований символ	`QgsCategorizedSymbolRenderer`	Малює об'єкти, використовуючи різні символи для кожної категорії
дипломований символ	`QgsGraduatedSymbolRenderer`	Малює об'єкти, використовуючи різні символи для кожного діапазону значень

Можуть також бути деякі спеціальні типи засобів візуалізації, тому не варо припускати, що існують лише ці типи. Ви можете запитати програму `QgsRendererRegistry`, щоб дізнатися доступні на цей момент рендерери:

```
print(QgsApplication.rendererRegistry().rendererslist())
['nullSymbol', 'singleSymbol', 'categorizedSymbol',
'graduatedSymbol', 'RuleRenderer', 'pointDisplacement',
```

```
'pointCluster', 'mergedFeatureRenderer',
'invertedPolygonRenderer', 'heatmapRenderer',
'25dRenderer', 'embeddedSymbol']
```

Якщо є можливість отримати дамп вмісту рендерера в текстовому вигляді – це може бути корисним для налаштування:

```
renderer.dump()
SINGLE: MARKER SYMBOL (1 layers) color 190,207,80,255
```

7.11.16. Відтворення одного символу

Ви можете отримати символ, який використовується для відтворення, викликавши `symbol()`, метод і змінити його за допомогою `setSymbol()` методу (примітка для розробників C++: засіб відтворення бере на себе право власності на символ).

Ви можете змінити символ, який використовується певним векторним шаром, викликавши `setSymbol()` передавання відповідного екземпляра символу. Символи для шарів *точок* , *ліній* і *багатокутників* можна створити, викликавши `createSimple()` функцію відповідних класів `QgsMarkerSymbol` і .`QgsLineSymbolQgsFillSymbol`.

Переданий словник `createSimple()` встановлює властивості стилю символу.

Наприклад, можна замінити символ, використовуваний певним **точковим** шаром, викликавши за допомогою `setSymbol()` передачу екземпляра `QgsMarkerSymbol`, як у такому прикладі коду:

```
symbol = QgsMarkerSymbol.createSimple({'name':
'square', 'color': 'red'})
layer.renderer().setSymbol(symbol)
# show the change
layer.triggerRepaint()
```

Властивість *name* вказує на форму маркера та може бути будь-яким з наведеного:

- circle;
- square;
- cross;
- rectangle;
- diamond;

- pentagon;
- triangle;
- equilateral_triangle;
- star;
- regular_star;
- arrow;
- filled_arrowhead;
- x.

Щоб отримати повний список властивостей для першого символьного шару екземпляра символу, ви можете виконати приклад коду:

```
print(layer.renderer().symbol().symbolLayers()[0].properties())
{'angle': '0', 'cap_style': 'square', 'color': '255,0,0,255', 'horizontal_anchor_point': '1', 'joinstyle': 'bevel', 'name': 'square', 'offset': '0,0', 'offset_map_unit_scale': '3x:0,0,0,0,0,0', 'offset_unit': 'MM', 'outline_color': '35,35,35,255', 'outline_style': 'solid', 'outline_width': '0', 'outline_width_map_unit_scale': '3x:0,0,0,0,0,0', 'outline_width_unit': 'MM', 'scale_method': 'diameter', 'size': '2', 'size_map_unit_scale': '3x:0,0,0,0,0,0', 'size_unit': 'MM', 'vertical_anchor_point': '1'}
```

Це може бути корисним, якщо ви хочете змінити деякі властивості:

```
# You can alter a single property...
layer.renderer().symbol().symbolLayer(0).setSize(3)
# ... but not all properties are accessible from methods,
# you can also replace the symbol completely:
props = layer.renderer().symbol().symbolLayer(0).properties()
props['color'] = 'yellow'
props['name'] = 'square'

layer.renderer().setSymbol(QgsMarkerSymbol.createSimple(props))
# show the changes
layer.triggerRepaint()
```

7.11.17. Категоризований рендерер символів

Використовуючи категоризований рендерер, ви можете запитувати та встановлювати атрибут, який призначений для класифікації: використовуйте методи `classAttribute()` та `setClassAttribute()`.

Приклад коду отримання список категорій:

```
categorized_renderer = QgsCategorizedSymbolRenderer()
# Add a few categories
cat1 = QgsRendererCategory('1', QgsMarkerSymbol(),
'category 1')
cat2 = QgsRendererCategory('2', QgsMarkerSymbol(),
'category 2')
categorized_renderer.addCategory(cat1)
categorized_renderer.addCategory(cat2)

for cat in categorized_renderer.categories():
    print(«{}: {} :: {}».format(cat.value(),
cat.label(), cat.symbol()))
: category 1 :: <qgis._core.QgsMarkerSymbol object at
0x7f378ffcd9d8>
: category 2 :: <qgis._core.QgsMarkerSymbol object at
0x7f378ffcd9d8>
```

Де `value()` – значення, яке використовується для розрізнення категорій, `label()` – це текст, який використовується для опису категорії, а `symbol()` метод повертає призначений символ.

Засіб візуалізації зазвичай зберігає також оригінальний символ і колірну лінійку, які були використані для класифікації: `sourceColorRamp()` і `sourceSymbol()` методи.

7.11.18. Градуйований інструмент візуалізації символів

Цей рендер дуже схожий на рендер описаними унікальними значеннями, але замість одного значення атрибута для класу він оперує діапазоном значень отже, може використовуватися тільки з числовими атрибутами.

Для того щоби дізнатися більше про діапазони, які використовуються в рендерері:

```
graduated_renderer = QgsGraduatedSymbolRenderer()
# Add a few categories
```

```
graduated_renderer.addClassRange(QgsRendererRange(QgsCla
ssificationRange('class 0-100', 0, 100),
QgsMarkerSymbol()))

graduated_renderer.addClassRange(QgsRendererRange(QgsCla
ssificationRange('class 101-200', 101, 200),
QgsMarkerSymbol()))

for ran in graduated_renderer.ranges():
    print(«{} - {}: {} {}».format(
        ran.lowerValue(),
        ran.upperValue(),
        ran.label(),
        ran.symbol()
      ))
0.0 - 100.0: class 0-100 <qgis._core.QgsMarkerSymbol
object at 0x7f8bad281b88>
101.0 - 200.0: class 101-200 <qgis._core.QgsMarkerSymbol
object at 0x7f8bad281b88>
```

Ви знову можете використовувати методи `classAttribute()` (щоб знайти назву атрибута класифікації) `sourceSymbol()` і `sourceColorRamp()`. Крім того, можна застосувати `mode()` метод, за яким визначити, як створювалися діапазони: за допомогою рівних інтервалів, квантилів або іншим методом.

Якщо ви бажаєте створити власний засіб відтворення градуйованих символів, це можете зробити, як показано у наведеному далі прикладі фрагмента (з метою створення простого розміщення двох класів):

```
 from qgis.PyQt import QtGui

 myVectorLayer =
QgsVectorLayer(«testdata/airports.shp», «airports»,
«ogr»)
 myTargetField = 'ELEV'
 myRangeList = []
 myOpacity = 1
 # Make our first symbol and range...
 myMin = 0.0
 myMax = 50.0
myLabel = 'Group 1'
```

```
myColour = QtGui.QColor('#ffee00')
mySymbol1 =
QgsSymbol.defaultSymbol(myVectorLayer.geometryType())
mySymbol1.setColor(myColour)
mySymbol1.setOpacity(myOpacity)
myRange1 = QgsRendererRange(myMin, myMax, mySymbol1,
myLabel)
myRangeList.append(myRange1)
#now make another symbol and range...
myMin = 50.1
myMax = 100
myLabel = 'Group 2'
myColour = QtGui.QColor('#00eeff')
mySymbol2 = QgsSymbol.defaultSymbol(
     myVectorLayer.geometryType())
mySymbol2.setColor(myColour)
mySymbol2.setOpacity(myOpacity)
myRange2 = QgsRendererRange(myMin, myMax, mySymbol2,
myLabel)
myRangeList.append(myRange2)
myRenderer = QgsGraduatedSymbolRenderer('',
myRangeList)
myClassificationMethod =
QgsApplication.classificationMethodRegistry().method(«E
qualInterval»)
myRenderer.setClassificationMethod(myClassificationMeth
od)
myRenderer.setClassAttribute(myTargetField)
myVectorLayer.setRenderer(myRenderer)
```

7.11.19. Робота з символами

Для представлення символів є **QgsSymbol** базовий клас із трьома похідними класами:

- `QgsMarkerSymbol` – для точкових ознак
- `QgsLineSymbol` – для лінійних особливостей
- `QgsFillSymbol` – для багатокутних об’єктів

Кожен символ складається з одного або кількох шарів символів (класів, похідних від `QgsSymbolLayer`). Шари символів виконують фактичну візуалізацію, сам клас символів слугує лише контейнером для шарів символів.

Маючи екземпляр символу (наприклад, із засобу візуалізації), його можна досліджувати: метод **type()** визначає, чи це маркер, лінія чи символ заповнення. Існує **dump()** метод, який повертає короткий опис символу. Щоб отримати список шарів символів виконуємо дії:

```
marker_symbol = QgsMarkerSymbol()
for i in range(marker_symbol.symbolLayerCount()):
   lyr = marker_symbol.symbolLayer(i)
   print(«{}: {}».format(i, lyr.layerType()))
0: SimpleMarker
```

Для того щоби дізнатися колір символу, використовуйте **color()** метод, і **setColor()** змініть його колір. Крім того, із символами маркерів можна запитувати розмір символу та обертання за допомогою методів **size()** і **angle()**. Для лінійних символів **width()** метод повертає ширину лінії.

Розмір і товщину за замовчуванням задаються у міліметрах, а кути – у градусах.

7.11.20. Робота з шарами символів

Як сказано раніше, шари символів (підкласи `QgsSymbolLayer`) визначають зовнішній вигляд функцій. Кілька базових класів шару символів придатні для загального використання. Можна впроваджувати нові типи шарів символів і таким чином довільно налаштовувати спосіб відображення функцій. Метод однозначно ідентифікує клас символьного шару `layerType()` – базовий і типовий за замовчуванням і типи символьного шару. `SimpleMarkerSimpleLineSimpleFill`

Ви можете отримати повний список типів шарів символів, які можна створити для певного класу шарів символів за допомогою такого коду:

```
from qgis.core import QgsSymbolLayerRegistry
    myRegistry = QgsApplication.symbolLayerRegistry()
    myMetadata =
    myRegistry.symbolLayerMetadata(«SimpleFill»)
for item in
myRegistry.symbolLayersForType(QgsSymbol.Marker):
    print(item)
```

```
AnimatedMarker
EllipseMarker
FilledMarker
FontMarker
GeometryGenerator
MaskMarker
RasterMarker
SimpleMarker
SvgMarker
VectorField
```

Клас `QgsSymbolLayerRegistry` керує базою даних усіх доступних типів шарів символів.

Для того щоби отримати доступ до даних шару символів, використовуйте його `properties()` метод, який повертає словник «ключ – значення» властивостей, які визначають зовнішній вигляд. Кожен тип символьного шару має певний набір властивостей, які він використовує. Крім того, є загальні методи `color()`, `size()`, `angle()` і `width()` з їхніми аналогами сетера. Звичайно, розмір і кут доступні лише для шарів символів маркерів, а ширина – для шарів символів лінії.

7.11.21. Створення власних типів шарів символів

Уявіть, що ви хочете налаштувати спосіб відображення даних. Ви можете створити власний клас символьного шару, який буде малювати об’єкти саме так, як ви бажаєте. Ось приклад маркера, який малює червоні кола із заданим радіусом:

```
from qgis.core import QgsMarkerSymbolLayer
from qgis.PyQt.QtGui import QColor

class FooSymbolLayer(QgsMarkerSymbolLayer):

  def __init__(self, radius=4.0):
      QgsMarkerSymbolLayer.__init__(self)
      self.radius = radius
      self.color = QColor(255,0,0)

 def layerType(self):
    return «FooMarker»
```

```
    def properties(self):
        return { «radius» : str(self.radius) }

    def startRender(self, context):
      pass

    def stopRender(self, context):
        pass

    def renderPoint(self, point, context):
        # Rendering depends on whether the symbol is
selected (QGIS >= 1.5)
        color = context.selectionColor() if
context.selected() else self.color
        p = context.renderContext().painter()
        p.setPen(color)
        p.drawEllipse(point, self.radius, self.radius)
    def clone(self):
        return FooSymbolLayer(self.radius)
```

Метод `layerType()` визначає назву символьного шару; він має бути унікальним серед усіх символьних шарів. Метод `properties()` використовується для збереження атрибутів. Метод `clone()` повинен повертати копію шару символу з абсолютно однаковими атрибутами. І, нарешті, є методи візуалізації: `startRender()` викликається перед рендерингом першої функції, `stopRender()` – коли рендеринг завершено, і `renderPoint()` викликається для виконання рендерингу. Координати точки (точок) уже перетворено на вихідні координати.

Для поліліній і багатокутників єдина різниця полягатиме в методі візуалізації: ви використовуєте `renderPolyline()`, який отримує список ліній, тоді як `renderPolygon()` отримує список точок на зовнішньому кільці як перший параметр і список внутрішніх кілець (або None) як другий параметр.

Зазвичай зручно додати графічний інтерфейс для встановлення атрибутів типу шару символів, щоб дати користувачам змогу налаштовувати зовнішній вигляд: у наведеному прикладі можна дати користувачеві змогу встановити радіус кола. Наступний код реалізує такий відмет:

```
from qgis.gui import QgsSymbolLayerWidget

 class FooSymbolLayerWidget(QgsSymbolLayerWidget):
     def __init__(self, parent=None):
         QgsSymbolLayerWidget.__init__(self, parent)

         self.layer = None

         # setup a simple UI
        self.label = QLabel(«Radius:»)
        self.spinRadius = QDoubleSpinBox()
        self.hbox = QHBoxLayout()
        self.hbox.addWidget(self.label)
        self.hbox.addWidget(self.spinRadius)
        self.setLayout(self.hbox)
        self.connect(self.spinRadius,
SIGNAL(«valueChanged(double)»), \
            self.radiusChanged)

    def setSymbolLayer(self, layer):
        if layer.layerType() != «FooMarker»:
            return
        self.layer = layer
        self.spinRadius.setValue(layer.radius)

    def symbolLayer(self):
        return self.layer

    def radiusChanged(self, value):
        self.layer.radius = value
        self.emit(SIGNAL(«changed()»))
```

Цей віджет можна вставити в діалогове вікно властивостей символу. Коли в діалоговому вікні властивостей символу обрано тип шару символів, створюється екземпляр шару символів і екземпляр віджета шару символів. Потім він викликає `setSymbolLayer()` метод для призначення шару символів віджету. У цьому методі віджет має оновити інтерфейс користувача, щоб відобразити атрибути шару символів. Цей `symbolLayer()` метод використовується для

повторного отримання шару символу за допомогою діалогового вікна властивостей з метою використання його для символу.

Після кожної зміни атрибутів віджет має видавати сигнал, `changed()`, щоби діалогове вікно властивостей оновлювало попередній перегляд символу.

Залишився останній штрих: розповісти QGIS про існування нових класів. Для цього достатньо додати символьний шар до реєстру. Звісно, можна використовувати символьний шар і не додаючи його до реєстру, але тоді деякі можливості будуть недоступні: наприклад, завантаження проєкту з символьними шарами користувача або неможливість редагувати властивості шару.

Нам потрібно буде створити метадані для шару символів:

```
from qgis.core import QgsSymbol,
QgsSymbolLayerAbstractMetadata, QgsSymbolLayerRegistry

 class
FooSymbolLayerMetadata(QgsSymbolLayerAbstractMetadata):

   def __init__(self):
     super().__init__(«FooMarker», «My new Foo
marker», QgsSymbol.Marker)

   def createSymbolLayer(self, props):
     radius = float(props[«radius»]) if «radius» in
props else 4.0
    return FooSymbolLayer(radius)

fslmetadata = FooSymbolLayerMetadata()
QgsApplication.symbolLayerRegistry().addSymbolLayerType
(fslmetadata)
```

Ви повинні передати тип шару (той самий, який повертає шар) і тип символу (маркер/лінія/заливка) до конструктора батьківського класу. Метод `createSymbolLayer()` піклується про створення екземпляра символьного шару з атрибутами, вказаними в словнику *реквізитів*. Крім того, є `createSymbolLayerWidget()` метод, який повертає віджет налаштувань для цього типу шару символів.

Останньою конструкцією ми додаємо символьний шар у реєстр – на цьому все.

7.11.22. Створення користувацьких рендерів

Можливість створення рендеру може бути корисною, якщо потрібно змінити правила вибору символів для відображення об'єктів. Прикладами таких ситуацій можуть бути: символ повинен визначатися на підставі значень кількох полів, розмір символу залежатиме від поточного масштабу та ін.

Наступний код показує простий налаштовуваний рендерер, який створює два символи-маркери та випадковим чином обирає один з них для кожної функції:

```
 import random
 from qgis.core import QgsWkbTypes, QgsSymbol,
QgsFeatureRenderer
  class RandomRenderer(QgsFeatureRenderer):
   def __init__(self, syms=None):
     super().__init__(«RandomRenderer»)
     self.syms = syms if syms else [

QgsSymbol.defaultSymbol(QgsWkbTypes.geometryType(QgsWkbTy
pes.Point)),
QgsSymbol.defaultSymbol(QgsWkbTypes.geometryType(QgsWkbTy
pes.Point))
    ]

  def symbolForFeature(self, feature, context):
    return random.choice(self.syms)

  def startRender(self, context, fields):
    super().startRender(context, fields)
    for s in self.syms:
      s.startRender(context, fields)

  def stopRender(self, context):
    super().stopRender(context)
    for s in self.syms:
      s.stopRender(context)

  def usedAttributes(self, context):
    return []

  def clone(self):
    return RandomRenderer(self.syms)
```

Конструктору `QgsFeatureRenderer` класу потрібна назва рендерера (яка має бути унікальною серед рендерерів). Метод `symbolForFeature()` – це той, який вирішує, який символ буде використано для певної функції. Опції `startRender()` і `stopRender()` призначені для ініціалізації/завершення відтворення символів. Метод `usedAttributes()` може повертати список імен полів, які очікує присутність рендерера. Нарешті, `clone()` метод повинен повертати копію рендерера.

Подібно до шарів символів, можна приєднати графічний інтерфейс для конфігурації рендерера. Він повинен бути похідним від `QgsRendererWidget`. Наведений далі приклад коду створює кнопку, яка дає користувачеві змогу встановити перший символ:

e

Конструктор отримує екземпляри активного шару (`QgsVectorLayer`), глобального стилю (`QgsStyle`) і поточного рендерера. Якщо рендерера немає або рендерер має інший тип, він буде замінений новим рендерером, інакше треба використовувати поточний рендерер (який уже має потрібний нам тип). Вміст віджета слід оновити, щоб відобразити поточний стан рендерера. Коли діалогове вікно візуалізації прийнято, викликається метод віджета `renderer()`, щоб отримати поточний рендерер – він буде призначений шару.

Останнім відсутнім фрагментом, якого бракує, є метадані рендерера та реєстрація в реєстрі, інакше завантаження шарів із рендерером не працюватиме і користувач не зможе обрати його зі списку рендерерів. Закінчимо наш приклад:

```
from qgis.core import (
  QgsRendererAbstractMetadata,
  QgsRendererRegistry,
  QgsApplication
)

class
RandomRendererMetadata(QgsRendererAbstractMetadata):

  def __init__(self):
```

```
    super().__init__(«RandomRenderer», «Random
renderer»)

  def createRenderer(self, element):
    return RandomRenderer()
  def createRendererWidget(self, layer, style,
renderer):
    return RandomRendererWidget(layer, style, renderer)
rrmetadata = RandomRendererMetadata()
QgsApplication.rendererRegistry().addRenderer(rrmetadata)
```

Подібно до символьних шарів, конструктор абстрактних метаданих очікує назви рендерера, імені, видимого для користувачів, і, за бажанням, імені піктограми рендерера. Метод `createRenderer()` передає `QDomElement` екземпляр, який можна використовувати для відновлення стану рендерера з дерева DOM. Метод `createRendererWidget()` створює віджет конфігурації. Він не повинен бути присутнім або може повернутися, `None` якщо рендерер не постачається з графічним інтерфейсом користувача.

Щоб пов'язати піктограму з рендерером, ви можете призначити її в `QgsRendererAbstractMetadata` конструкторі як третій (необов'язковий) аргумент – конструктор базового класу у функції `RandomRendererMetadata __init__()` стає:

```
    QgsRendererAbstractMetadata.__init__(self,
           «RandomRenderer»,
           «Random renderer»,
           QIcon(QPixmap(«RandomRendererIcon.png»,
    «png»)))
```

Піктограму також можна асоціювати в будь-який час за допомогою `setIcon()` методу класу метаданих. Піктограму можна завантажити з файла (як показано вище) або завантажити з ресурсу Qt (PyQt5 містить компілятор .qrc для Python).

7.12. Обробка геометрії

Points, linestrings і polygons у QGIS представлені `QgsGeometry` класом.

Одночаси однієї geometry є в даний час колекція простих (одна частина) geometries. Така geometry називається багаточастинною multi-part geometry. Якщо він містить тільки один тип simple geometry, потрібно скористатися `multi-point`, `multi-linestring` або `multi-polygon`. Для прикладу, країна, що містить багато islands, може бути представлена як `multi-polygon`.

Координати, що описують геометрію, можуть бути у будь-якій системі координат (CRS). Коли доступ до об'єктів шару, асоційовані геометрії будуть видані з координатами в СК шару.

Зображення і специфікації всіх можливих геометричних примітивів та відношень між класами об’єктів доступні в OGC `Simple Feature Access Standards`.

PyQGIS забезпечує кілька варіантів для створення `geometry`:

```
from coordinates
gPnt = QgsGeometry.fromPointXY(QgsPointXY(1,1))
print(gPnt)
gLine = QgsGeometry.fromPolyline([QgsPoint(1, 1),
QgsPoint(2, 2)])
print(gLine)
gPolygon =
QgsGeometry.fromPolygonXY([[QgsPointXY(1, 1),
QgsPointXY(2, 2), QgsPointXY(2, 1)]])
print(gPolygon)
```

Координати оголошено за допомогою `QgsPoint` класу або `QgsPointXY` класу. Відмінність між ними в тому, що `QgsPoint` підтримують M і Z значення вимірів.

Складові геометрії мають додатковий рівень вкладеності, як-от: мульти-точка – множина точок, мульти-лінія – множина ліній, а мульти-полігон є множиною полігонів.

```
geom = QgsGeometry.fromWkt(«POINT(3 4)»)
print(geom)
```

```
well-known binary (WKB)
g = QgsGeometry()

# вхідна геометрія у WKB форматі
wkb =
bytes.fromhex(«0101000000000000000000454000000000000001440»)
g.fromWkb(wkb)

# подання вихідної геометрії у WKT форматі
print(g.asWkt())
```

7.13. Доступ до геометрії

Спершу треба перейти від типу geometry. Метод `wkbType()` є одним із поширених методів перетворення. Метод повертає значення з переліку `QgsWkbTypes.Type`.

```
if gPnt.wkbType() == QgsWkbTypes.Point:
  print(gPnt.wkbType())
  # output: 1 for Point
if gLine.wkbType() == QgsWkbTypes.LineString:
  print(gLine.wkbType())
  # output: 2 for LineString
if gPolygon.wkbType() == QgsWkbTypes.Polygon:
  print(gPolygon.wkbType())
  # output: 3 for Polygon
```

Як альтернативу, можна використовувати `type()` метод, який відновить значення від `QgsWkbTypes.GeometryType` назви.

Ви можете використовувати `displayString()` функцію для отримання людського readable geometry типу:

```
print(QgsWkbTypes.displayString(gPnt.wkbType()))
# output: 'Point'
print(QgsWkbTypes.displayString(gLine.wkbType()))
# output: 'LineString'
print(QgsWkbTypes.displayString(gPolygon.wkbType()))
# output: 'Polygon'
```

```
Point
LineString
Polygon
```

Є також допоміжна функція `isMultipart()`, щоб визначити, де геометрія є мульти- чи простою.

Для отримання додаткової інформації з geometrie є функцією accessor для кожного векторного типу. Тут ви знайдете, як використати ці accessors:

```
1print(gPnt.asPoint())
2# output: <QgsPointXY: POINT(1 1)>
3print(gLine.asPolyline())
4# output: [<QgsPointXY: POINT(1 1)>,
<QgsPointXY: POINT(2 2)>]
5print(gPolygon.asPolygon())
6# output: [[<QgsPointXY: POINT(1 1)>,
<QgsPointXY: POINT(2 2)>, <QgsPointXY: POINT(2
1)>, <QgsPointXY: POINT(1 1)>]]
```

Пара координат (x,y) не є реальною, це `QgsPoint` клас об'єктів, значення яких отримано з методів `x()` та `y()`.

Для мульти-геометрії є аналогічні функції функції: `asMultiPoint()`, `asMultiPolyline()` та `asMultiPolygon()`.

Це може бути використане ітераційно для всіх частин мульти-геометрії, незалежно від типу геометрії:

```
geom = QgsGeometry.fromWkt( 'MultiPoint( 0 0, 1 1, 2
2)' )
for part in geom.parts():
  print(part.asWkt())
Point (0 0)
Point (1 1)
Point (2 2)
geom = QgsGeometry.fromWkt( 'LineString( 0 0, 10 10
)' )
for part in geom.parts():
  print(part.asWkt())
```

```
LineString (0 0, 10 10)
gc = QgsGeometryCollection()
gc.fromWkt('GeometryCollection( Point(1 2), Point(11
12), LineString(33 34, 44 45))')
print(gc[1].asWkt())
Point (11 12)
```

Це також можливо, щоб змінити їхню частину geometry використовуючи `QgsGeometry.parts()` метод.

```
geom = QgsGeometry.fromWkt( 'MultiPoint( 0 0, 1 1,
2 2)' )
for part in geom.parts():
  part.transform(QgsCoordinateTransform(
    QgsCoordinateReferenceSystem(«EPSG:4326»),
    QgsCoordinateReferenceSystem(«EPSG:3111»),
    QgsProject.instance())
  )

print(geom.asWkt())
MultiPoint ((-10334728.12541878595948219 -
5360106.25905461423099041),(-
10462135.16126426123082638 -
5217485.4735023295506835),(-
10589399.84444035589694977 -
5072021.45942386891692877))
```

7.14. Геометричні предикати та операції

QGIS використовує GEOS бібліотеку для розширених геометричних операцій, наприклад, геометричні предикати (`contains()`, `intersects()`, ...) та операції (`combine()`, `difference()`, ...). Це також можуть бути складені картометричні властивості геопросторових об'єктів, наприклад ***`area`*** та ***`lengths`***.

Наводимо приклад визначення площ і периметрів країн світу, які подані полігональними об'єктами:

```
# let's access the 'countries' layer
layer = QgsProject.instance().
mapLayersByName('countries')[0]
```

```
 # let's filter for countries that begin with Z,
then get their features
 query = '«name» LIKE \'Z%\''
 features =
layer.getFeatures(QgsFeatureRequest().setFilterExpre
ssion(query))

 # now loop through the features, perform geometry
computation and print the results
 for f in features:
  geom = f.geometry()
  name = f.attribute('NAME')
  print(name)
  print('Area: ', geom.area())
  print('Perimeter: ', geom.length())
Zambia
Area:  62.822790653431014
Perimeter:  50.65232014052552
Zimbabwe
Area:  33.41113559136511
Perimeter:  26.608288555013935
```

Тепер ви повинні розрізняти і завантажити області і периметри геометричних. Це існують області і периметри, щоб не використовувати CRS в аккаунті, коли комп'ютер використовує `area()` і `length()` методи від `QgsGeometry` класу. Для більшої потужної області і величини калькуляції, `QgsDistanceArea` клас може бути використаний, який може виконувати обчислення на референц-еліпсоїді:

```
d = QgsDistanceArea()
d.setEllipsoid('WGS84')

 layer =
QgsProject.instance().mapLayersByName('countries')[0
]

 # let's filter for countries that begin with Z,
then get their features
 query = '«name» LIKE \'Z%\''
```

```
 features =
layer.getFeatures(QgsFeatureRequest().setFilterExpre
ssion(query))

for f in features:
  geom = f.geometry()
  name = f.attribute('NAME')
  print(name)
  print(«Perimeter (m):», d.measurePerimeter(geom))
  print(«Area (m2):», d.measureArea(geom))

  # let's calculate and print the area again, but
this time in square kilometers
  print(«Area (km2):»,
d.convertAreaMeasurement(d.measureArea(geom),
QgsUnitTypes.AreaSquareKilometers))
Zambia
Perimeter (m): 5539361.250294601
Area (m2): 751989035032.9031
Area (km2): 751989.0350329031
Zimbabwe
Perimeter (m): 2865021.3325076113
Area (m2): 389267821381.6008
Area (km2): 389267.8213816008
```

Крім того, ви можете подумати про відстань і розрив між двома пунктами.

```
 d = QgsDistanceArea()
 d.setEllipsoid('WGS84')

 # Let's create two points.
 # Santa claus is a workaholic and needs a summer
break,
 # lets see how far is Tenerife from his home
 santa = QgsPointXY(25.847899, 66.543456)
 tenerife = QgsPointXY(-16.5735, 28.0443)

print(«Distance in meters: «, d.measureLine(santa,
tenerife))
```

Є чимало прикладів алгоритмів, які містяться у QGIS, і використовуються для аналізу і перетворення векторних даних:

1. Визначення довжини і площи з використанням класу `QgsDistanceArea`
2. Алгоритм перетворення ліній на полігони

7.15. Підтримка проєкцій

7.15.1. Системи координат

Системи відліку координат (CRS) інкапсульовані класом **QgsCoordinateReferenceSystem**. Екземпляри цього класу можна створити кількома різними способами.

Вкажіть CRS за своїм ідентифікатором:

```
# EPSG 4326 is allocated for WGS84
crs = QgsCoordinateReferenceSystem(«EPSG:4326»)
print(crs.isValid())
True
```

QGIS підтримує різні ідентифікатори CRS у таких форматах:

1. `EPSG:<code>` – ідентифікатор, поданий організацією EPSG – обробляється `createFromOgcWms()`
2. `POSTGIS:<srid>` – ідентифікатор, використовуваний в базах даних PostGIS, – обробляється `createFromSrid()`
3. `INTERNAL:<srsid>` – ідентифікатор, використовуваний у внутрішній базі даних QGIS, – обробляється **createFromSrsId()**
4. `PROJ:<proj>` – обробляється з `createFromProj()`
5. `WKT:<wkt>` – обробляється з `createFromWkt()`

Якщо префікс не вказано, передбачається визначення WKT. Укажіть CRS за його загальновідомим текстом (WKT):

```
wkt = 'GEOGCS[«WGS84», DATUM[«WGS84»,
SPHEROID[«WGS84», 6378137.0, 298.257223563]],' \
      'PRIMEM[«Greenwich», 0.0],
UNIT[«degree»,0.0174532925199433295],' \
      'AXIS[«Longitude»,EAST],
AXIS[«Latitude»,NORTH]]'
```

```
crs = QgsCoordinateReferenceSystem(wkt)
print(crs.isValid())
True
```

Створити недійсну систему координат CRS, а потім використати одну з функцій `create*` для її ініціалізації. У наступному прикладі використовуємо рядок Proj для ініціалізації проєкції.

```
crs = QgsCoordinateReferenceSystem()
crs.createFromProj(«+proj=longlat +ellps=WGS84
+datum=WGS84 +no_defs»)
print(crs.isValid())
True
```

Доцільно перевірити, чи було створення CRS (тобто пошук у базі даних) успішним: `isValid()` має повернути `True`.

Зауважте, що для ініціалізації систем просторового відліку QGIS має шукати відповідні значення у своїй внутрішній базі даних `srs.db`. Таким чином, якщо ви створюєте незалежну програму, потрібно правильно встановити шляхи за допомогою `QgsApplication.setPrefixPath()`, інакше вона не зможе знайти базу даних. Якщо ви виконуєте команди з консолі QGIS Python або розробляєте плагін, вас це не обходить: усе вже налаштовано.

Доступ до інформації про систему просторового відліку:

```
crs = QgsCoordinateReferenceSystem(«EPSG:4326»)

print(«QGIS CRS ID:», crs.srsid())
print(«PostGIS SRID:», crs.postgisSrid())
print(«Description:», crs.description())
print(«Projection Acronym:»,
crs.projectionAcronym())
print(«Ellipsoid Acronym:», crs.ellipsoidAcronym())
print(«Proj String:», crs.toProj())
# check whether it's geographic or projected
coordinate system
print(«Is geographic:», crs.isGeographic())
# check type of map units in this CRS (values
defined in QGis::units enum)
print(«Map units:», crs.mapUnits())
```

Результат:

```
QGIS CRS ID: 3452
PostGIS SRID: 4326
Description: WGS 84
Projection Acronym: longlat
Ellipsoid Acronym: EPSG:7030
Proj String: +proj=longlat +datum=WGS84 +no_defs
Is geographic: True
Map units: 6
```

7.15.2. Трансформування систем координат

Перетворення між різними просторовими системами відліку можна виконувати за допомогою `QgsCoordinateTransform` класу. Найпростіший спосіб його використання – створити вихідну та цільову CRS й створити екземпляр `QgsCoordinateTransform` з ними та поточним проєктом. Потім потрібно кілька разів викликати `transform()` функцію, щоби виконати перетворення. За замовчуванням він виконує пряме перетворення, але здатен виконувати і зворотне перетворення:

```
 crsSrc = QgsCoordinateReferenceSystem(«EPSG:4326»)
# WGS 84
 crsDest =
QgsCoordinateReferenceSystem(«EPSG:32633»)  # WGS 84
/ UTM zone 33N
 transformContext =
QgsProject.instance().transformContext()
 xform = QgsCoordinateTransform(crsSrc, crsDest,
transformContext)

 # forward transformation: src -> dest
 pt1 = xform.transform(QgsPointXY(18,5))
 print(«Transformed point:», pt1)

# inverse transformation: dest -> src
pt2 = xform.transform(pt1,
QgsCoordinateTransform.ReverseTransform)
print(«Transformed back:», pt2)
```

Результат:

```
Transformed point: <QgsPointXY:
POINT(832713.79873844375833869
553423.98688333143945783)>
Transformed back: <QgsPointXY: POINT(18
4.99999999999999911)>
```

7.16. Використання полотна карти

Віджет «Полотно карти» є одним із найважливіших віджетів у QGIS, оскільки він показує карту з накладених шарів, і дає змогу взаємодіяти з картою та шарами. На полотні завжди відображається частина карти, визначена поточною протяжністю полотна. Взаємодія здійснюється за допомогою інструментів карти: є інструменти для панорамування, масштабування, визначення шарів, вимірювання, векторного редагування та інші. Подібно до інших графічних програм, завжди є один активний інструмент, і користувач може перемикатися між доступними інструментами.

Полотно карти реалізовано за допомогою `QgsMapCanvas` класу в модулі *qgis.gui*. Реалізація базується на фреймворку `Qt Graphics View`. Цей фреймворк зазвичай надає поверхню та подання, де розміщуються спеціальні графічні елементи, і користувач може з ними взаємодіяти. Ми припустимо, що ви достатньо знайомі з Qt, щоб зрозуміти концепції графічної сцени, подання та елементів. Якщо ні, прочитайте огляд фреймворку.

Щоразу, коли карту було панорамовано, збільшено/зменшено (або виеонано будь-яку іншу дію, яка ініціює оновлення), карта знову відображається в межах поточного екстенту. Шари рендеряться в зображення (за допомогою `QgsMapRendererJob` класу), і це зображення відображається на полотні. Клас `QgsMapCanvas` також керує оновленням візуалізованої карти. Окрім цього елемента, який виконує роль фону, можуть бути інші **елементи полотна карти**.

Типовими елементами полотна карти є гумки (використовуються для вимірювання, векторного редагування тощо) або маркери вершин. Елементи полотна зазвичай використовуються для візуального зворотного зв’язку для інструментів карти, наприклад, під час створення нового багатокутника інструмент карти створює елемент полотна

гумової стрічки, який відображає поточну форму багатокутника. Усі елементи полотна карти є підкласами, `QgsMapCanvasItem` які додають додаткові функції базовим *`QGraphicsItem`* об'єктам.

Таким чином, архітектурно – мапа складається з трьох елементів:

- картка – для відображення даних;
- елементи полотна карти – додаткові елементи, які можна відобразити на полотні карти;
- інструменти карти – для взаємодії з полотном карти.

7.16.1. Вбудовування карти

Це створить окреме вікно з полотном карти. Його також можна вбудувати в наявний віджет або вікно. Під час використання `.ui` файлів і Qt Designer розмістіть QWidget у формі та перемістіть його до нового класу: встановіть QgsMapCanvas як назву класу та встановіть qgis.gui як файл заголовка pyuic5. Про це подбає комунальне підприємство. Це дуже зручний спосіб вбудовування полотна. Інша можливість полягає в тому, щоб вручну написати код для створення полотна карти та інших віджетів (як дітей головного вікна чи діалогового вікна) і створити макет.

```
canvas = QgsMapCanvas()
canvas.show()
```

За замовчуванням полотно карти має чорний фон і не використовує згладжування. Щоб установити білий фон і ввімкнути згладжування для плавного відтворення виконаємо такі дії:

```
canvas.setCanvasColor(Qt.white)
canvas.enableAntiAliasing(True)
```

Тепер настав час додати кілька шарів карти. Спочатку відкриємо шар і додамо його до поточного проекту. Потім встановимо розмір полотна та список шарів для полотна:

```
vlayer = QgsVectorLayer('testdata/airports.shp',
«Airports layer», «ogr»)
 if not vlayer.isValid():
     print(«Layer failed to load!»)
```

```
# add layer to the registry
 QgsProject.instance().addMapLayer(vlayer)

 # set extent to the extent of our layer
 canvas.setExtent(vlayer.extent())

# set the map canvas layer set
canvas.setLayers([vlayer])
```

Після виконання цих команд на карті має з'явитися завантажений шар.

7.16.2. Смуги та маркери вершин

Для того щоби показати деякі додаткові дані поверх карти на полотні, тркба використатися елементами полотна карти. Можна створювати власні класи елементів полотна (розглянуто далі), однак для зручності є два корисні класи елементів полотна: `QgsRubberBand` для малювання ламаних ліній або багатокутників і `QgsVertexMarker` для малювання точок. Обидва вони працюють із координатами карти, тому фігура автоматично переміщується/масштабується під час панорамування чи масштабування полотна.

Щоб показати полілінію:

```
r = QgsRubberBand(canvas, QgsWkbTypes.LineGeometry)  # line
points = [QgsPoint(-100, 45), QgsPoint(10, 60),
QgsPoint(120, 45)]
r.setToGeometry(QgsGeometry.fromPolyline(points), None)
```

Показати багатокутник:

```
r = QgsRubberBand(canvas, QgsWkbTypes.PolygonGeometry)  #
polygon
points = [[QgsPointXY(-100, 35), QgsPointXY(10, 50),
QgsPointXY(120, 35)]]
r.setToGeometry(QgsGeometry.fromPolygonXY(points), None)
```

Зауважте, що вузли полігону представлені не плоским списком; насправді це список меж полігону. Перше кільце описує зовнішній контур, решта (не обов'язкові) – відповідні діркам у полігоні.

Гумки дають змогу змінювати колір і ширину лінії:

```
r.setColor(QColor(0, 0, 255))
r.setWidth(3)
```

Елементи полотна прив’язані до сцени полотна. Щоб тимчасово приховати їх (і показати знову), використовують комбінацію клавіш `hide()` і `show()`. Щоб повністю видалити предмет, його потрібно видалити зі сцени полотна `canvas.scene().removeItem(r)`.

(Використовуючм C++ можна просто видалити елемент, однак у Python видалить лише посилання, а сам об'єкт залишиться на місці, тому що його власником є карта) `del r`.

Гумку також можна використовувати для малювання точок, але `QgsVertexMarker` клас краще підходить для цього (`QgsRubberBand` намалював би лише прямокутник навколо потрібної точки).

Маркер вершини використовують так:

```
m = QgsVertexMarker(canvas)
m.setCenter(QgsPointXY(10,40))
```

Це намалює червоний хрест на позиції **[10,45]**. Можна налаштувати тип значка, розмір, колір і ширину пера:

```
m.setColor(QColor(0, 255, 0))
m.setIconSize(5)
m.setIconType(QgsVertexMarker.ICON_BOX) # or
ICON_CROSS, ICON_X
m.setPenWidth(3)
```

Для того щоби тимчасово приховати маркери вершин або видалити їх з полотна, застосовують ті самі методи, що й для гумок.

7.16.3. Використання інструментів картки

У наступному прикладі створюється вікно, яке містить полотно карти, та основні інструменти карти для панорамування та масштабування карти. Для активації кожного інструмента виконано дії: панорамування – за допомогою `QgsMapToolPan`, збільшення/зменшення масштабу – за допомогою пари `QgsMapToolZoom` примірників. Дії встановлюються як доступні для перевірки та пізніше призначають інструментам, щоб дозволити автоматичну обробку позначеного/неперевіреного стану дій – коли інструмент карти активується, його дія позначається як обрана, а дія

попереднього інструмента карти скасовується. Інструменти карти активуються за допомогою `setMapTool()` методу.

```
from qgis.gui import *
from qgis.PyQt.QtWidgets import QAction, QMainWindow
from qgis.PyQt.QtCore import Qt

 class MyWnd(QMainWindow):
     def __init__(self, layer):
         QMainWindow.__init__(self)

          self.canvas = QgsMapCanvas()
         self.canvas.setCanvasColor(Qt.white)

         self.canvas.setExtent(layer.extent())
         self.canvas.setLayers([layer])

         self.setCentralWidget(self.canvas)

         self.actionZoomIn = QAction(«Zoom in», self)
         self.actionZoomOut = QAction(«Zoom out»,
self)
         self.actionPan = QAction(«Pan», self)

         self.actionZoomIn.setCheckable(True)
         self.actionZoomOut.setCheckable(True)
         self.actionPan.setCheckable(True)


self.actionZoomIn.triggered.connect(self.zoomIn)

self.actionZoomOut.triggered.connect(self.zoomOut)
       self.actionPan.triggered.connect(self.pan)

         self.toolbar = self.addToolBar(«Canvas
actions»)
         self.toolbar.addAction(self.actionZoomIn)
         self.toolbar.addAction(self.actionZoomOut)
         self.toolbar.addAction(self.actionPan)

         # create the map tools
         self.toolPan = QgsMapToolPan(self.canvas)
         self.toolPan.setAction(self.actionPan)
```

```
            self.toolZoomIn = QgsMapToolZoom(self.canvas,
        False) # false = in
            self.toolZoomIn.setAction(self.actionZoomIn)
            self.toolZoomOut =
        QgsMapToolZoom(self.canvas, True) # true = out

        self.toolZoomOut.setAction(self.actionZoomOut)

            self.pan()

        def zoomIn(self):
            self.canvas.setMapTool(self.toolZoomIn)

        def zoomOut(self):
            self.canvas.setMapTool(self.toolZoomOut)

        def pan(self):
            self.canvas.setMapTool(self.toolPan)
```

Ви можете випробувати наведений код у редакторі консолі Python. Щоб викликати вікно полотна, додайте наступні рядки для створення екземпляра `MyWnd` класу. Вони візуалізують обраний шар на щойно створеному полотні.

```
w = MyWnd(iface.activeLayer())
w.show()
```

7.16.4. Вибір об'єкта за допомогою `QgsMapToolIdentifyFeature`

Ви можете використовувати інструмент карти `QgsMapToolIdentifyFeature`, щоб попросити користувача обрати функцію, яка буде надіслана до функції зворотного виклику.

```
def callback(feature):
    ««««Code called when the feature is selected by the user»»»»
    print(«You clicked on feature {}».format(feature.id()))

canvas = iface.mapCanvas()
feature_identifier = QgsMapToolIdentifyFeature(canvas)

# indicates the layer on which the selection will be done
feature_identifier.setLayer(vlayer)
```

```
# use the callback as a slot triggered when the user
identifies a feature
feature_identifier.featureIdentified.connect(callback)

# activation of the map tool
canvas.setMapTool(feature_identifier)
```

7.16.5. Додавання елементів до контекстного меню полотна карти

Взаємодія з полотном карти також може здійснюватися за допомогою записів, які ви можете додати до його контекстного меню за допомогою сигналу `contextMenuAboutToShow`.

Наступний код додає дію **«Моє меню ► Моя дія»** поряд із записами за замовчуванням, коли ви клацаєте правою кнопкою миші на полотні карти.

```
# a slot to populate the context menu
def populateContextMenu(menu: QMenu, event:
QgsMapMouseEvent):
    subMenu = menu.addMenu('My Menu')
    action = subMenu.addAction('My Action')
    action.triggered.connect(lambda *args:
                             print(f'Action
triggered at {event.x()},{event.y()}'))

canvas.contextMenuAboutToShow.connect(populateConte
xtMenu)
canvas.show()
```

7.16.6. Створення власних інструментів карти

Ви можете написати власні інструменти, щоб реалізувати власну поведінку для дій, які виконують користувачі на полотні.

Інструменти карти мають успадкувати від `QgsMapTool` класу або будь-якого похідного класу та обрати як активні інструменти на полотні за допомогою `setMapTool()` методу, як ми вже бачили, про це вже йшлося.

Ось приклад інструмента карти, який дає змогу визначати прямокутний екстент, клацаючи та перетягуючи полотно. Коли прямокутник визначено, він друкує його граничні координати на

консолі. Він використовує елементи гумової смуги, описані раніше, щоб показати обраний прямокутник, як він визначається.

```
 class RectangleMapTool(QgsMapToolEmitPoint):
  def __init__(self, canvas):
    self.canvas = canvas
    QgsMapToolEmitPoint.__init__(self, self.canvas)
    self.rubberBand = QgsRubberBand(self.canvas,
QgsWkbTypes.PolygonGeometry)
    self.rubberBand.setColor(Qt.red)
    self.rubberBand.setWidth(1)
    self.reset()

  def reset(self):
   self.startPoint = self.endPoint = None
    self.isEmittingPoint = False
    self.rubberBand.reset(QgsWkbTypes.PolygonGeometry)

  def canvasPressEvent(self, e):
    self.startPoint = self.toMapCoordinates(e.pos())
    self.endPoint = self.startPoint
    self.isEmittingPoint = True
    self.showRect(self.startPoint, self.endPoint)

  def canvasReleaseEvent(self, e):
    self.isEmittingPoint = False
    r = self.rectangle()
    if r is not None:
      print(«Rectangle:», r.xMinimum(),
            r.yMinimum(), r.xMaximum(), r.yMaximum()
           )

  def canvasMoveEvent(self, e):
    if not self.isEmittingPoint:
      return

    self.endPoint = self.toMapCoordinates(e.pos())
    self.showRect(self.startPoint, self.endPoint)

  def showRect(self, startPoint, endPoint):
    self.rubberBand.reset(QgsWkbTypes.PolygonGeometry)
    if startPoint.x() == endPoint.x() or startPoint.y()
== endPoint.y():
      return
```

```
    point1 = QgsPointXY(startPoint.x(), startPoint.y())
    point2 = QgsPointXY(startPoint.x(), endPoint.y())
    point3 = QgsPointXY(endPoint.x(), endPoint.y())
    point4 = QgsPointXY(endPoint.x(), startPoint.y())

    self.rubberBand.addPoint(point1, False)
    self.rubberBand.addPoint(point2, False)
    self.rubberBand.addPoint(point3, False)
    self.rubberBand.addPoint(point4, True)    # true to
update canvas
    self.rubberBand.show()

  def rectangle(self):
    if self.startPoint is None or self.endPoint is None:
      return None
    elif (self.startPoint.x() == self.endPoint.x() or \
          self.startPoint.y() == self.endPoint.y()):
      return None

      return QgsRectangle(self.startPoint, self.endPoint)

  def deactivate(self):
    QgsMapTool.deactivate(self)
    self.deactivated.emit()
```

7.16.7. Створення власних елементів карти

Ось приклад власного елемента полотна, який малює коло:

```
    class CircleCanvasItem(QgsMapCanvasItem):
      def __init__(self, canvas):
        super().__init__(canvas)
        self.center = QgsPoint(0, 0)
        self.size   = 100

      def setCenter(self, center):
        self.center = center

      def center(self):
        return self.center

      def setSize(self, size):
        self.size = size
```

```
    def size(self):
      return self.size

    def boundingRect(self):
      return QRectF(self.center.x() - self.size/2,
        self.center.y() - self.size/2,
        self.center.x() + self.size/2,
        self.center.y() + self.size/2)

    def paint(self, painter, option, widget):
      path = QPainterPath()
      path.moveTo(self.center.x(), self.center.y());
      path.arcTo(self.boundingRect(), 0.0, 360.0)
      painter.fillPath(path, QColor(«red»))

# Using the custom item:
item = CircleCanvasItem(iface.mapCanvas())
item.setCenter(QgsPointXY(200,200))
item.setSize(80)
```

7.17. Відображення карти та друк

Загалом ї два підходи, коли вхідні дані мають відтворюватися як карта: або зробити це швидко за допомогою `QgsMapRendererJob`, або отримати більш точно налаштований вихід, створивши карту за допомогою `QgsLayout` class.

7.17.1. Просте відображення

Візуалізація виконується шляхом створення `QgsMapSettings` об'єкта для визначення параметрів візуалізації, а потім створення `QgsMapRendererJob` з цими параметрами, що потім використовується для створення кінцевого зображення. Ось приклад:

```
 image_location =
os.path.join(QgsProject.instance().homePath(),
«render.png»)

 vlayer = iface.activeLayer()
 settings = QgsMapSettings()
 settings.setLayers([vlayer])
 settings.setBackgroundColor(QColor(255, 255, 255))
 settings.setOutputSize(QSize(800, 600))
```

```
 settings.setExtent(vlayer.extent())

render = QgsMapRendererParallelJob(settings)

def finished():
    img = render.renderedImage()
    # save the image; e.g.
img.save(«/Users/myuser/render.png»,»png»)
    img.save(image_location, «png»)

render.finished.connect(finished)

# Start the rendering
render.start()

# The following loop is not normally required, we
# are using it here because this is a standalone
example.
from qgis.PyQt.QtCore import QEventLoop
loop = QEventLoop()
render.finished.connect(loop.quit)
loop.exec_()
```

7.17.2. Відтворення шарів з різними системами координат

Якщо у вас шарів більше, ніж один, і вони мають різні CRS, щойно наведений простий приклад, ймовірно, не спрацює: щоб отримати правильні значення з обчислень екстентів, потрібно явно встановити цільову CRS:

```
layers = [iface.activeLayer()]
settings = QgsMapSettings()
settings.setLayers(layers)
settings.setDestinationCrs(layers[0].crs())
```

7.17.3. Виведення з використанням макета друку

Макет друку є дуже зручним інструментом, якщо ви бажаєте зробити більш складний вихід, ніж проста візуалізація. Можна створювати макети карт, що складаються з видів карт, підписів, легенд, таблиць та інших елементів, які зазвичай є на паперових картах. Потім макети можна експортувати в PDF, SVG, растрові зображення або безпосередньо роздрукувати на принтері.

Макет складається з групи класів. Усі вони належать до основної бібліотеки. Застосунок QGIS має зручний графічний інтерфейс для розміщення елементів, хоча його немає в бібліотеці графічного інтерфейсу.

Центральним класом макета є `QgsLayout` клас, який походить від класу `QGraphicsScene Qt`. Створимо його екземпляр:

```
project = QgsProject.instance()
layout = QgsPrintLayout(project)
layout.initializeDefaults()
```

Це ініціалізує макет із деякими параметрами за замовчуванням, зокрема шляхом додавання порожньої сторінки A4 до макета. Ви можете створювати макети, не викликаючи `initializeDefaults()` метод, але вам доведеться самостійно подбати про додавання сторінок до макета.

Попередній код створює «тимчасовий» макет, який не відображається в GUI. Це може бути зручно, наприклад, для того, щоби швидко додати деякі елементи та експортувати, не змінюючи сам проект і не показуючи ці зміни користувачеві. Якщо ви хочете, щоб макет зберігався/відновлювався разом із проєктом і був доступний у менеджері макетів, додайте:

```
layout.setName(«MyLayout»)
project.layoutManager().addLayout(layout)
```

Тепер ми можемо додавати різноманітні елементи (карту, напис тощо) до макета. Усі ці об’єкти представлені класами, які успадковують базовий `QgsLayoutItem` клас.

Ось опис деяких основних елементів макета, які можна додати до макета.

- map – тут ми створюємо карту нестандартного розміру та візуалізуємо поточне полотно карти:

```
map = QgsLayoutItemMap(layout)
# Set map item position and size (by default, it is a
0 width/0 height item placed at 0,0)
map.attemptMove(QgsLayoutPoint(5,5,
QgsUnitTypes.LayoutMillimeters))
```

```
map.attemptResize(QgsLayoutSize(200,200,
QgsUnitTypes.LayoutMillimeters))
# Provide an extent to render
map.zoomToExtent(iface.mapCanvas().extent())
layout.addLayoutItem(map)
```

– label – дає змогу відображати мітки. Можна змінити його шрифт, колір, вирівнювання та поля:

```
label = QgsLayoutItemLabel(layout)
label.setText(«Hello world»)
label.adjustSizeToText()
layout.addLayoutItem(label)
```

– легенда

```
legend = QgsLayoutItemLegend(layout)
legend.setLinkedMap(map) # map is an instance of
QgsLayoutItemMap
layout.addLayoutItem(legend)
```

– шкала шкали:

```
item = QgsLayoutItemScaleBar(layout)
item.setStyle('Numeric') # optionally modify the
style
item.setLinkedMap(map) # map is an instance of
QgsLayoutItemMap
item.applyDefaultSize()
layout.addLayoutItem(item)
```

– форма на основі вузлів

```
 polygon = QPolygonF()
 polygon.append(QPointF(0.0, 0.0))
 polygon.append(QPointF(100.0, 0.0))
 polygon.append(QPointF(200.0, 100.0))
 polygon.append(QPointF(100.0, 200.0))

 polygonItem = QgsLayoutItemPolygon(polygon, layout)
 layout.addLayoutItem(polygonItem)
props = {}
props[«color»] = «green»
props[«style»] = «solid»
props[«style_border»] = «solid»
props[«color_border»] = «black»
props[«width_border»] = «10.0»
```

```
props[«joinstyle»] = «miter»
symbol = QgsFillSymbol.createSimple(props)
polygonItem.setSymbol(symbol)
```

Щойно елемент додано до макета, його можна переміщувати та змінювати розмір:

```
item.attemptMove(QgsLayoutPoint(1.4, 1.8,
QgsUnitTypes.LayoutCentimeters))
item.attemptResize(QgsLayoutSize(2.8, 2.2,
QgsUnitTypes.LayoutCentimeters))
```

Навколо кожного елемента за замовчуванням намальовано рамку. Видалити його можна таким чином:

```
# for a composer label
label.setFrameEnabled(False)
```

Окрім створення елементів макета вручну, QGIS підтримує шаблони макетів, які, по суті, є композиціями з усіма їхніми елементами, збереженими у файлі .qpt (із синтаксисом XML).

Коли композиція буде готова (елементи макета створено та додано до композиції), можемо розпочати створення растрового та/або векторного виведення.

7.17.4. Перевірка правильності макета

Макет складається з набору взаємопов'язаних елементів, і може статися, що ці зв'язки порушуються під час модифікацій (легенда, пов'язана з видаленою картою, елемент зображення без вихідного файла…), або можна застосувати спеціальні обмеження до елементів макета. Досягти цього допомагає `QgsAbstractValidityCheck`. Основна перевірка має такий вигляд:

```
@check.register(type=QgsAbstractValidityCheck.Typ
eLayoutCheck)
def my_layout_check(context, feedback):
  results = ...
  return results
```

Ось перевірка, яка видає попередження щоразу, коли для елемента карти макета встановлено проєкцію веб-меркатора:

```
@check.register(type=QgsAbstractValidityCheck.TypeLayoutC
heck)
 def layout_map_crs_choice_check(context, feedback):
   layout = context.layout
   results = []
   for i in layout.items():
     if isinstance(i, QgsLayoutItemMap) and
i.crs().authid() == 'EPSG:3857':
       res = QgsValidityCheckResult()
       res.type = QgsValidityCheckResult.Warning
       res.title = 'Map projection is misleading'
      res.detailedDescription = 'The projection for the
map item {} is set to <i>Web Mercator (EPSG:3857)</i>
which misrepresents areas and shapes. Consider using an
appropriate local projection
instead.'.format(i.displayName())
      results.append(res)

  return results
```

А ось більш складний приклад, який видає попередження, якщо для будь-яких елементів карти макета встановлено значення CRS, яке є дійсним лише за межами екстенту, показаного в цьому елементі карти:

```
  @check.register(type=QgsAbstractValidityCheck.TypeLayou
  tCheck)
   def layout_map_crs_area_check(context, feedback):
       layout = context.layout
       results = []
       for i in layout.items():
           if isinstance(i, QgsLayoutItemMap):
               bounds = i.crs().bounds()
               ct =
  QgsCoordinateTransform(QgsCoordinateReferenceSystem('EP
  SG:4326'), i.crs(), QgsProject.instance())
               bounds_crs =
  ct.transformBoundingBox(bounds)
             if not bounds_crs.contains(i.extent()):
                 res = QgsValidityCheckResult()
```

```
                res.type =
QgsValidityCheckResult.Warning
               res.title = 'Map projection is
incorrect'
                res.detailedDescription = 'The
projection for the map item {} is set to \'{}\', which
is not valid for the area displayed within the
map.'.format(i.displayName(), i.crs().authid())
                results.append(res)
    return results
```

7.17.5. Експорт макета

Для того щоби експортувати макет, `QgsLayoutExporter` потрібно використовувати клас:

```
base_path =
os.path.join(QgsProject.instance().homePath())
pdf_path = os.path.join(base_path, «output.pdf»)
exporter = QgsLayoutExporter(layout)
exporter.exportToPdf(pdf_path,
QgsLayoutExporter.PdfExportSettings())
```

Використовуйте `exportToSvg()` або `exportToImage()`, якщо хочете експортувати відповідно у файл SVG або зображення замість файла PDF.

7.17.6. Експорт атласа макета

Якщо ви хочете експортувати всі сторінки з макета, для якого налаштовано та ввімкнено параметр атласа, вам потрібно використати метод `atlas()` в експортері (`QgsLayoutExporter`) з невеликими налаштуваннями. У наведеному далі прикладі сторінки експортовано в зображення PNG:

```
exporter.exportToImage(layout.atlas(), base_path,
'png', QgsLayoutExporter.ImageExportSettings())
```

Зауважте, що результати будуть збережені в папці основного шляху з використанням виразу імені вихідного файла, налаштованого в atlas.

8. ЗАСТОСУВАННЯ PYTHON У ARCGIS

8.1. Бібліотека ArcPy

ArcPy – це бібліотека Python (іноді називається ArcPy site-package), що дає доступ з Python до всіх інструментів геообробки, зокрема до додаткових модулів, а також пропонує велику кількість корисних функцій і класів для роботи з даними ГІС. Використовуючи Python і ArcPy, можна розробляти велику кількість зручних програм для роботи з географічними даними.

Модуль ArcPy – це файл Python, який містить функції і класи. ArcPy підтримується різними модулями, зокрема таким як модуль доступу до даних (arcpy.da), модуль картографії (arcpy.mp), додатковий модуль ArcGIS Spatial Analyst (arcpy.sa) і додатковий модуль ArcGIS Network Analyst (arcpy.na).

Додатки та скрипти ArcGIS написані за допомогою ArcPy, що дає змогу отримати доступ до численних модулів Python, розроблених користувачами ГІС і програмістами, що працюють в різних галузях. Ще одна перевага використання ArcPy в середовищі Python полягає в тому, що Python є універсальною мовою програмування, яку легко освоїти. Це мова, що інтерпретується з динамічною типізацією, що дає змогу швидко моделювати і перевіряти скрипти в інтерактивному середовищі і разом з тим уможливлює написання великих застосунків.

З технічного погляду інструменти геообробки являють собою функції з arcpy, тобто доступ до них здійснюється так само, як і до будь-яких інших функцій Python. Проте, щоб уникнути плутанини між інструментами і відмінними від них функціями (такими як службові функції на зразок ListFeatureClasses()), беруть до уваги:

1. Інструменти документуються не так, як функції. У кожного інструмента є власна довідкова сторінка в довідковій системі. Функції документуються в документації ArcPy.
2. Інструменти, на відміну від функцій, повертають об'єкт Result.
3. Інструменти створюють повідомлення, до яких можна звертатися за допомогою багатьох функцій, таких як GetMessages (). Функції не створюють повідомлень.

8.2. ArcToolbox

ArcGIS надає великий набір інструментів для обробки даних. Панель ArcToolbox в ArcCatalog (і ArcMap) містить набори інструментів (toolboxes) – 3D Analyst, Analysis, Cartography та ін. (рис. 8.1). Інструменти (tools) згруповані в набори інструментів за типом дій, які вони виконують, і кожен набір інструментів містить групи інструментів (toolsets), що додатково групують інструменти за їхньою функціональністю. Кожен toolbox і toolset може бути розгорнутий для демонстрації вмісту. На рис. 8.2 наведено групи інструментів Extract, Overlay, Proximity, і Statistics в Analysis toolbox. На цьому рисунку група інструментів Proximity також розширюється.

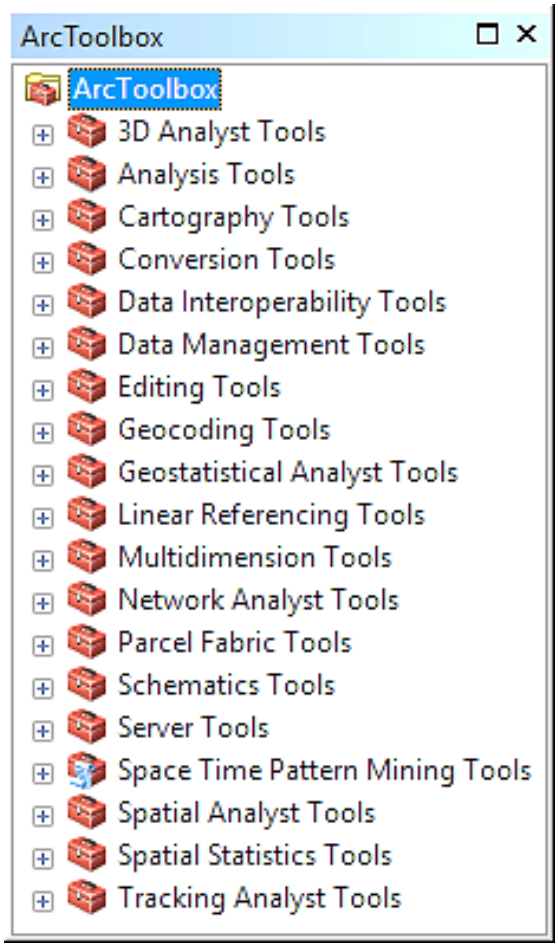

Рис. 8.1. Застосунок ArcToolbox з усіма наборами інструментів

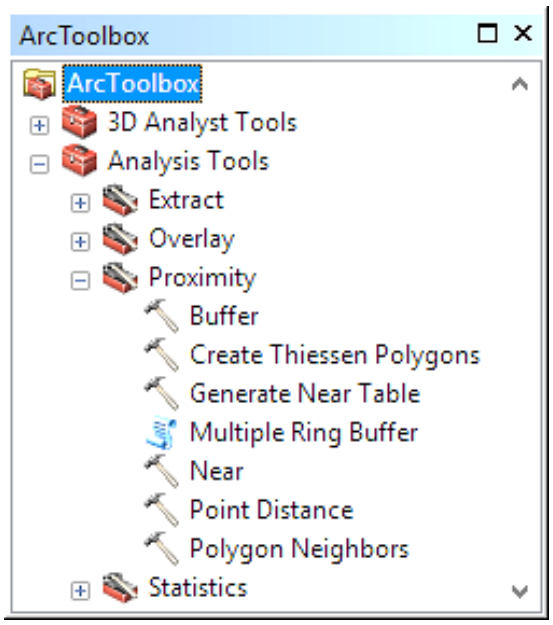

Рис. 8.2. Набір інструментів Analyst і група інструментів Proximity

Ця група інструментів містить шість інструментів, Буфер (Buffer), Створити полігони Тіссена (Create Thiessen Polygons) та ін. Хоча піктограми змінюються на третьому рівні, це все інструменти. Інструмент може бути запущений з ArcToolbox вибором інструмента, заповненням форми, яку він запускає, і натисканням кнопки «OK».

Доступність інструментів залежить від рівня ліцензії встановленого ArcGIS Desktop (базовий, стандартний або просунутий). Деякі інструменти, такі як Spatial Analyst, не доступні на базовому і стандартному рівнях. Коли ці інструменти доступні, розширення потрібно підтвердити для інструментів, які стають функціональними. Наприклад, щоб використовувати інструменти Spatial Analyst з розширеним встановленням, розширення Spatial Analyst має бути перевірене (ArcCatalog > Customize > Extensions > Check Spatial Analyst). Скрипти повинні також перевірити ці розширення, як показано далі. Панель Search (Пошук), доступну в ArcCatalog (і ArcMap), можна використовувати для навігації по інструментах. Натисніть на кнопку Search (або скористайтеся комбінацією клавіш Ctrl + F), щоб відкрити цю панель, оберіть опцію Tools і введіть назву інструмента.

На рис. 8.3 наведено результати пошуку. Натисніть на інструмент в результатах (наприклад, Buffer (Analysis) Tool), і це відкриє графічний інтерфейс для запуску інструмента. Форма містить кнопку Tool Help, яка запустить локальну довідку ArcGIS Desktop для цього інструмента.

Працюючи з прикладами в цій книзі, ви можете знайти інструменти, ознайомитися з їхніми функціональними можливостями, читаючи довідку, і запускати їх за допомогою графічного інтерфейсу.

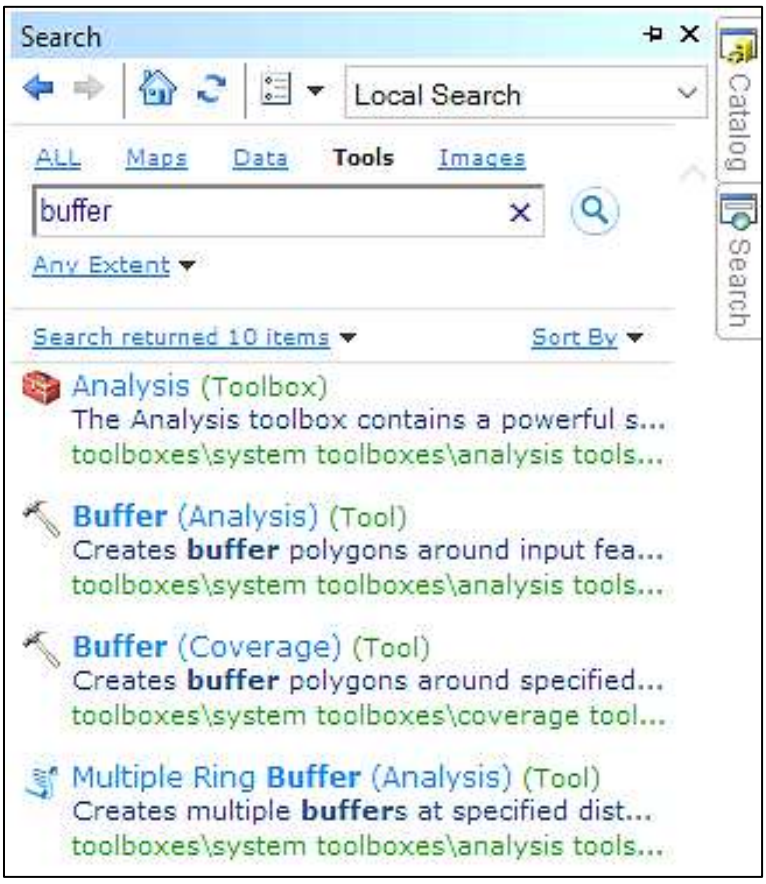

Рис. 8.3. Панель пошуку

8.3. Ресурси ArcGIS Python

Сайт ArcGIS Resources (resources.arcgis.com) є основним ресурсом, який вам знадобиться для роботи з Python в ArcGIS. Сайт надає повну документацію про функціональність ArcGIS Python. Обов'язковою умовою для роботи з Python в ArcGIS є використання рядка «Search ArcGIS Resources». Використовуйте рядок пошуку, щоб отримати список сторінок, відповідних вашому запиту, і використовуйте фільтри пошуку, щоб звузити область пошуку (рис. 8.4). Наприклад, введіть «buffer» у вікні пошуку. Це повене тисячі результатів, зокрема блоги, повідомлення про помилки, веб-відображення допомоги та ін. Звузити пошук можна з допомогою фільтрів «Help» та «Desktop», як показано на рис. 8.5.

Рис. 8.4. Пошук довідки у вікні пошуку ArcGIS Resource

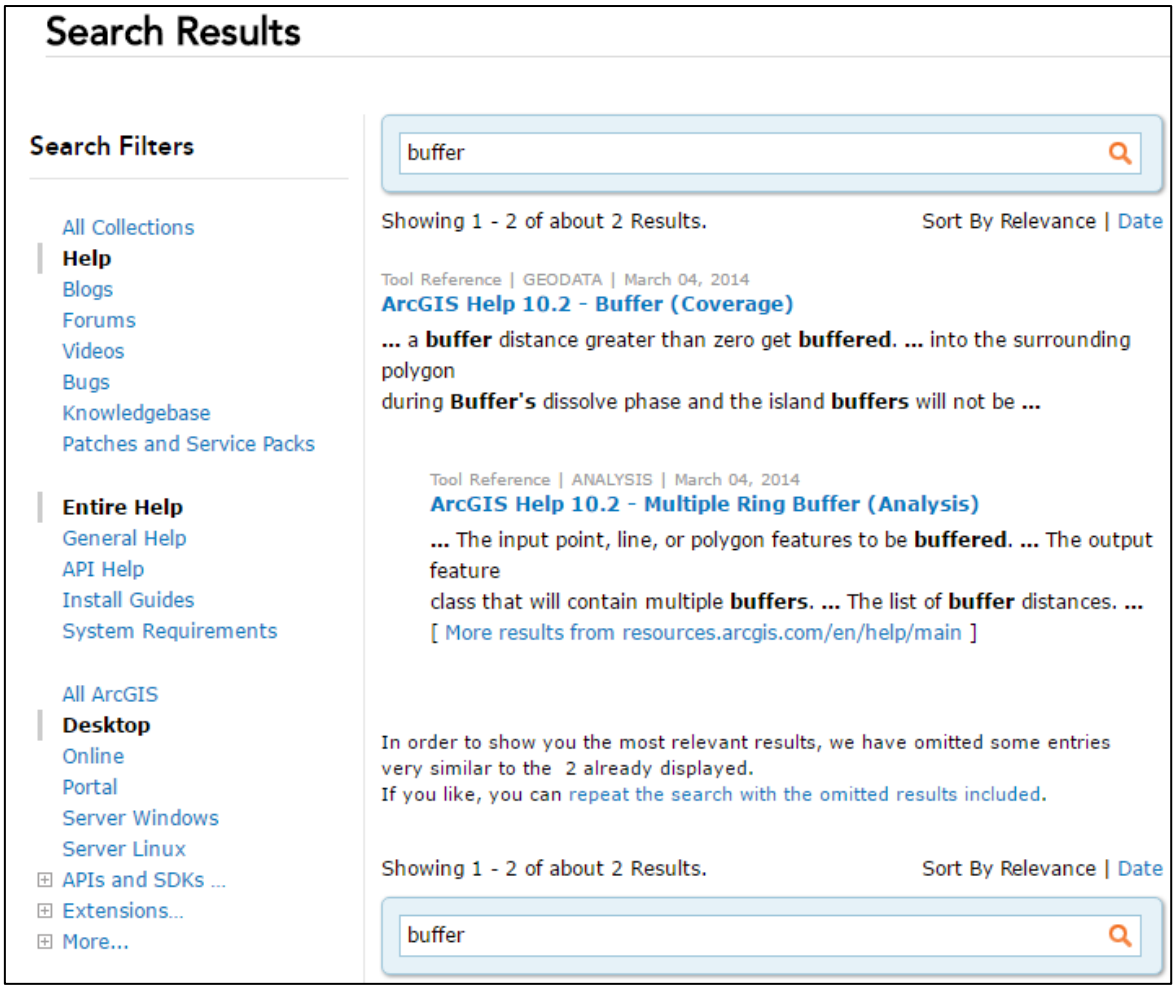

Рис. 8.5. Фільтр пошуку релевантних результатів. Продемонстрований тут пошук повертає тільки «Help» за темою «ArcGIS Desktop».

Зверніть увагу на те, що результати відрізняються від результатів пошуку ArcCatalog для того самого терміну; допомога ArcGIS Desktop організована інакше, ніж в інтерактивній довідці. Інтерактивна довідка містить найактуальнішу, повну документацію. Набір описових ідентифікаторів передбачено для кожного посилання в інтерактивній довідці. Посилання Buffer (Analysis) має ідентифікатори Tool Reference і Analysis (рис. 8.6). Останній ідентифікатор відображає дату останньої модифікації контенту. Кожен інструмент ArcGIS має сторінку Tool Reference, відповідний вбудованій допомозі для цього інструмента.

Посилання на цей сайт називається ArcGIS Resources. Ключові слова для пошуку направлятимуть вас до конкретних розділів довідки на даному сайті.

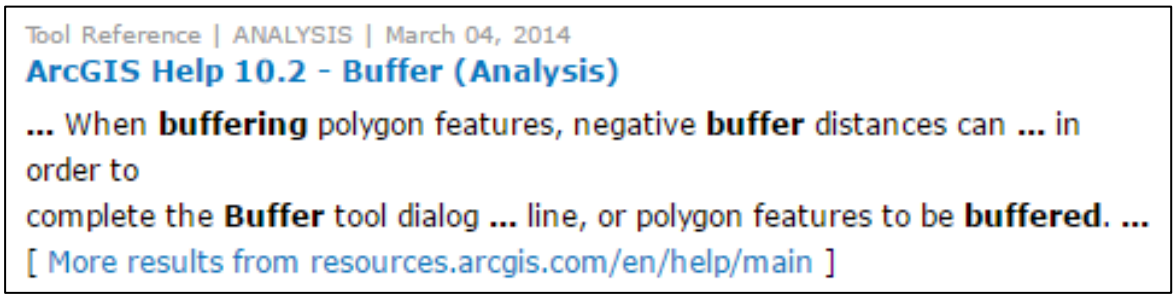

Рис. 8.6. Ключові слова «Tool reference» та «ANALYSIS» з'являються над кожним результатом пошуку

8.4. Експорт моделей

За допомогою Python можна викликати практично всі інструменти в ArcToolbox, це дає змогу автоматизувати повторювані процеси. Перед тим як почати писати скрипти з нуля, розглянемо приклад, який автоматично генерується застосунком ArcGIS ModelBuilder.

ModelBuilder – це вбудований в ArcCatalog (і ArcMap) застосунок, що дає користувачам змогу створювати візуалізацію робочого потоку (workflow), що називається *моделлю*. Моделі не лише візуалізують роботу потоку, але також можуть бути запущені для виконання робочого потоку. Інструменти ArcGIS Toolbox можуть бути запущені також за допомогою ModelBuilder; інструменти можуть бути перетягнуті на панель моделі і під'єднані для створення роботи потоку. Після запуску моделт [вона] виконує інструменти і вирази, що лежать в основі коду і відповідні частинам моделі. Базовий код може бути експортований в скрипт Python, тоді можна порівняти візуалізацію роботи потоку з кодом. Виконайте кроки 1-3, щоб створити і експортувати просту модель.

1. Виконайте дії у ArcCatalog, щоб створити модель, як на рис. 8.7:

1.1. Запустіть ModelBuilder за допомогою кнопки  на панелі інструментів Standard.

1.2. Відкрийте ArcToolbox за допомогою кнопки  ArcToolbox на панелі інструментів Standard.

1.3. Знайдіть інструмент Buffer (Analysis) в ArcToolbox (ArcToolbox > Analysis Tools > Proximity > Buffer)

1.4. Виберіть інструмент Buffer і перетягніть його в ModelBuilder з набору інструментів в ArcToolbox. З'явиться прямокутник з написом Buffer.

1.5. Клацніть правою кнопкою миші на новому прямокутнику Buffer > Make variable > From Parameter > Input Features.

1.6. Знову клацніть правою кнопкою миші на прямокутнику Buffer > Make variable > From Parameter > Distance.

1.7. Двічі клацніть по овалу Input Features і перейдіть до: «D:/gispy/data/ch05/park.shp».

1.8. Двічі клацніть на овалі Distance і встановіть значення 100 футів.

1.9. Клацніть правою кнопкою миші по овалу Input Features > перейменуйте на «inputFeatures».

1.10. Клацніть правою кнопкою миші по овалу Distance > перейменуйте на «distance».

1.11. Клацніть правою кнопкою миші по овалу Output Feature Class > перейменуйте на «outputFeatures».

1.12. Переконайтеся, що результат має вигляд, як на рис 8.7.

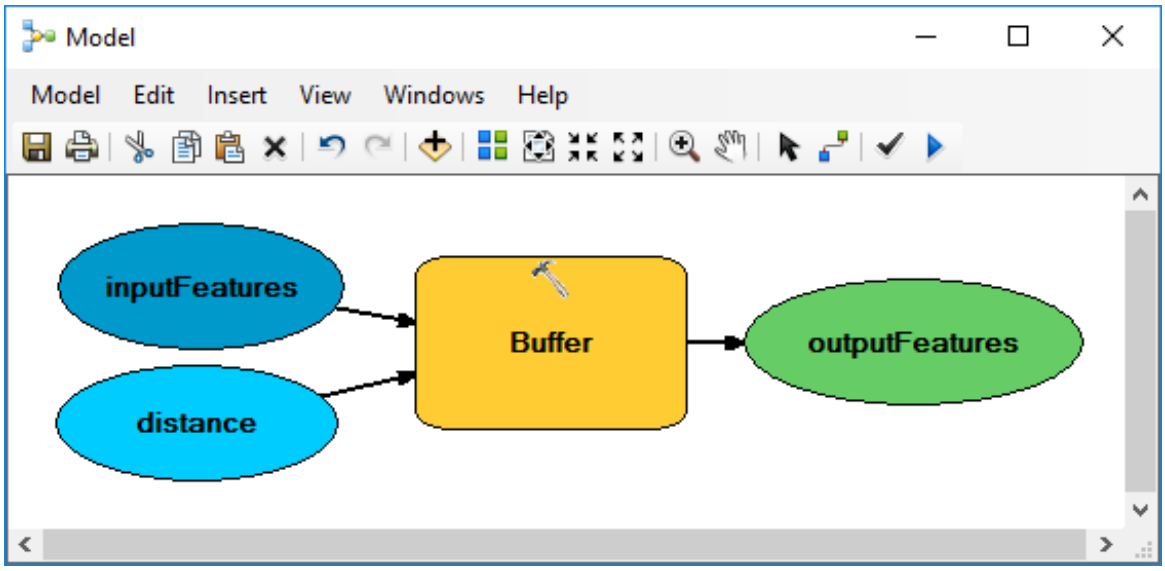

Рис. 8.7. Модель ModelBuilder буферизації вхідних об’єктів.

2. Запустіть модель, щоб переконатися, що вона працює (меню Model > Run).

3. Експортуйте модель як скрипт (Model > Export > Pythons Script). Це повинно створити скрипт, як на рис. 8.8. Відкрийте сценарій в IDLE для перегляду коду.

```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# arcpymodule.py
# Created on: 2016-11-06 14:53:37.00000
#   (generated by ArcGIS/ModelBuilder)
# Description:
# ---------------------------------------------------------------------------

# Import arcpy module
import arcpy


# Local variables:
inputFeatures = "D:\\gispy\\data\\ch05\\park.shp"
distance = "100 Feet"
outputFeatures = "C:\\Users\\Family\\Documents\\ArcGIS\\Default.gdb\\park_Buffer"

# Process: Buffer
arcpy.Buffer_analysis(inputFeatures, outputFeatures, distance, "FULL", "ROUND", "NONE", "", "PLANAR")
```

Рис. 8.8. Експортований з наведеної вище моделі скрипт.

Для порівняння моделі і скрипта переглянемо кожен рядок коду.

1. Рядки 1-7, 9, 13 і 18 це коментарі.
2. Рядок 10 імпортує arcpy. Цей рядок коду дає змогу використовувати у скрипті команди ArcGIS.
3. Рядки 14-16 містять оператори надання для рядкових змінних, *inputFeatures, distance і outputFeatures*. Вони відповідні змінним моделі, овалам, які визначають вхід і вихід для інструмента. Рядковий літерал, що надається змінним скрипта, залежить від значення, наданого в ModelBuilder. Наприклад, в рядку 13 inputFeatures в цей час надається значення «*D:\\gispy\\data\\ch05\\park.shp*», оскільки змінній моделі було надане це значення до того, як модель була експортована.
4. Рядок 19 викликає інструмент Buffer (Analysis). Запуск інструмента в Python має вигляд як виклик функції. Ми викликаємо інструмент і передаємо йому аргументи. Інструмент виконує нашу вказівку і створює вихідний об’єкт або повертає значення. Змінні і рядкові літерали в дужках передають інформацію до інструмента. Інструмент Buffer потребує трьох вхідних параметрів (інші параметри не є обов'язковими). Ці обов’язкові параметри представлені в наведеній

моделі овалами. Перші три аргументи у Python відповідні цим параметрам. Експортований код був заповнений значеннями за замовчуванням для інших параметрів. Таким чином, лінія 19 діє як прямокутник в моделі; вона створює буфер навколо вхідних об’єктів на задану відстань і зберігає результати у вихідних об’єктах.

За допомогою цих спостережень ви можете зауважити зв'язок між компонентами моделі і рядками коду в сценарії. Теоретично експортовані моделі можна було б використовувати як відправну точку для скриптів, але цей підхід може бути громіздким з кількох причин. По-перше, скрипти дають змогу створити більш гнучкі, повторно використовувані робочі потоки, зокрема за межами функціональних можливостей геообробки ArcGIS. По-друге, експортовані скрипти звичайно потребують модифікації для виконання за бажанням, роблячи його більш продуктивним для модифікації зразків коду, ніж для створення та експорту моделей.

8.5. Робота в модулі ArcCatalog

Щоразу, запускаючи скрипт на Python, який генерує на виході геообробку, ви захочете перевірити результати. Об’єкти географічних даних Esri і пов'язані з ними таблиці можна переглянути в ArcCatalog на вкладці «Preview». Перейдіть до шейп-файла в ArcCatalog у «Catalog Tree» й оберіть вкладку «Preview». Оберіть перегляд «Geography» в нижній частині панелі попереднього перегляду, щоб переглянути географічні об’єкти. Потім оберіть «Table», щоб побачити відповідну таблицю атрибутів. На рис. 8.9 і 8.10 показано географічне і табличне відображення «park.shp». Тільки сім рядків таблиці наведено на рис. 8.10. В цілому у таблиці міститься 426 рядків, по одному запису даних для кожного полігону.

Коли дані переглядаються в ArcCatalog, він блокується, так що інші програми не можуть їх змінювати. Файл з розширенням «*sr.lock*» з'являється в провіднику Windows, коли дані заблоковані. Наприклад, файл може мати назву на зразок «*park.shp.HAL.5532.5620.sr.lock*» на комп'ютері названий «Hal». Перед обробкою файла в сценарії Python вам слід переконатися, що дані не заблоковані. Для того щоби розблокувати дані після їхнього перегляду в ArcCatalog, оберіть робочу область («*C:/gispy/data/CH05*» на рис. 8.9 і 8.10) й оновіть ArcCatalog (натисніть

F5). Вибір іншого файла в межах одного робочого простору і поновлення Catalog Tree не розблокує замок; робоча область повинна бути оновлена.

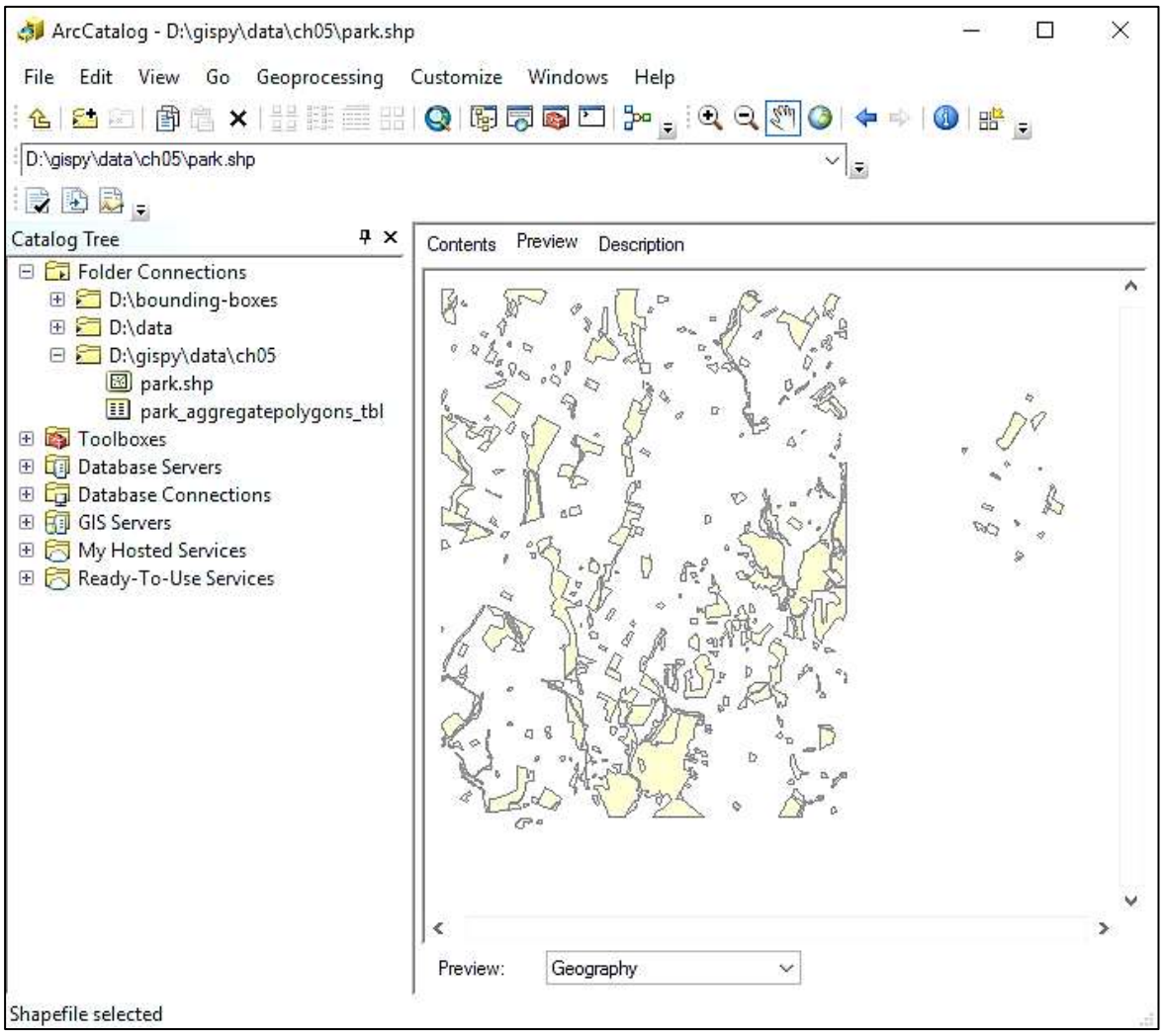

Рис. 8.9. Географічний попередній перегляд «park.shp»

Примітка. Вирішальне значення перед проведенням геообробки Python має розблокувати дані, інакше скрипт може згенерувати непередбачені помилки.

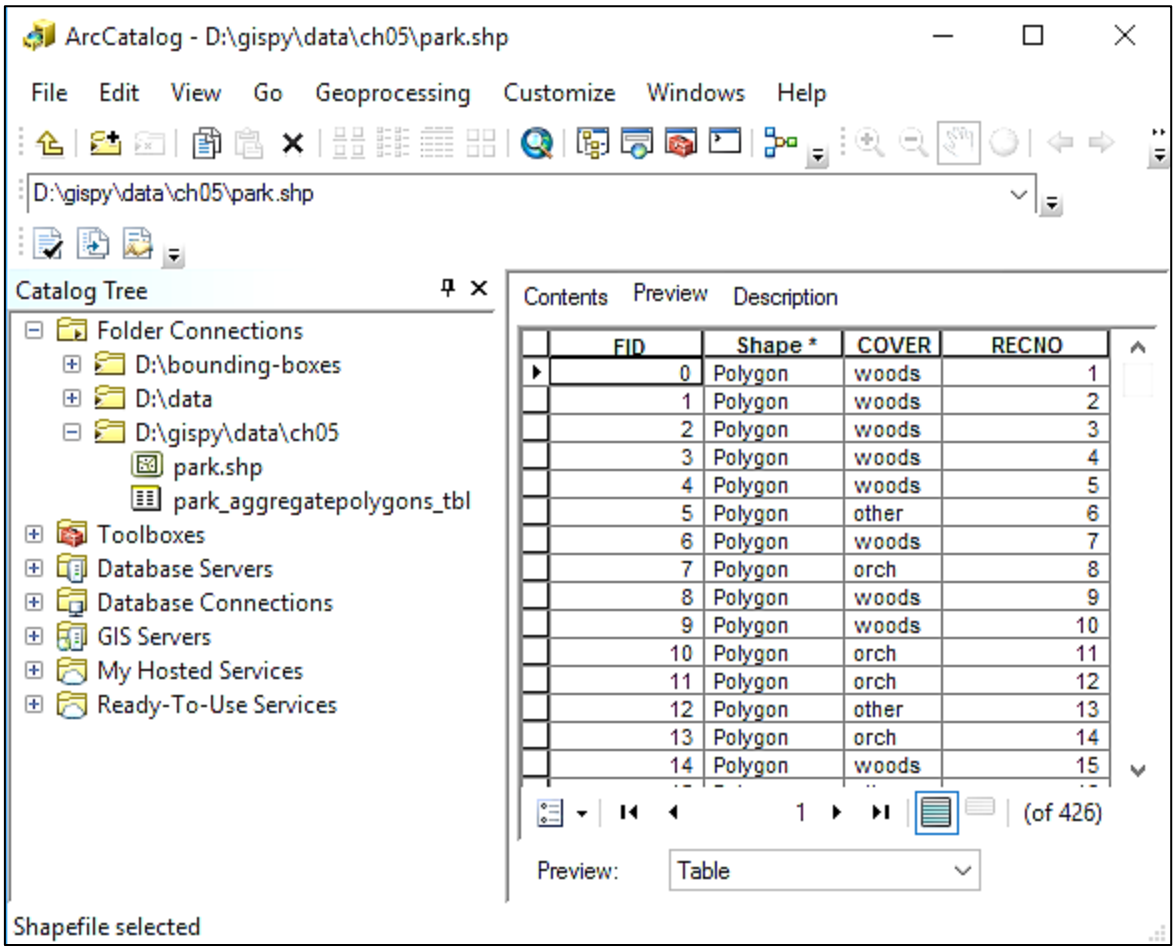

Рис. 8.10. Табличний попередній перегляд «park.shp»

Для того щоби побачити географічне представлення більш ніж одного файла за раз, потрібно використовувати ArcMap. Щоби переглянути дані в ArcMap, найпростіше перейти до даних у вбудованому в ArcMap дереві ArcCatalog і перетягнути їх на порожню карту. Доти, доки ArcMap працює, дані, які використовуються таким чином, будуть заблоковані. Навіть якщо документ карти закритий, блокування не може бути зняте, поки програма не буде закритою.

8.6. Робота з пакетом arcpy у середовищі ArcGIS

Тепер, коли ви обазнані з ArcToolbox, ModelBuilder і переглядом даних в ArcCatalog, настав час представити пакет arcpy. Скрипти геообробки починають шляхом імпорту arcpy (див., наприклад, рис. 8.8); Пакет arcpy є засобом мови Python для доступу до ArcGIS з Python. Для того щоби використовувати функціональні можливості ArcGIS в Python,

скрипт повинен імпортувати arcpy. Терміни «import», «пакет» і «arcpy» пояснюються тут:

1. Ключове слово «import» вживають для посилання на модуль або пакет. Це потрібно для доступу до функціональних можливостей за межами вбудованих функцій Python. Термін, який стоїть після ключового слова «import», є модулем або пакетом.

2. Нагадаємо, що *модуль* – це єдиний скрипт Python (файл '.py'), що містить тісно пов'язані визначення і вирази. Пакет являє собою особливий спосіб структурування набору пов'язаних модулів Python. Пакет – це каталог, що містить модулі та, іноді, підпакети, які також містять модулі. Модуль з назвою '*__init__.py*' повідомляє Python, що каталог містить пакет. Модулі структуровані в рамках пакету і елементи з цих модулів можна отримати за допомогою точкової нотації.

3. Arcpy – це пакет, що встановлюється разом з ArcGIS. Arcpy також можна розглядати як об'єкт Python, який має велику кількість методів, зокрема інструменти геообробки. Переглянувши директорію, де встановлено ArcGIS, ви знайдете каталог arcpy. Цей пакет потребує імпорту. Arcpy надає значну кількість функціональних можливостей, які ми будемо використовувати. У табл. 8.1 наведено основні моменти arcpy.

Таблиця 8.1

Основна функціональність arcpy

Розділи ArcGIS	Функціональність	Приклад коду
Tools	Виклик інструментів ArcToolbox	arcpy.Buffer_analysis('park.shp', 'output.shp', '1 Mile')
Інші функції	Ліцензування, керування даними, а також інші різні потреби	arcpy.CheckOutExtension('Spatial')
Environment variables	Встановлення і отримання значень змінних середовищ	arcpy.env.overwriteOutput = True
Describe	Опис властивостей даних	arcpy.Describe('park.shp')
Listing data	Елементи списку (наприклад, растри, класи об’єктів і робочі простори)	arcpy.ListFeatureClasses()

Закінчення табл. 8.1

Розділи ArcGIS	Функціональність	Приклад коду
Cursor	Читання/модифікація елементів атрибутивної таблиці	arcpy.da.SearchCursor('park.shp', fieldNames)
Messaging	Отримання і друк повідомлень геообробки	arcpy.GetMessages()
Mapping	Маніпулювання шарами карт, додавання шарів карт, модифікування оточення, маніпулювання символікою	arcpy.mapping.MoveLayer(df, refLayer, moveLayer, 'BEFORE')

Пакет arcpy має об'єктно орієнтований дизайн, який може бути визначений з використанням об'єктно орієнтованого підходу.

Все у Python є *об'єктом*, зокрема модулі і пакети. Arcpy – це об'єкт. Arcpy також використовує об'єкти, такі як ValueTable, Describe і Result.

Об'єкти містять пов'язані з ними методи. Метод – це функція, яка виконує деяку дію на об'єкті. Методи просто є специфічним типом функцій. Терміни «методи, що викликають», «передавання аргументів» і «повернення значень» застосовують до методів так само, як і до функцій.

Точкова нотація створює контекстне меню для arcpy, показуючи список доступних методів і властивостей arcpy. У разі використання точкової нотації з рядками і списками контекстне меню з'являється автоматично, але це вбудовані типи даних. Для того щоб переглянути контекстне меню для arcpy, потрібно спочатку імпортувати arcpy, даючи Python доступ до інформації, яку він використовує для заповнення контекстного меню. Іншими словами, він розпізнає слово arcpy як пакет. Після того як ви імпортуєте arcpy, в рамках сеансу PythonWin контекстне меню стане доступним для всієї сесії. Скористайтеся кодом, відображенимна скріншоті, щоб побачити контекстне меню для властивостей arcpy, які чутливі до регістру і повинні бути написані правильно. Вибір варіанта з контекстного меню гарантує точну орфографію і регістр. Щоб скористатися контекстним меню, ви можете почати вводити назву, і меню буде прокручуватися до найближчого

варіанта. Натисніть клавішу «Tab» щоб вставити поточний вибір в код. Якщо ви не бачите свого вибору, то це означає, що сталася орфографічна помилка чи неправильним є регістр. Меню arcpy містить список функцій arcpy, параметри середовища, інструменти і модулі (рис. 8.11).

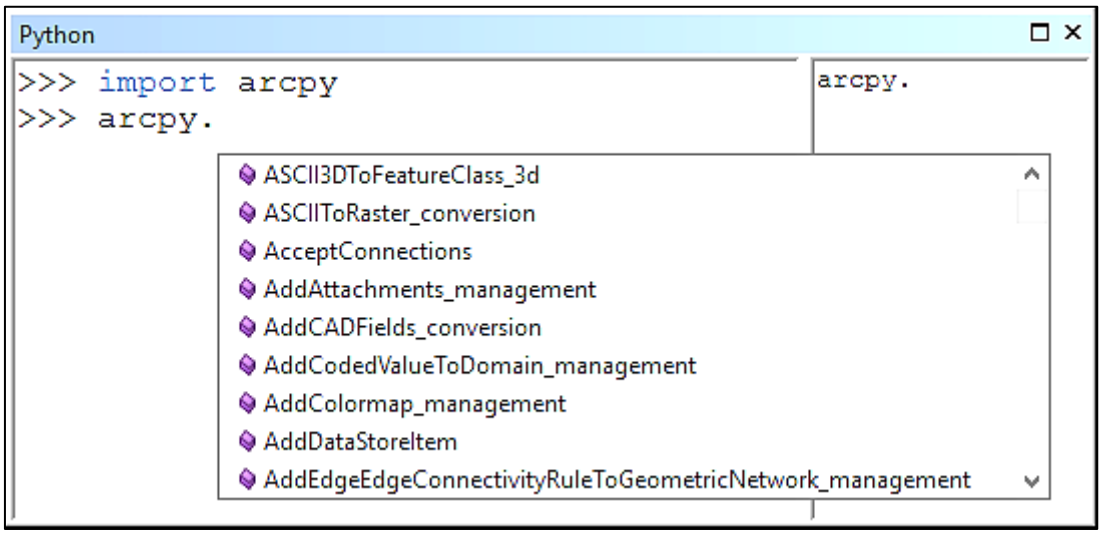

Рис. 8.11. Меню arcpy в середовищі ArcGIS

З кожним випуском програмного забезпечення функціональність arcpy зростає. Було б важко вивчати кожен об'єкт, метод чи властивість arcpy. Насправді вам не потрібно знати всіх деталей; вони можуть бути знайдені в довідковій документації. Замість цього ми прагнемо до загального огляду наявних можливостей. На рис. 8.12 представлено огляд функціональних можливостей arcpy. Символи використовуються для властивостей, методів і об'єктів, як показано в ключі (внизу праворуч). У цій скороченій діаграмі об'єктної моделі коробки містять функціональні категорії і лише кілька прикладів наведених для кожної категорії. Вміст не є вичерпним, але рис 8.12 слугуватиме основою для обговорення arcpy. Пакети, модулі і класи є конструктами Python для організації коду. Ці конструкції можуть містити функції (або методи) і властивості. Щоб переглянути повний список функцій/методів і властивостей для будь-якої з цих конструкцій, скористайтеся пошуком на сайті ArcGIS Resources.

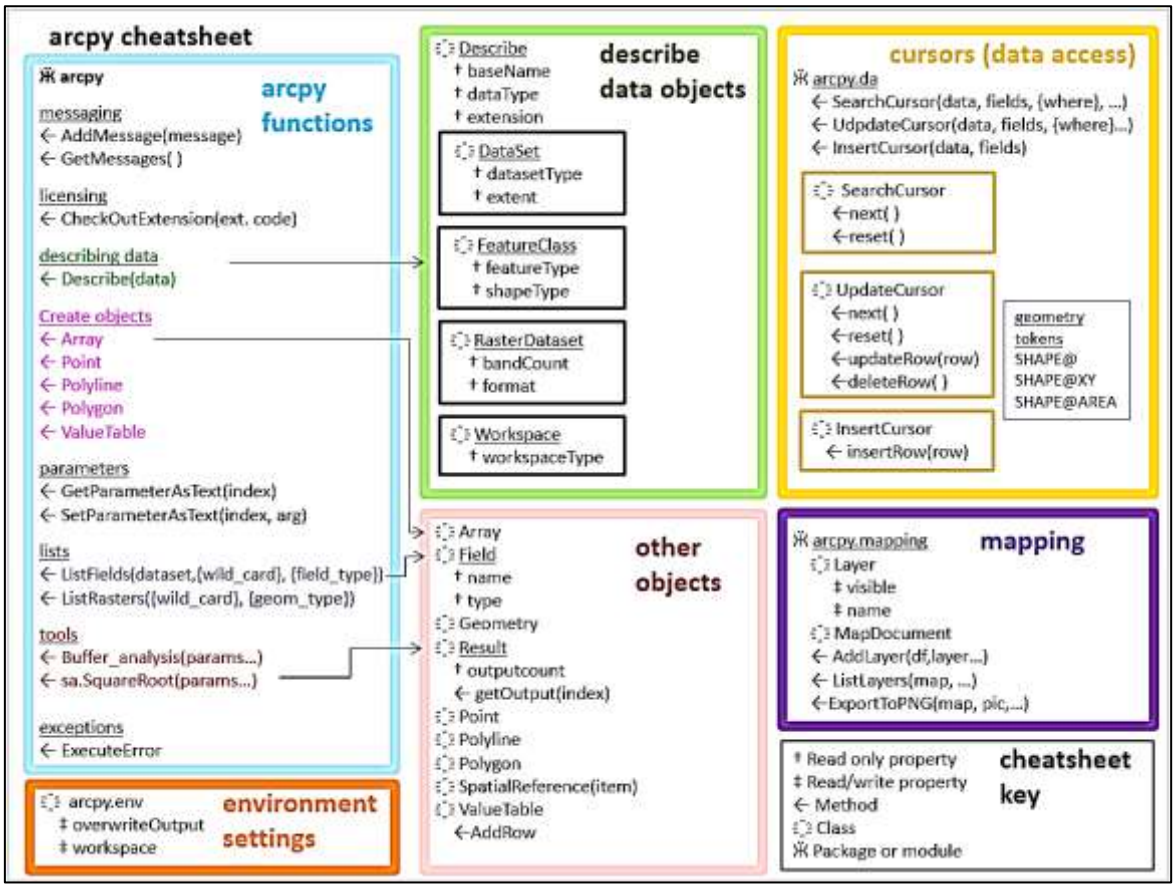

Рис. 8.12. Основні компоненти функціональності arcpy
з кількома прикладами, наведеними для кожного елемента

8.7. Функції arcpy

Функції arcpy (верхня ліва рамка на рис. 8.12) підтримують робочі потоки геообробки. Наприклад, функції можна використовувати для відображення наборів даних, отримання властивостей набору даних, перевірки існування даних, перевірки імені таблиці перед її додаванням в базу геоданих або виконувати безліч інших корисних скриптових завдань. Синтаксис для виклику функцій в пакеті arcpy використовує точкову нотацію з arcpy перед точкою та ім'ям функції після точки:

arcpy.functionName(argument1, argument2, argument3,...)

У наступному прикладі функція «*Check Extension*» приймає один аргумент, код розширення «3D», для 3D Analyst toolbox:

```
>>> import arcpy
>>> arcpy.CheckExtension('3D')
```

Відповідь означає, що ліцензія 3D Analyst Extension доступна для перевірки. Літера «U» на початку рядка означає як «Unicode» – спосіб кодування рядків. Але кодування рядка не має для нас жодних практичних наслідків, так що можна це проігнорувати. Якщо ви маєте намір надрукувати значення за допомогою вбудованої функції друку, u не додаватиметься.

```
>>> print arcpy.CheckExtension('3D')
   Available
```

Зверніть увагу, що в наведеному прикладі arcpy не імпортувався. Якщо arcpy імпортувався в цій сесії, його не потрібно імпортувати ще раз, але якщо сесія Python була закрита і знову відкрита, arcpy треба імпортувати ще раз.

Навіть якщо функції не потребують аргументів, для них як і раніше, потрібно використовувати круглі дужки. Наприклад, функція *ListPrinterNames* повертає список доступних для виклику принтерів і не приймає жодного аргументу:

```
>>> arcpy.ListPrinterNames()
[u'\u041d\u0430\u0434\u0456\u0441\u043b\u0430\u0442\u0438
\u0434\u043e OneNote 2013', u'PDF-XChange 5.0 for ABBYY',
u'Microsoft XPS Document Writer', u'Microsoft Print to
PDF', u'MapInfo PDF Printer Version 12.5', u'Fax']
```

Багато функцій arcpy повертають значення. Оператор надання може надати змінній значення, що повертається. Наступний код створює arcpy об'єкт Point. Функція *CreateObject* повертає об'єкт Point, і він зберігається в змінній з ім'ям pt. Другий рядок виводить значення цієї змінної, двовимірний об'єкт Point, що знаходиться в (0,0):

```
>>> pt = arcpy.CreateObject('Point')
>>> pt
<Point (0.0, 0.0, #, #)>
```

Ці arcpy функції слугують різним потребам скриптів, деякі пов'язані з адміністративними проблемами, такими як ліцензування (наприклад, *CheckExtension*), інші, що стосуються керування даними. Наприклад, функція Exists приймає один аргумент, набір даних і перевіряє, чи він існує:

```
>>> arcpy.Exists('D:/gispy/data/ch05/park.shp')
True
```

Інші функції стосуються таких питань, як керування базами геоданих, обмін повідомленнями, поля, інструменти, параметри та курсори. Виконайте пошук за «alphabetical list of arcpy functions», на сайті ArcGIS Resources, щоб побачити повний список функцій. Технічно функції інструментів ArcToolbox – це функції arcpy, але вони окремо перераховані в іншому місці на сайті. Синтаксис для виклику інструментів схожий на синтаксис виклику інших функцій arcpy. Перед тим як навчитися викликати інструменти, потрібно трохи дізнатися про управління параметрами середовища. Синтаксис налаштувань середовища використовує звичайну точкову нотацію.

8.8. Параметри середовища ArcMap

Кожен інструмент має налаштування, які він використовує під час виконаня операції, наприклад, допуску або вихідної директорії. *Параметри середовища* – це умови, які можуть бути застосовані до всіх інструментів всередині програми (наприклад, всі операції в ArcCatalog можуть бути обмежені до міри, встановленої в параметрах середовища). ArcGIS має значення за замовчуванням для цих установок і користувачі можуть змінювати значення за допомогою діалогового вікна в ArcGIS (Geoprocessing > Environments запускає діалогове вікно, показане на рис. 8.13).

Команди Python також можна використовувати для отримання або встановлення значення цих параметрів. Arcpy містить клас env, спеціальну структуру для збереження пов'язаних властивостей. Властивості env управління параметрами навколишнього середовища. Env належить до arcpy і властивості належать до класу env, тому імена властивостей мають дві точки: одну після arcpy й одну після env. Формат для налаштування цих властивостей:

arcpy.env.property = value

Формат для отримання цих властивостей:

variable = arcpy.env.property

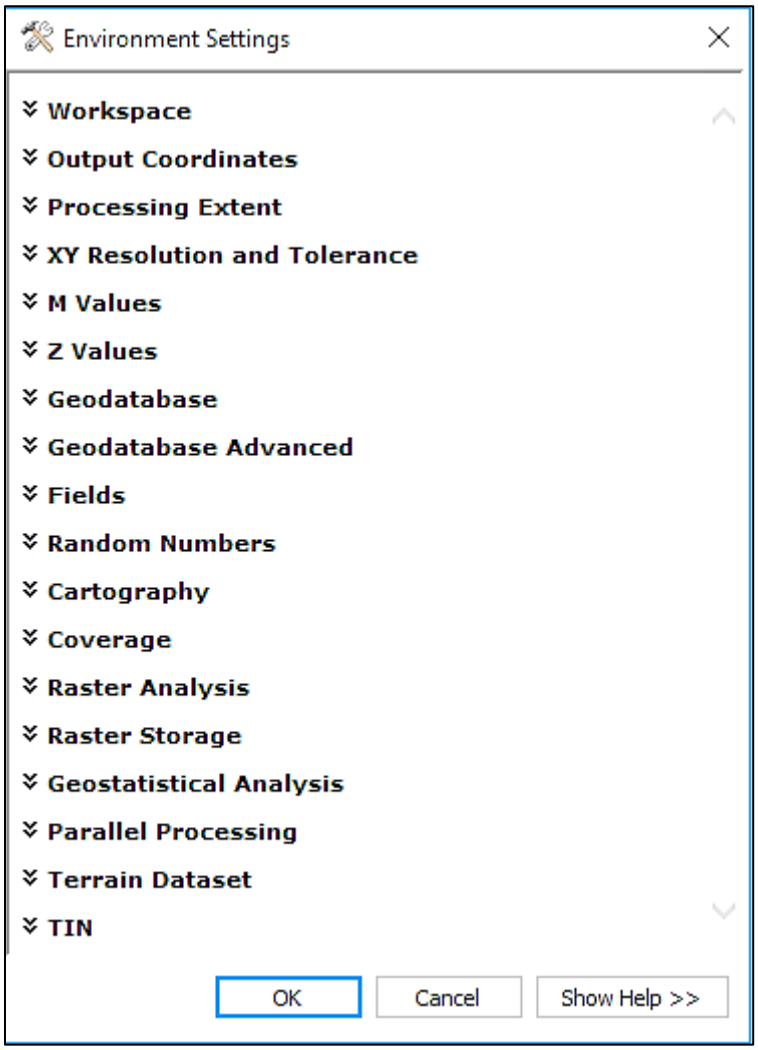

Рис. 8.13. Інтерфейс для зміни параметрів середовища ArcMap

Шлях до робочого простору і статус запису виведення є важливими властивостями середовища. Шлях до робочого простору визначає структуру, таку як каталог або файл геоданих, який містить відповідні дані. Інструменти шукають вхідні дані в робочій області і розміщують результат в робочій області. Якщо вхідні дані знаходяться в робочому просторі, використовуватиметься лише ім’я файла. Arcpy буде автоматично здійснювати пошук в робочому просторі. Крім того, робочий простір – це розміщення за замовчуванням для виведення, якщо не вказано інше.

```
>>> # Setting the current workspace path
>>> arcpy.env.workspace = 'D:/Data/Forestry'
>>> # Getting the current workspace path
```

```
>>> mydir = arcpy.env.workspace
>>> mydir
u'D:/Data/Forestry'
```

Налаштування робочого простору здійснюється наданням рядка. Перевірки на наявність робочого простору в цій точці не виконується. Це відбувається лише тоді, коли інструмент намагається використати робочий простір для читання або запису даних. Статус записування виходу, True або False, контролює можливість запису з інструментів наявних або ні файлів. Вбудовані константи True або False потрібно писати з великої літери. Статус запису виходу може бути встановлений як 1 або 0 (1 для True і 0 для False). Значення за замовчуванням для властивості overwriteOutput False:

```
>>> arcpy.env.overwriteOutput
False
```

Це захищає користувача від ненавмисного перезапису файла, але є незручним в процесі розроблення програмного забезпечення, коли скрипти повинні бути запущені кілька разів для тестування. Для складних скриптів буде доречним встановити значення overwriteOutput True, розмістивши цей вираз біля початку скрипта, після імпорту arcpy, але раніше за будь-які виклики інструментів:

```
>>> arcpy.env.overwriteOutput = True
```

Оскільки Python чутливий до регістру, обов'язково використовуйте верблюжий для overwriteOutput. Для параметрів середовища не буде помилкою, якщо застосувати правільний регістр; він просто не буде працювати, як і варто було сподіватися. У наступному прикладі ми використаємо неправильний регістр, і повідомлення про помилку не буде, але значення властивості overwriteOutput не змінюється на значення False:

```
>>> arcpy.env.overwriteoutput = False
>>> arcpy.env.overwriteOutput
True
```

Інші параметри оточення можуть бути корисні для конкретних проблем. Наприклад, створюючі набори растрових даних, можна

встановити параметр розмір файла, який також визначає висоту і ширину даних, що зберігаються в блоках. За замовчуванням розмір – 128 на 128 пікселів:

```
>>> arcpy.env.tileSize
u'128 128'
```

Введіть наступний рядок коду, щоби надрукувати повний список доступних параметрів середовища:

```
>>> arcpy.ListEnvironments()
```

Використовуйте ці кроки для створення і тестування «*aggregate.py*». Кроки проходять через створення моделі, її експорт, і запуск в ролі скрипта. Сценарій буде викликати інструмент Aggregate Polygons (Cartography), який групує близько розміщені об’єкти (рис. 8.14). Якщо вхідні об’єкти знаходяться в межах певної відстані агрегації, цей скрипт об'єднуватиме їх в єдине ціле.

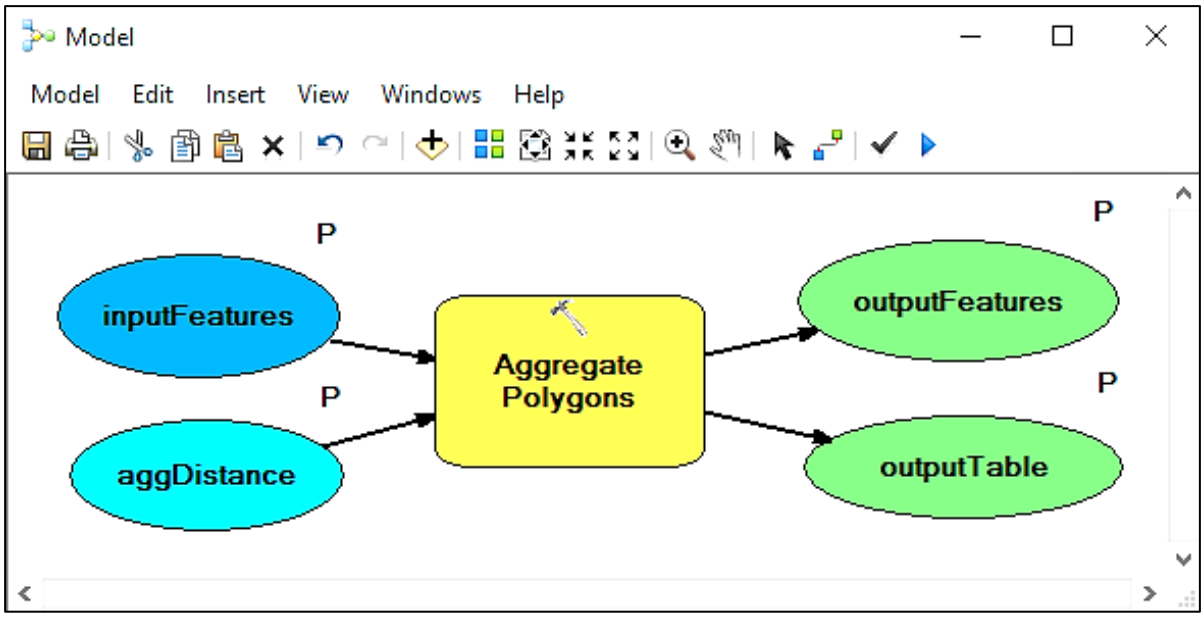

Рис. 8.14. Приклад сценарію у ModelBuilder

ЗАПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Наведіть загальні правила синтаксису Python.
2. З якої конструкції починається умова у синтаксисі програмування?
3. Які методи створення словників у Python вам відомі?
4. Наведіть популярні бібліотеки для читання і запису геопросторових даних та охарактеризуйте стисло їхні особливості
5. Що таке Shapely та для чого він призначений?
6. Назвіть методи запуску коду Python під час кожного запуску QGIS
7. Яким чином створюють просторові для векторного шару у QGIS?
8. Що таке рендер та для чого він призначений?
9. Які основні елементи макета карти можна додати за допомогою Python у QGIS?
10. З якою метою використовують ключове слово import у бібліотеці arcpy?
11. Що таке ModelBuilder?
12. Яким чином ModelBuilder пов’язаний з arcpy?
13. Назвіть основні компоненти функціональності arcpy.
14. Що називають параметрами середовища у ArcGIS?
15. Яким чином параметри середовища налаштовують за допомогою arcpy?

СПИСОК ЛІТЕРАТУРИ

1. Закон України Про національну інфраструктуру геопросторових даних : прийнятий 13 квіт. 2020 року № 554-IX// Відомості Верховної Ради України. – 2020. – № 37. – Ст. 277. URL: https://zakon.rada.gov.ua/laws/show/554-20#Text (дата звернення: 25.05.2023). – Назва з екрана.
2. Про затвердження Порядку функціонування національної інфраструктури геопросторових даних: Постанова Кабінету Міністрів України від 26 трав. 2021 р. № 532. – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/532-2021-п#Text (дата звернення: 25.05.2023). – Назва з екрана.
3. Про затвердження технічних вимог до геопросторових даних, метаданих і геоінформаційних сервісів національної інфраструктури геопросторових даних: Наказ Міністерства аграрної політики та продовольства України від 10 листопада 2021 р. N 347. – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0021-22#Text (дата звернення: 25.05.2023). – Назва з екрана.
4. Карпінський Ю.О. Геопросторовий аналіз: навчальний посібник / Ю.О. Карпінський, А.А. Лященко, Ю.В. Кравченко. – Київ : КНУБА, 2013. – 205 с.
5. Карпінський Ю.О. Стратегія формування національної інфраструктури геопросторових даних в Україні : манаграція / Ю.О. Карпінський, А.А. Лященко. – Київ : НДІГК, 2006. – 106 с.
6. Прикладне програмування в ГІС: методичні вказівки до виконання лабораторних робіт / уклад.: Н.Ю. Лазоренко-Гевель та ін. – Київ : КНУБА, 2021. – 88 с.
7. ArcGIS Pro [Електронний ресурс]. – Режим доступу: https://pro.arcgis.com/en/ (дата звернення: 25.05.2023). – Назва з екрана.
8. GDAL documentation. – [Електронний ресурс]. – Режим доступу: http://www.gdal.org (дата звернення: 25.05.2023). – Назва з екрана.
9. Mapnik [Електронний ресурс]. – Режим доступу: http://mapnik.org (дата звернення: 25.05.2023). – Назва з екрана.
10. Lawhead, J. (2015). QGIS Python programming cookbook. *Packt Publishing Ltd.* p. 340.

11. OGC SFA – Simple feature access – Part 1: Common architecture. 2010.

12. OGC SFA-S – Simple feature access – Part 2: SQL option, 2010.

13. Python. [Електронний ресурс]. – Режим доступу: https://www.python.org (дата звернення: 25.05.2023). – Назва з екрана.

14. PyQGIS Cookbook (QGIS 3.28) [Електронний ресурс]. – Режим доступу: https://docs.qgis.org/3.28/en/docs/pyqgis_developer_cookbook/composer.html (дата звернення: 25.05.2023). – Назва з екрана.

15. PyQGIS Cookbook (QGIS 3.16). – [Електронний ресурс]. – Режим доступу: https://docs.qgis.org/3.16/pdf/en/QGIS-3.16-PyQGISDeveloperCookbook-en.pdf (дата звернення: 25.05.2023). – Назва з екрана.

16. QGIS. – [Електронний ресурс]. – Режим доступу: https://www.qgis.org/uk/site/about/index.html (дата звернення: 25.05.2023). – Назва з екрана.

17. Tateosian L. Pyhon for ArcGIS. – *Springer International Publishing*, 2015 – 538 p.: ill.

18. Toms S. (2015). ArcPy and ArcGIS–Geospatial Analysis with Python. *Packt Publishing Ltd*., p. 343.

19. Westra E. Python Geospatial Development. – Birmingham.: *Packt Publishing Ltd*, 2012 – 508 p.: ill.

20. What is ArcPy?–ArcMap | Documentation. – [Електронний ресурс]. – Режим доступу: http://desktop.arcgis.com/ru/desktop/latest/analyze/arcpy/what-isarcpy-.htm (дата звернення: 25.05.2023). – Назва з екрана.

21. The Open Source Geospatial Foundation. – [Електронний ресурс]. – Режим доступу: http://www.osgeo.org (дата звернення: 25.05.2023). – Назва з екрана.

ГЛОСАРІЙ

Базові геопросторові дані – загальнодоступні геопросторові дані, що становлять уніфіковану цифрову координатно-просторову основу для виробництва, інтеграції та провадження іншої діяльності з різними геопросторовими даними.

Веб-сайт – сукупність програмних засобів, розміщених за унікальною адресою в обчислювальній мережі, зокрема в мережі Інтернет, разом з інформаційними ресурсами, що перебувають у розпорядженні певних суб'єктів і забезпечують доступ юридичних та фізичних осіб до цих інформаційних ресурсів та інших інформаційних послуг через обчислювальну мережу.

Геоінформаційний сервіс – спеціалізований веб-сервіс, що надається геопорталом через інтерфейс прикладного програмування за стандартом Відкритого геопросторового консорціуму OGC для перетворення, керування або відображення геопросторових даних або метаданих.

Геоінформаційні ресурси – результати інтелектуальної діяльності в усіх сферах життєдіяльності людини, суспільства і держави, що містять відомості про геопросторові об'єкти та зафіксовані на відповідних матеріальних носіях інформації як окремі набори геопросторових даних, бази та банки геопросторових даних, каталоги та бази метаданих і геоінформаційні сервіси.

Геопортал – комплекс програмно-технічних засобів, мережевих сервісів та сервісів геопросторових даних, що забезпечують відображення в мережі Інтернет геопросторових даних та метаданих, а також доступ користувачів до таких даних.

Геопросторовий об'єкт – об'єкт, що характеризується певним місцезнаходженням на Землі і визначеними у встановленій системі просторово-часовими координатами.

Геопросторові дані – сукупність даних про геопросторовий об'єкт.

Дані – інформація у формі, придатній для її автоматизованої обробки засобами обчислювальної техніки.

Інформаційна (автоматизована) система – організаційно-технічна система, в якій реалізується технологія обробки інформації з використанням технічних і програмних засобів.

Електронна карта – це зображення карти на екрані дисплея або автоматизована роздруківка електронного зображення карти на принтерах/плотерах або це файл зображення електронної карти у форматах, що слугують отриманню її копії на екрані або принтері без застосування ГІС.

Модуль – це файл, що містить код, який можна використовувати в інших програмах.

Національна інфраструктура геопросторових даних – взаємопов'язана сукупність організаційної структури, технічних і програмних засобів, базових та тематичних наборів геопросторових даних, метаданих, сервісів, технічних регламентів, стандартів, технічних специфікацій, потрібних для виробництва, оновлення, оброблення, зберігання, оприлюднення, використання геопросторових даних та метаданих, іншої діяльності з такими даними.

Національний геопортал – офіційний геопортал національної інфраструктури геопросторових даних, що слугує оприлюдненню та доступу до геопросторових даних та метаданих.

Рядки — це послідовності символів з довільним доступом, що використовуються для зберігання і подіння текстової інформації, тому за допомогою рядків можна працювати з усім, що може бути подане в текстовій формі.

Сервіс – програмно-технічний засіб, за допомогою якого надається можливість виконувати пошук, перегляд, доступ, завантаження, перетворення геопросторових даних та метаданих та іншу діяльність з такими даними.

ArcPy – це бібліотека Python, що забезпечує доступ з Python до всіх інструментів геообробки, зокрема до додаткових модулів, а також пропонує велику кількість корисних функцій і класів для роботи з даними ГІС.

ModelBuilder – це вбудований додаток у ArcGIS, що дає користувачам змогу створювати візуалізацію робочого потоку даних.

Lambda-функція – це функція, яка має довільну кількість аргументів (зокрема необов'язкові аргументи) і повертає значення одного виразу.

Навчальне видання

ЛАЗОРЕНКО Надія Юріївна,
ДЕНИСЮК Богдан Іванович,
КІНЬ Данило Олексійович

ПРИКЛАДНЕ ПРОГРАМУВАННЯ В ГІС

Навчальний посібник

Редагування та коректура *Г.В. Кобриної*
Комп'ютерне верстання *Т.І. Кукарєвої*

Підписано до друку 2023. формат $60*40_{1/16}$
Ум. друк. арк. . Обл.-вид. арк. .
Тираж 25 прим. Вид № 29/І-23. Зам. № /1-23

Видавець і виготовлювач
Київський національний університет будівництва і архітектури

Повітрофлотський проспект, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб'єктів
Видавничої справи ДК №808 від 13.02.2002 р.