



Робота з файлами



```
from pathlib import Path
```

```
p = Path('/home/user/file.txt') # Абсолютний шлях
```

```
p = Path('documents/report.pdf') # Відносний шлях
```

```
p = Path('/home/user') / 'documents' / 'report.pdf'
```

```
p = Path('/home/user/documents/report.pdf')
```

```
print(p.name) # report.pdf (назва файлу)
```

```
print(p.stem) # report (назва файлу без розширення)
```

```
print(p.suffix) # .pdf (розширення файлу)
```

```
print(p.parent) # /home/user/documents (батьківська директорія)
```

```
print(p.anchor) # / (корінь шляху)
```

```
print(p.exists()) # Перевірка, чи існує файл/директорія
```

```
print(p.is_file()) # Чи є це файлом?
```

```
print(p.is_dir()) # Чи є це директорією?
```



Характеристика	Абсолютний шлях	Відносний шлях
Початкова точка	Завжди починається з кореня (/, C:\)	Від поточної робочої директорії
Залежність від контексту	Незалежний	Залежний від поточної директорії
Довжина	Зазвичай довший	Зазвичай коротший
Використання	Для глобального доступу до файлів	Для локальних проектів

Створення директорій

```
p = Path('project/folder/subfolder')  
p.mkdir(parents=True, exist_ok=True)
```


parents (тип: bool, за замовчуванням False)

Вказує, чи потрібно створювати всі проміжні директорії, якщо їх не існує.

exist_ok (тип: bool, за замовчуванням False)

казує, чи потрібно ігнорувати помилку, якщо директорія вже існує.

Якщо exist_ok=True, метод не викличе помилку, якщо директорія вже існує.



```
p = Path('example_folder')  
p.mkdir() # Помилка, якщо директорія вже існує
```

```
p = Path('parent/child/grandchild')  
p.mkdir(parents=True) # Створює всі проміжні директорії
```

```
p = Path('existing_folder')  
p.mkdir(exist_ok=True) # Не виникає помилка, якщо директорія вже є
```

```
p.mkdir(mode=0o700) # Тільки власник може читати, писати та виконувати (актуально для Unix/Linux)
```

Метод read_text

Path.read_text(encoding=None, errors=None)

encoding (тип: str, за замовчуванням None)

Визначає кодування файлу.

utf-8, latin1, ascii

errors (тип: str, за замовчуванням None)

Вказує, як обробляти помилки декодування.

'strict', 'ignore', 'replace':

Метод read_bytes()



Параметр	<code>read_text</code>	<code>read_bytes</code>
Тип поверненого значення	Рядок (str)	Байти (bytes)
Кодування	Використовує вказане кодування	Не використовує кодування
Обробка помилок	Підтримує параметр <code>errors</code> для декодування	Помилки не обробляються
Тип файлів	Використовується для текстових файлів	Використовується для бінарних файлів
Аргументи	<code>encoding, errors</code>	Немає аргументів



Path.write_text(data, encoding=None, errors=None, newline=None)

data (тип: str) Текстовий вміст, який потрібно записати у файл.

encoding (тип: str, за замовчуванням None)

Визначає кодування

errors (тип: str, за замовчуванням None)

Визначає, як обробляти помилки, що виникають під час кодування тексту.

newline (тип: str, за замовчуванням None)

None: Використовує стандартні символи нового рядка для поточної платформи.

'\n', '\r\n', або '\r': Використовує відповідний символ нового рядка незалежно від платформи.

Контекстный менеджер

```
with expression [as variable]:  
    block
```

```
file = open("example.txt", "r")  
try:  
    content = file.read()  
print(content)  
finally: file.close()
```

```
with open("example.txt", "r") as file:  
    content = file.read()  
    print(content)
```



Режим	Опис
'r'	Відкрити файл для читання (за замовчуванням). Файл має існувати.
'w'	Відкрити для запису. Якщо файл існує, він буде очищений; якщо ні, створюється новий.
'x'	Відкрити для ексклюзивного створення. Якщо файл існує, викликається помилка.
'a'	Відкрити для дозапису. Дані додаються в кінець файлу. Якщо файл не існує, він створюється.
'b'	Бінарний режим (використовується з іншими режимами, наприклад, 'rb' або 'wb').
't'	Текстовий режим (за замовчуванням, використовується з іншими режимами, наприклад, 'rt').
'+'	Відкрити для читання і запису (наприклад, 'r+' або 'w+').



Метод	Опис
<code>json.dump()</code>	Запис Python-об'єкта у файл у форматі JSON.
<code>json.dumps()</code>	Перетворення Python-об'єкта у JSON-рядок.
<code>json.load()</code>	Читання JSON з файлу і перетворення його у Python-об'єкт.
<code>json.loads()</code>	Перетворення JSON-рядка у Python-об'єкт.



JSON	Python
object	dict (словник)
array	list (список)
string	str (рядок)
number	int або float
true	True
false	False
null	None



```
import csv
```

```
with open("data.csv", "r", encoding="utf-8") as file:
```

```
    reader = csv.reader(file)
```

```
    for row in reader:
```

```
        print(row)
```

```
with open("data.csv", "r", encoding="utf-8") as file:
```


```
    reader = csv.DictReader(file)
```

```
    for row in reader:
```

```
        print(row)
```

Альтернатива для роботи з даними


```
import pandas as pd  
df = pd.read_csv("data.csv")  
print(df)
```



```
import xml.etree.ElementTree as ET
# Парсинг XML-файлу
tree = ET.parse("data.xml")
root = tree.getroot()
```

```
# Виведення тегів і їх значень
```

```
for user in root.findall("user"):
    user_id = user.get("id") # Отримуємо атрибут
    name = user.find("name").text # Отримуємо текст тегу <name>
    age = user.find("age").text # Отримуємо текст тегу <age>
    print(f"ID: {user_id}, Name: {name}, Age: {age}")
```



```
import xml.etree.ElementTree as ET
# Створюємо кореневий елемент
root = ET.Element("data") # Додаємо користувачів

user1 = ET.SubElement(root, "user", id="1")
ET.SubElement(user1, "name").text = "Іван"
ET.SubElement(user1, "age").text = "30"

user2 = ET.SubElement(root, "user", id="2")
ET.SubElement(user2, "name").text = "Ольга"
ET.SubElement(user2, "age").text = "25"
# Створюємо дерево і зберігаємо у файл
tree = ET.ElementTree(root)
tree.write("output.xml", encoding="utf-8", xml_declaration=True)
```




```
<data>
```

```
  <user id="1">
```

```
    <name>Иван</name>
```

```
    <age>30</age>
```

```
  </user>
```

```
  <user id="2">
```

```
    <name>Ольга</name>
```

```
    <age>25</age> </user>
```

```
</data>
```