Тема 8. Модуль OS та робота з файловою системою, Модулі shutil, pathlib, glob та os.path

I. Модуль OS та робота з файловою системою

Ряд можливостей роботи з каталогами і файлами надає вбудований модуль os. Він містить багато функцій, нижче розглянуті лише основні з них:

- mkdir(): створює нову папку
- rmdir(): видаляє папку
- rename(): перейменовує файл
- remove(): видаляє файл

Створення та видалення папки

Для створення папки застосовується функція mkdir(), в яку передається шлях до створюваної папки:

```
importos

шлях щодо поточного скрипту
os.mkdir("hello")
абсолютний шлях
os.mkdir("c://somedir")
os.mkdir("c://somedir/hello")
```

Для видалення папки використовується функція rmdir(), в яку передається шлях до папки, що видаляється:

```
importos

шлях щодо поточного скрипту
os.rmdir("hello")
абсолютний шлях
os.rmdir("c://somedir/hello")
```

Перейменування файлу

Для перейменування викликається функція rename(source, target), перший параметр якої шлях до вихідного файлу, а другий - нове ім'я файлу. Як шляхи можуть використовуватися як абсолютні, так і відносні.

Наприклад, нехай у папці C://SomeDir/ розміщується файл somefile.txt. Перейменуємо його на файл "hello.txt":

```
importos
```

```
os.rename("C://SomeDir/somefile.txt",
"C://SomeDir/hello.txt")
```

Видалення файлу

Для видалення викликається функція remove(), яку передається шлях до файлу:

```
importos

os.remove("C://SomeDir/hello.txt")
```

Існування файлу

Якщо спробувати відкрити файл, який немає, то Python «викине» виняток FileNotFoundError.

Для вилову виключення можна використовувати конструкцію try...except. Однак можна вже до відкриття файлу перевірити, чи існує він за допомогою методу os.path.exists(path).

У цей метод передається повне ім'я файлу (шлях та ім'я), існування якого необхідно перевірити:

```
filename = input("Введіть шлях до файлу: ")
ifos.path.exists(filename):
    print("Вказаний файл існує")
else:
    print("Файл не існує")
```

Робота з операційною системою

Модуль os також надає безліч функцій для роботи з операційною системою, причому їхня поведінка, як правило, не залежить від ОС, тому програми залишаються переносимими. Тут будуть наведені найчастіше використовувані з них (деякі функції цього модуля підтримуються не всіма ОС).

os.name- Ім'я операційної системи. Доступні варіанти: 'posix', 'nt', 'mac', 'os2', 'ce', 'java'.

os.environ- словник змінних оточення. Змінюваний (можна додавати та видаляти змінні оточення).

os.getlogin() - Ім'я користувача, що увійшов до терміналу (Unix).

os.getpid() – поточний id процесу.

os.uname() - інформація про ОС. Повертає об'єкт з атрибутами: sysname – ім'я операційної системи, nodename – ім'я машини в мережі (визначається реалізацією), release – реліз, version – версія, machine – ідентифікатор машини.

os.access(path, mode, *, dir_fd=None, effective_ids=False, follow_symlinks=True) – перевірка доступу до об'єкта у поточного

користувача. Прапори: os.F_OK – об'єкт існує, os.R_OK – доступний на читання, os.W_OK – доступний на запис, os.X_OK – доступний на виконання.

os.chdir(path) – зміна поточної директорії.

os.chmod(path, mode, *, dir_fd=None, follow_symlinks=True) – зміна прав доступу до об'єкта (mode – вісімкове число).

os.chown(path, uid, gid, *, dir_fd=None, follow_symlinks=True) - змінює id власника та групи (Unix).

os.getcwd() – поточна робоча директорія.

os.link(src, dst, *, src_dir_fd=None, dst_dir_fd=None, follow_symlinks=True) - створює жорстке посилання.

os.listdir(path=".") - список файлів та директорій у папці.

os.mkdir(Path, mode = 0o777, *, dir_fd = None) - створює директорію. OSError, якщо директорія існує.

os.makedirs(Path, mode = 0o777, exist_ok = False) - створює директорію, створюючи при цьому проміжні директорії.

os.remove(path, *, dir_fd=None) – видаляє шлях до файлу.

os.rename(src, dst, *, src_dir_fd=None, dst_dir_fd=None) - перейменовує файл або директорію з src на dst.

os.renames(old, new) - перейменовує old на new, створюючи проміжні директорії.

os.replace(src, dst, *, src_dir_fd=None, dst_dir_fd=None) - перейменовує з src на dst з примусовою заміною.

os.rmdir(Path, *, dir_fd = None) - видаляє порожню директорію.

os.removedirs(path)- Видаляє директорію, потім намагається видалити батьківські директорії, і видаляє їх рекурсивно, поки вони порожні.

os.symlink(source, link_name, target_is_directory = False, *, dir_fd = None) - Створює символічне посилання на об'єкт.

os.sync() – записує всі дані на диск (Unix).

os.truncate(path, length) – обрізає файл до довжини length.

os.utime(path, times=None, *, ns=None, dir_fd=None, follow_symlinks=True) - модифікація часу останнього доступу та зміни файлу. Або times - кортеж (час доступу на секундах, час зміни на секундах), чи ns - кортеж (час доступу на наносекундах, час зміни на наносекундах).

os.walk(top, topdown = True, on error = None, followlinks = False) - генерація імен файлів у дереві каталогів, зверху вниз (якщо topdown дорівнює True), або знизу вгору (якщо False). Для кожного каталогу функція walk повертає кортеж (шлях до каталогу, список каталогів, список файлів).

приклад– вивести на консоль список усіх *.txt файлів, розміщених у папці l:\dist та її підпапках:

```
importos
```

```
fromos.path import join
for(root, dirs, files) in os.walk('l:\\dist'):
      for filename in files:
            if filename.endswith('.txt'):
                  thefile = os. path.join(root, filename)
                  print( os.path.getsize(thefile), thefile)
```

os.system(command) - виконує системну команду, повертає код її завершення (у разі успіху 0).

os.urandom(n) – n випадкових байт. Можливе використання цієї функції у криптографічних цілях.

II. Приклад команд роботи з файлами та файловою системою

Розглянемо 8 вкрай важливих команд для роботи з файлами, папками і файловою системою в цілому.

Показати поточний каталог

Найпростіша і водночас одна з найважливіших команд для Python-розробника. Вона потрібна тому, що найчастіше розробники мають справу із відносними шляхами. Але в деяких випадках важливо знати, де ми знаходимося.

Відносний шлях хороший тим, що працює для всіх користувачів з будь-якими системами, кількістю дисків і так далі.

Так ось, щоб показати поточний каталог використовуємо вбудовану в Python OS-бібліотеку:

```
import os
os.getcwd()
```

Майте на увазі, що шлях, що повертається, є абсолютним.

Перевіряємо, чи існує файл або каталог

Перш ніж задіяти команду створення файлу або каталогу, варто переконатися, що аналогічних елементів немає. Це допоможе уникнути ряду помилок під час роботи програми, включаючи перезапис існуючих елементів з даними.

Функція os.path.exists () приймає аргумент рядкового типу, який може бути або ім'ям каталогу або файлом.

Перевіримо, чи існує каталог sample_data:

```
os.path.exists('sample_data')
```

```
[3] os.path.exists('sample_data')

    True
```

Ця ж команда підходить і для роботи з файлами:
`os.path.exists('sample_data/README.md')`

```
[4] os.path.exists('sample_data/README.md')

    True
```

Якщо папки чи файлу немає, команда повертає false.

```
[5] os.path.exists('foobar')

    False
```

Об'єднання компонентів шляху

У попередньому прикладі використовувався сліш "/" для роздільника компонентів колії. У принципі, це нормально, але не рекомендується. Якщо ви хочете, щоб ваша програма була крос платформною, такий варіант не підходить. Так, деякі старі версії ОС Windows розпізнають лише слєш "\" як роздільник.

Python вирішує цю проблему завдяки функції os.path.join().

Давайте перепишемо варіант із прикладу в попередньому пункті, використовуючи цю функцію:
`os.path.exists(os.path.join('sample_data', 'README.md'))`

```
[7] os.path.exists(os.path.join('sample_data', 'README.md'))

    True
```

Створення директорії

Створимо директорію з ім'ям test_dir усередині поточної директорії. Для цього можна використовувати функцію os.mkdir():

```
os.mkdir('test_dir')
```

Погляньмо, як це працює на практиці.

```
[9] print(f"test_dir existing: {os.path.exists('test_dir')}")
    os.mkdir('test_dir')
    print(f"test_dir existing: {os.path.exists('test_dir')}")

    test_dir existing: False
    test_dir existing: True
```

Якщо ми спробуємо створити каталог, який вже існує, то отримаємо виняток.

```
[11] os.mkdir('test_dir')

---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-11-48d7b58469aa> in <module>()
----> 1 os.mkdir('test_dir')

FileExistsError: [Errno 17] File exists: 'test_dir'
```

Саме тому рекомендується завжди перевіряти наявність каталогу з певною назвою перед створенням нового:

```
if not os.path.exists('test_dir'):
    os.mkdir('test_dir')
```

Ще одна порада щодо створення каталогів. Іноді нам потрібно створити підкаталоги з рівнем вкладеності 2 або більше. Якщо ми досі використовуємо os.mkdir(), нам потрібно буде зробити це кілька разів.

І тут ми можемо використовувати os.makedirs(). Ця функція створить усі проміжні каталоги:

```
os.makedirs(os.path.join('test_dir', 'level_1',
'level_2', 'level_3'))
```

Ось що виходить у результаті.

Показуємо вміст директорії

Ще одна корисна команда – os.listdir(). Вона показує весь вміст каталогу.

Команда відрізняється від os.walk(), де останній рекурсивно показує все, що знаходиться під каталогом. os.listdir () набагато простіше у використанні, тому що просто повертає список вмісту:

```
os.listdir('sample_data')
```

```
[13] os.listdir('sample_data')
```

```
['README.md',
 'anscombe.json',
 'california_housing_train.csv',
 'california_housing_test.csv',
 'mnist_train_small.csv',
 'mnist_test.csv']
```

У деяких випадках потрібно щось більш просунуте, наприклад, пошук усіх CSV-файлів у каталозі «sample_data». У цьому випадку найпростіший спосіб - використовувати вбудовану бібліотеку glob (див. нижче):

```
з glob import globlist(glob(os.path.join('sample_data',
'*.csv'))))
```

```
[14] from glob import glob
```

```
[15] list(glob(os.path.join('sample_data', '*.csv')))
```

```
['sample_data/california_housing_train.csv',
 'sample_data/california_housing_test.csv',
 'sample_data/mnist_train_small.csv',
 'sample_data/mnist_test.csv']
```

Переміщення файлів

Саме час спробувати перемістити файли з однієї папки до іншої. Рекомендований спосіб – ще одна вбудована бібліотека shutil (див. нижче).

Перемістимо всі CSV-файли з директорії sample_data до директорії test_dir:

```
import shutil for file in
list(glob(os.path.join('sample_data', '*.csv')))):

    shutil.move(file, 'test_dir')
```

До речі, є два способи виконати задумане. Наприклад, ми можемо використовувати бібліотеку OS, якщо не хочеться імпортувати додаткові бібліотеки. Як os.rename, і os.replace підходять вирішення завдання.

Але обидві вони недостатньо «розумні», щоб дозволити перемістити файли в каталог.

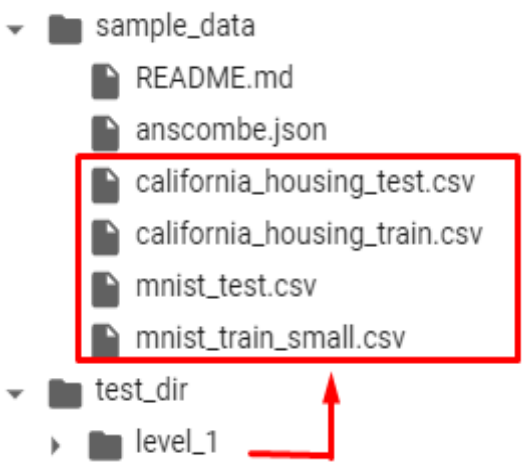
```
[21] for file in list(glob(os.path.join('test_dir', '*.csv'))):
         os.rename(file, 'sample_data')

     ---------------------------------------------------------------------------
     IsADirectoryError                         Traceback (most recent call last)
     <ipython-input-21-9675c94bbcdd> in <module>()
           1 for file in list(glob(os.path.join('test_dir', '*.csv'))):
     ----> 2     os.rename(file, 'sample_data')

     IsADirectoryError: [Errno 21] Is a directory: 'test_dir/california_housing_train.csv' -> 'sample_data'
```

Щоб усе це працювало, потрібно явно вказати ім'я файлу на місці призначення. Нижче код, який це дозволяє зробити:

```
for file in list(glob(os.path.join('test_dir',
'*.csv'))):
    os.rename(
        file,
        os.path.join(
            'sample_data',
            os.path.basename(file)
    ))
```



Тут функція os.path.basename() призначена для вилучення імені файлу зі шляху з будь-якою кількістю компонентів.

Інша функція, os.replace(), робить те саме. Але різниця в тому, що os.replace() не залежить від платформи, тоді як os.rename() працюватиме лише в системі Unix/Linux.

Ще один мінус у тому, що обидві функції не підтримують переміщення файлів з різних файлових систем, на відміну від shutil.

Тому рекомендується використовувати shutil.move() для переміщення файлів.

Копіювання файлів

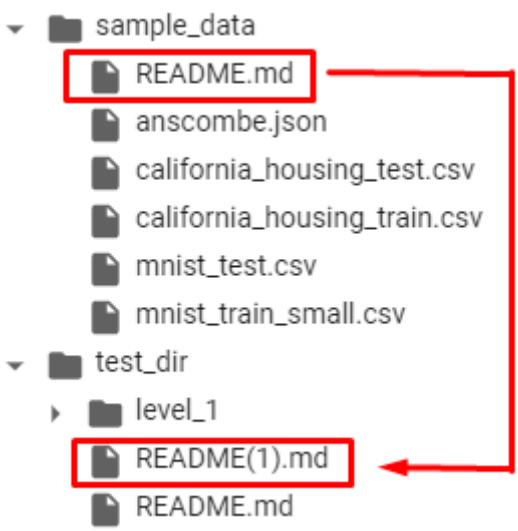
Аналогічно shutil підходить і для копіювання файлів з вже згаданих причин.

Якщо потрібно скопіювати файл README.md з папки sample_data в папку test_dir, допоможе функція shutil.copy():

```
shutil.copy(
    os.path.join('sample_data', 'README.md'),
    os.path.join('test_dir')
)
```

```
[26] shutil.copy(
         os.path.join('sample_data', 'README.md'),
         os.path.join('test_dir', 'README(1).md')
     )

     'test_dir/README(1).md'
```



Видалення файлів та папок

Коли потрібно видалити файл, потрібно скористатися командою os.remove():

```
os.remove(os.path.join('test_dir', 'README(1).md'))
```

Якщо потрібно видалити каталог, використовуємо os.rmdir():

```
os.rmdir(os.path.join('test_dir', 'level_1', 'level_2',
'level_3'))
```

Однак os.rmdir() може видалити лише порожній каталог.

На наведеному рис. бачимо, що можна видалити лише каталог level_3.

Що, якщо необхідно рекурсивно видалити каталог level_1? У цьому випадку використовуємо shutil.

```
[28] os.rmdir(os.path.join('test_dir', 'level_1'))

---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-28-352a126b9aab> in <module>()
----> 1 os.rmdir(os.path.join('test_dir', 'level_1'))

OSError: [Errno 39] Directory not empty: 'test_dir/level_1'
```

Функція shutil.rmtree() зробить все, що потрібно:

```
shutil.rmtree(os.path.join('test_dir', 'level_1'))
```

Користуватися нею потрібно обережно, оскільки вона безповоротно видаляє весь вміст каталогу.

III. Модуль shutil

Модуль shutil містить набір функцій високого рівня обробки файлів, груп файлів, і папок. Зокрема, доступні тут функції дозволяють копіювати, переміщати та видаляти файли та папки. Часто використовується разом із модулемos.

Операції над файлами та директоріями

shutil.copyfileobj(fsrc, fdst [, length])

- Скопіювати вміст одного файлового об'єкта (fsrc) в інший (fdst).

Необов'язковий параметр length - розмір буфера при копіюванні (щоб весь, можливо величезний, файл не читався повністю на згадку).

При цьому, якщо позиція покажчика fsrc не 0 (тобто до цього було зроблено щось на зразок fsrc.read(47)), то буде копіюватися вміст починаючи з поточної позиції, а не з початку файлу.

shutil.copyfile(src, dst, follow_symlinks = True)

- Копіює вміст (але не метадані) файлу src у файл dst.

Повертає dst (тобто куди файл було скопійовано). src та dst це рядки - шляхи до файлів. dst має бути повним ім'ям файлу.

Якщо src і dst є один і той же файл, виключення shutil.SameFileError.

Якщо dst існує, він буде перезаписаний.

Якщо follow_symlinks=False та src є посиланням на файл, то буде створено нове символічне посилання замість копіювання файлу, на який це символічне посилання вказує.

shutil.copymode(src, dst, follow_symlinks = True)

- Копіює права доступу з src в dst. Вміст файлу, власник і група не змінюються.

shutil.copystat(src, dst, follow_symlinks = True)

- копіює права доступу, час останнього доступу, останньої зміни та прапори src в dst. Вміст файлу, власник і група не змінюються.

shutil.copy(src, dst, follow_symlinks=True) - копіює вміст файлу src у файл чи папку dst. Якщо dst є директорією, файл буде скопійовано з тією самою назвою, що була у src. Функція повертає шлях до розташування нового скопійованого файлу.

Якщо follow_symlinks=False і src це посилання, dst буде посиланням.

Якщо follow_symlinks=True, і src це посилання, dst буде копією файлу, який посилається src

copy() копіює вміст файлу та права доступу.

shutil.copy2(src, dst, follow_symlinks = True)

- як copy(), але намагається копіювати всі метадані.

shutil.copytree(src, dst, symlinks=False, ignore=None, copy_function=copy2, ignore_dangling_symlinks=False)

- рекурсивно копіює все дерево директорій з коренем у src, повертає директорію призначення.

Директорія dst має існувати. Вона буде створена разом із пропущеними батьківськими директоріями.

Права та часи у директорій копіюються copystat(), файли копіюються за допомогою функції copy_function (за замовчуванням shutil.copy2()).

Якщо symlinks=True, посилання в дереві src будуть посиланнями в dst і метадані будуть скопійовані настільки, наскільки це можливо.

Якщо False (за промовчанням), буде скопійовано вміст і метадані файлів, на які вказували посилання.

Якщо symlinks=False, якщо файл, на який вказує посилання, немає, буде додано виняток до списку помилок, у виключенні shutil.Error в кінці копіювання.

Можна встановити прапорець ignore_dangling_symlinks=True, щоб приховати цю помилку.

Якщо ignore не None, то це має бути функція, яка приймає як аргументи ім'я директорії, в якій зараз copytree(), і список вмісту, що повертається os.listdir(). Т.к. copytree() викликається рекурсивно, ignore викликається 1 раз для кожної піддиректорії. Вона повинна повертати список об'єктів щодо поточного імені директорії (тобто підмножина елементів у другому аргументі). Ці об'єкти не будуть скопійовані.

shutil.ignore_patterns(*patterns)

- функція, яка створює функцію, яка може бути використана як ignore для copytree(), ігноруючи файли та директорії, які відповідаютьglob-стилі шаблонів.

Наприклад:

```
copytree(source, destination,
ignore=ignore_patterns('*.pyc', 'tmp*'))
# Копіює всі файли, крім тих, що закінчуються на .pyc
або починаються з tmp
```

shutil.rmtree(path, ignore_errors=False, onerror=None)

- видаляє поточну директорію та всі піддиректорії; path повинен вказувати на директорію, а не символічне посилання.

Якщо ignore_errors=True, то помилки, що виникають в результаті невдалого видалення, будуть проігноровані. Якщо False (за замовчуванням), ці помилки будуть передаватися обробнику onerror, або, якщо його немає, то виняток.

На ОС, які підтримують функції на основі файлових дескрипторів, за замовчуванням використовується версія rmtree(), не вразлива до атак символічних посилань.

На інших платформах це не так: за вибраного часу та обставин "хакер" може, маніпулюючи посиланнями, видалити файли, які недоступні йому в інших обставинах.

Щоб перевірити, чи вразлива система до подібних атак, можна використовувати атрибут rmtree.avoids_symlink_attacks.

Якщо заданий haerror, це має бути функція з 3 параметрами: function, path, excinfo.

Перший параметр, function, це функція, яка створила виняток; вона залежить від платформи та інтерпретатора. Другий параметр, path, це шлях, що передається функції. Третій параметр excinfo - це інформація про виключення, що повертається sys.exc_info(). Винятки, спричинені нанебезпеку, не обробляються.

shutil.move(src, dst, copy_function = copy2)

- рекурсивно переміщає файл або директорію (src) до іншого місця (dst), і повертає місце призначення.

Якщо dst – існуюча директорія, то src переміщається всередину директорії. Якщо dst існує, але з директорія, воно може бути перезаписано.

shutil.disk_usage(Path)

- повертає статистику використання дискового простору як намідтуплю з атрибутами total, used і free, в байтах.

shutil.chown(path, user=None, group=None)
- змінює власника та/або групу у файлу чи директорії.

shutil.which(cmd, mode=os.F_OK | os.X_OK, path=None)
- Повертає шлях до виконуваного файлу по заданій команді.
Якщо немає відповідності з жодним файлом, то None. mode це права доступу, потрібні від файлу, за замовчуванням шукає лише виконувані.

Архівація

Високорівневі функції для створення та читання архівованих та стислих файлів. Засновані на функціях із модулів zipfile та tarfile.
shutil.make_archive(base_name, format[, root_dir[, base_dir[, verbose[, dry_run[, owner[, group[, logger]]]]]]])
- Створює архів і повертає його ім'я.
`base_name`це ім'я файлу для створення, включаючи шлях, але не включаючи розширення (не потрібно писати ".zip" і т.д.).
`format`- Формат архіву.
`root_dir`- Директорія (щодо поточної), яку ми архівуємо.
`base_dir`- директорія, в яку архівуватиметься (тобто всі файли в архіві будуть у цій папці).
Якщо dry_run=True, архів не буде створений, але операції, які мали бути виконані, запишуться в logger.
`owner`та group використовуються при створенні tar-архіву.

shutil.get_archive_formats()
- Список доступних форматів для архівування.

```
>>>
>>> shutil.get_archive_formats()
[('bztar', "bzip2'ed tar-file"),
('gztar', "gzip'ed tar-file"),
('tar', 'uncompressed tar file'),
('xztar', "xz'ed tar-file"),
('zip', 'ZIP file')]
```

shutil.unpack_archive(filename[, extract_dir[, format]])
- Розпаковує архів. filename - повний шлях до архіву.
`extract_dir`- те, куди витягуватиметься вміст (за замовчуванням у поточну).
`format`- формат архіву (за замовчуванням намагається вгадати розширення файла).
shutil.get_unpack_formats() – список доступних форматів для розархівування.

Запит розміру терміналу виведення

shutil.get_terminal_size(fallback = (columns, lines))

- Повертає розмір вікна терміналу.

fallbackповернеться, якщо не вдалося дізнатися про розмір терміналу (термінал не підтримує такі запити, або програма працює без терміналу). За замовчуванням (80, 24).

```
>>>
>>> shutil.get_terminal_size()
os.terminal_size(columns=102, lines=29)
>>> shutil.get_terminal_size() # Зменшили вікно
os.terminal_size(columns=67, lines=17)
```

IV. Приклади використання модуля shutil.

Копіювання файлу

Функція shutil.copyfile() копіює вміст джерела в місце призначення та викликає виключення IOError, якщо у нього немає дозволу на запис до файлу призначення.

```
>>> import shutil, os
>>> from glob import glob
# створимо тимчасову директорію
>>> os.mkdir('example')
# створимо тестовий файл
>>> open('example/test_file.txt', 'w').close()
# Копіювання
>>> shutil.copyfile('example/test_file.txt',
'example/test_file.txt.copy')
# 'example/test-file.txt.copy'

# дивимося результат
>>> pprint.pprint(glob('example/*'))
# ['example/test_file.txt.copy',
'example/test_file.txt']

# видаляємо
>>> shutil.rmtree('example')
```

Функція`shutil.copy()`інтерпретує ім'я вихідного файлу як інструмент командного рядка з Unix cp. Якщо шлях призначення вказаний як каталог, а не файл, то в каталозі створюється новий файл із використанням його базового імені.

```
>>> import shutil, os
>>> from glob import glob
```

```
# створимо тестовий файл
>>> open('shutil_copy.txt', 'w').close()
# створимо тимчасову директорію
>>> os.mkdir('example')
>>> glob('example/*')
# []

# Копіюємо
>>> shutil.copy('shutil_copy.txt', 'example')
# 'example/shutil_copy.txt'

# дивимося результат
>>> glob('example/*')
# ['example/shutil_copy.txt']

# видаляємо
>>> shutil.rmtree('example')
```

Рекурсивне копіювання каталогу

Функція`shutil.copytree()`рекурсивно копіює весь каталог.

```
# для того, щоб протестувати функцію copytree()
# створимо ієрархію каталогів
import pathlib, itertools, glob, shutil
path = 'test/'
tmp = {'script':'.py', 'text':'.txt'}
for d, ext in tmp.items():
    comb = itertools.combinations([d, 1, 0], r=2)
    for a, b in comb:
        name = f'{a}{b}{ext}'
        pathlib.Path(path, d).mkdir(parents=True,
exist_ok=True)
        pathlib.Path(path, d,
name).touch(exist_ok=True)

f = glob.glob('test/**/*', recursive=True)
print(f)
# ['test/text', 'test/script', 'test/text/10.txt',
# 'test/text/text1.txt', 'test/text/text0.txt',
# 'test/script/10.py', 'test/script/script0.py',
# 'test/script/script1.py']

# Копіюємо
shutil.copytree('test', 'test_cp')
# дивимося результат
f_cp = glob.glob('test_cp/**/*', recursive=True)
```

```
print(f_cp)

# ['test/text', 'test/script', 'test/text/10.txt',
# 'test/text/text1.txt', 'test/text/text0.txt',
# 'test/script/10.py', 'test/script/script0.py',
# 'test/script/script1.py']

# видаляємо
shutil.rmtree('test_cp')
shutil.rmtree('test')
```

Вибіркове рекурсивне копіювання файлів каталогу

Приклад, який використовує помічник`shutil.ignore_patterns()`.

Тут копіюється все, крім файлів .pyc і файлів або каталогів, ім'я яких починається з tmp.

```
від shutil import copytree, ignore_patterns
copytree(source, destination,
ignore=ignore_patterns('*.pyc', 'tmp*'))
```

Інший приклад, який використовує аргумент ignore для додавання виклику логування копіювання файлів:

```
від shutil import copytree
import logging

def _logpath(path, names):
    logging.info('Working in %s', path)
    return [] # nothing will be ignored

copytree(source, destination, ignore=_logpath)
```

Рекурсивне видалення каталогу

Робота функції`shutil.rmtree()`,яка виконує рекурсивне видалення каталогу демонструвалася вище. Розберемо ситуації складніше.

У цьому прикладі показано, як видалити дерево каталогів у Windows, де для деяких файлів встановлений біт тільки для читання.

Він використовує зворотний виклик haerror, щоб очистити біт readonly і повторити спробу видалення.

```
import os, stat
import shutil
```

```
def remove_readonly(func, path, _):
    "Clear the readonly bit and reattempt the removal"
    os.chmod(path, stat.S_IWRITE)
    func(path)

shutil.rmtree(directory, onerror=remove_readonly)
```

У функції shutil.rmtree(), так само можна використовувати помічник shutil.ignore_patterns() для вибіркового рекурсивного видалення файлів, як це робиться у вибірковому рекурсивному копіюванні файлів.

Приклад реалізації функції shutil.copytree()

Цей приклад є реалізацією функції shutil.copytree() з опущеним рядком документації. Він демонструє багато інших функцій, що надаються цим модулем.

```
def copytree (src, dst, symlinks = False):
    names = os.listdir(src)
    os.makedirs(dst)
    errors = []
    for name in names:
        srcname = os.path.join(src, name)
        dstname = os.path.join(dst, name)
        try:
            if symlinks and os.path.islink(srcname):
                linkto = os.readlink(srcname)
                os.symlink(linkto, dstname)
            elif os.path.isdir(srcname):
                copytree(srcname, dstname, symlinks)
            else:
                copy2(srcname, dstname)
            # XXX What about devices, sockets etc.?
        except OSError as why:
            errors.append((srcname, dstname, str(why))))
        # catch the Error from the recursive copytree so that
we can
        # continue with other files
        except Error as err:
            errors.extend(err.args[0])
    try:
        copystat(src, dst)
    except OSError as why:
```

```
            # can't copy file access times on Windows
            if why.winerror is None:
                errors.extend((src, dst, str(why))))
        if errors:
            raise Error(errors)
```

Архівування каталогів

У цьому прикладі створимо архів tar-файлів з використанням gzip, що містить усі файли, знайдені в каталозі .ssh користувача:

```
від shutil import make_archive
import os
archive_name = os.path.expanduser(os.path.join('~',
'myarchive'))

root_dir = os.path.expanduser(os.path.join('~',
                                            '.ssh'))
make_archive(archive_name, 'gztar', root_dir)

'/Users/docs-python/myarchive.tar.gz'
```

Отриманий архів містить:

```
~$ tar -tzvf /home/docs-python/myarchive.tar.gz

drwx------ docs-python/docs-python 0 2019-12-17 16:57 ./
-rw------- docs-python/docs-python 1554 2019-12-17 16:53 ./known_hosts
-rw------- docs-python/docs-python 1554 2019-12-17 14:06 ./known_hosts.old
-rw------- docs-python/docs-python 1679 2019-09-16 12:02 ./id_rsa
-rw-r--r-- docs-python/docs-python 399 2019-09-16 12:02 ./id_rsa.pub
```

V. Модуль pathlib

Python 3 включає модуль pathlib для маніпуляції шляхами файлових систем незалежно від операційної системи.

pathlibсхожий на модуль os.path, але pathlib пропонує більш розвинений та зручний інтерфейс порівняно з os.path.

Зазвичай файли на комп'ютері ідентифікують за допомогою ієрархічних шляхів.

Наприклад, можна ідентифікувати файл wave.txt на комп'ютері за допомогою цього шляху: /Users/sammy/ocean/wave.txt.

Операційні системи представляють шляхи дещо по-різному. Windows може представляти шлях до файлу wave.txt як C:\Users\sammy\ocean\wave.txt.

Модуль pathlib може бути корисним, якщо у програмі Python ви створюєте або переміщаєте файли у файловій системі, вказуючи всі файли у файловій системі, що збігаються з даним розширенням або шаблоном, або

створюєте шляхи файлу, що відповідають файловій системі на основі наборів неформатованих рядків.

Хоча ви можете використовувати інші інструменти, наприклад модуль os.path, для виконання більшої частини цих завдань, але модуль pathlib дозволяє виконувати ці операції з більшим ступенем читання та мінімальною кількістю кодів.

Розглянемо деякі способи використання модуля pathlib для представлення та маніпуляції шляхами файлових систем.

Модуль pathlib надає кілька класів, але одним із найважливіших є клас Path.

Примірники класу Path представляють шлях до файлу або каталогу файлової системи вашого комп'ютера.

Наприклад, наступний код отримує екземпляр Path, який представляє частину шляху до файлу wave.txt:

```
від pathlib import Path

wave = Path("ocean", "wave.txt")
print(wave)
```

Якщо запустити цей код, результат буде виглядати так:

```
Output
ocean/wave.txt
```

`від pathlib import Path`робить клас Path доступним для нашої програми.

Потім Path("ocean", "wave.txt") отримує новий екземпляр Path.

З висновку результату видно, що Python «додав» відповідний роздільник оперативної системи між двома заданими нами компонентами шляху "ocean" і "wave.txt".

Примітка.Залежно від операційної системи, висновок може трохи відрізнятися від прикладів, наведених у цьому посібнику. У Windows, наприклад, висновок цього прикладу може виглядати як ocean\wave.txt.

У прикладі об'єкт Path, наданий змінною wave, містить відносний шлях.

Можна використовувати Path.home() для отримання абсолютного шляху до домашнього каталогу поточного користувача:

```
home = Path.home()
wave_absolute = Path(home, "ocean", "wave.txt")
print(home)
print(wave_absolute)
```

Якщо запустити цей код, результат буде виглядати приблизно так:

```
Output
```

```
/Users/sammy
/Users/sammy/ocean/wave.txt
```

Примітка. Як згадувалося раніше, висновок залежатиме від операційної системи

`Path.home()` повертає екземпляр Path з абсолютним шляхом до домашнього каталогу поточного користувача.

Потім ми передамо цей екземпляр Path і рядки "ocean" і "wave.txt" до іншого конструктора Path, щоб створити абсолютний шлях до файлу wave.txt.

Висновок показує, що перший рядок – це домашній каталог, а другий рядок – домашній каталог плюс ocean/wave.txt.

Цей приклад також ілюструє важливу функцію класу Path: конструктор Path приймає обидві рядки і об'єкти Path, що раніше існували.

Давайте детальніше розглянемо підтримку рядків та об'єктів Path у конструкторі Path:

```
shark = Path(Path.home(), "ocean", "animals",
Path("fish", "shark.txt"))

print(shark)
```

Якщо запустити цей код Python, результат буде виглядати так:

```
Output
/Users/sammy/ocean/animals/fish/shark.txt
```

`shark`— це Path до файлу, який ми створили за допомогою об'єктів Path (Path.home() та Path("fish", "shark.txt") та рядків "ocean" та "animals").

Конструктор Path інтелектуально обробляє обидва типи об'єктів та акуратно з'єднує їх за допомогою відповідного роздільника операційної системи, в даному випадку /.

Доступ до атрибутів файлу

Розглянемо, як можна використовувати екземпляри Path для доступу до інформації про файл.

Ми можемо використовувати атрибути name і suffix для доступу до імен та розширень файлів:

```
wave = Path("ocean", "wave.txt")
print(wave)
print(wave.name)
print(wave.suffix)
```

Запустивши цей код і отримаємо висновок, аналогічний до наступного:

```
Output
/Users/sammy/ocean/wave.txt
```

```
wave.txt
.txt
```

Цей висновок показує, що ім'я файлу в кінці нашого шляху wave.txt, а розширення файлу .txt.

Примірники Path також пропонують функцію with_name, що дозволяє без перешкод створювати новий об'єкт Path з іншим ім'ям:

```
wave = Path("ocean", "wave.txt")
tides = wave.with_name("tides.txt")
print(wave)
print(tides)
```

Якщо запустити його, результат буде виглядати так:

```
ocean/wave.txt
ocean/tides.txt
```

Код спершу створює екземпляр Path, який вказує на файл з ім'ям wave.txt.

Потім викликається метод with_name у wave, щоб повернути другий екземпляр Path, що вказує на новий файл з ім'ям tides.txt.

Частина каталогу ocean/ залишається незміненою та залишає фінальний шлях у вигляді ocean/tides.txt

Доступ до попередніх об'єктів

Іноді корисно отримати доступ до каталогів, які містять певний шлях. Давайте розглянемо приклад:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
print(shark)
print(shark.parent)
```

Якщо запустити цей код, результат буде виглядати так:

```
Output
ocean/animals/fish/shark.txt
ocean/animals/fish
```

Атрибут parent в екземплярі Path повертає найближчого попередника шляху файлу. У цьому випадку він повертає каталог із файлом shark.txt: ocean/animals/fish.

Можемо отримувати доступ до атрибуту parent кілька разів у команді, щоб пройти вгору кореневим деревом даного файлу:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
```

```
print(shark)
print(shark.parent.parent)
```

Якщо виконати цей код, побачимо наступні:

```
Output
ocean/animals/fish/shark.txt
ocean/animals
```

Висновок буде схожий на попередній висновок, але тепер ми перейшли на рівень вище, отримавши доступ до .parent вдруге. Два каталоги від shark.txt – це каталог ocean/animals.

Використання шаблону пошуку для списку файлів

Також можна використовувати клас Path для списку файлів за допомогою glob.

Припустимо, у нас є структура каталогу, яка виглядає так:

```
└── ocean
    ├── animals
    │   └── fish
    │   └── shark.txt
    ├── tides.txt
    └── wave.txt
```

Каталог ocean містить файли tides.txt та wave.txt. У нас є файл з ім'ям shark.txt, вкладений у каталог ocean, каталог animals та каталог fish: ocean/animals/fish.

Щоб перерахувати всі файли .txt у каталозі ocean, можна використовувати:

```
for txt_path in Path("ocean").glob("*.txt"):
    print(txt_path)
```

Цей код зробить такий висновок:

```
Output
ocean/wave.txt
ocean/tides.txt
```

Шаблон пошуку "*.txt" знаходить усі файли, що закінчуються на .txt. Оскільки приклад коду виконує цей пошук у каталозі ocean, він повертає два файли .txt у каталозі ocean: wave.txt та tides.txt.

Також можна використовувати метод glob рекурсивно.

Щоб перерахувати всі файли .txt у каталозі ocean і всі його підкаталоги, запишемо:

```
for txt_path in Path("ocean").glob("**/*.txt"):
    print(txt_path)
```

Якщо запустити цей код, результат буде виглядати так:

```
Output
ocean/wave.txt
ocean/tides.txt
ocean/animals/fish/shark.txt
```

Частина ** шаблону пошуку буде відповідати цьому каталогу та всім каталогам під ним рекурсивно.

Тому у висновку у нас будуть не тільки файли wave.txt та tides.txt, але також ми отримаємо файл shark.txt, вкладений у ocean/animals/fish.

Обчислення відносних шляхів

Можна використовувати метод Path.relative_to для обчислення шляхів, які стосуються один одного. Метод relative_to корисний, якщо, наприклад, ви хочете отримати частину довгого шляху файлу.

Розгляньте наступний код:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
below_ocean = shark.relative_to(Path("ocean"))
below_animals = shark.relative_to(Path("ocean",
"animals"))

print(shark)
print(below_ocean)
print(below_animals)
```

Якщо запустити його, результат буде виглядати так:

```
Output
ocean/animals/fish/shark.txt
animals/fish/shark.txt
fish/shark.txt
```

Метод relative_to повертає новий об'єкт Path, що належить до цього аргументу. У нашому прикладі ми обчислимо Path до shark.txt, що відноситься до каталогу ocean, а потім стосується обох каталогів ocean і animals.

Якщо relative_to не зможе вирахувати відповідь, оскільки ми даємо йому не пов'язаний шлях, він видасть ValueError:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
shark.relative_to(Path("unrelated", "path"))
```

Ми отримаємо виняток ValueError, що виник з цього коду, який виглядатиме так:

```
Output
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/Python3.8/pathlib.py", line 899, in
relative_to
    raise ValueError("{!r} does not start with {!r}"
ValueError: 'ocean/animals/fish/shark.txt' does not start with
'unrelated/path'
```

`unrelated/path`не є частиною ocean/animals/fish/shark.txt, тому Python не зможе обчислити відносний шлях.

Модуль pathlib представляє додаткові класи та утиліти, Можна скористатися посиланням надокументацію модуля pathlibдля отримання додаткової інформації.

VI. Модуль glob

Модуль glob знаходить всі шляхи, що збігаються із заданим шаблоном відповідно до правил, що використовуються оболонкою Unix.

Обробляються символи "*" (довільна кількість символів), "?" (один символ) та діапазони символів за допомогою [].

Для використання тильди "~" та змінних оточення необхідно використовуватиos.path.expanduser() та os.path.expandvars().

Для пошуку спецсимволів укладайте їх у квадратні дужки. Наприклад, [?] відповідає символу "?".

glob.glob(pathname)

- Повернення списку (можливо, порожній) шляхів, що відповідають шаблону pathname. Шлях може бути абсолютним (наприклад, /usr/src/Python-1.5/Makefile) або відносний (як ../../Tools/*/*.gif).

glob.iglob(pathname)

- Повертає ітератор, що дає ті ж значення, що й glob.glob.

glob.escape(pathname)

- екранує всі спеціальні символи для glob ("?", "*" та "["). Спеціальні символи в імені диска не екрануються (оскільки вони там не враховуються), тобто у Windows escape("//?/c:/Quo vadis?.txt") повертає "//?/c:/Quo vadis[?].txt".

Розглянемо, наприклад, каталог, що містить лише такі файли: 1.gif, 2.txt та card.gif.

glob.glob() поверне наступні результати. (Зверніть увагу, що будь-які провідні компоненти шляху зберігаються):

```
>>>
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```

Якщо каталог містить файли, що починаються з ".", вони не включатимуться за замовчуванням. Розглянемо, наприклад, каталог, що містить card.gif та .card.gif:

```
>>>
>>> import glob
>>> glob.glob('*.gif')
['card.gif']
>>> glob.glob('.c*')
['.card.gif']
```

VII. Модуль os.path

os.pathє вкладеним модулем модуль os, і реалізує деякі корисні функції для роботи з шляхами.

os.path.abspath(path) – повертає нормалізований абсолютний шлях.

os.path.basename(path) – базове ім'я шляху (еквівалентно os.path.split(path) [1]).

os.path.commonprefix(list) - повертає найдовший префікс усіх шляхів у списку.

os.path.dirname(path) – повертає ім'я директорії шляху path.

os.path.exists(path) – повертає True, якщо path вказує на існуючий шлях або дескриптор відкритого файлу.

os.path.expanduser(path) - замінює ~ або ~user на домашню директорію користувача.

os.path.expandvars(path) - повертає аргумент із підставленими змінними оточення ($name або ${name} замінюються змінною оточення name). Неіснуючі імена не замінює. Windows також замінює %name%.

os.path.getatime(Path) - час останнього доступу до файлу, в секундах.

os.path.getmtime(Path) - час останньої зміни файлу, в секундах.

os.path.getctime(path) – час створення файлу (Windows), час останньої зміни файлу (Unix).

os.path.getsize(Path) - розмір файлу в байтах.

os.path.isabs(path) – чи є шлях абсолютним.

os.path.isfile(path) – чи є шлях файлом.

os.path.isdir(path) – чи є шлях директорією.

os.path.islink(path) – чи є шлях символічним посиланням.

os.path.ismount(path) – чи є шлях точкою монтування.

os.path.join(path1[, path2[, ...]]) - з'єднує шляхи з урахуванням особливостей операційної системи.

os.path.normcase(path) - нормалізує регістр шляху (на файлових системах, які не враховують регістр, наводить шлях до нижнього регістру).

os.path.normpath(path) – нормалізує шлях, прибираючи надлишкові роздільники та посилання на попередні директорії. На Windows перетворює прямі сліші на зворотні.

os.path.realpath(path) - повертає канонічний шлях, забираючи всі символічні посилання (якщо вони підтримуються).

os.path.relpath(path, start=None) - обчислює шлях щодо директорії start (за умовчанням - щодо поточної директорії).

os.path.samefile(path1, path2) - чи вказують path1 і path2 на той самий файл або директорію.

os.path.sameopenfile(fp1, fp2) - чи вказують дескриптори fp1 і fp2 на той самий відкритий файл.

os.path.split(path) – розбиває шлях на кортеж (голова, хвіст), де хвіст – останній компонент шляху, а голова – все інше. Хвіст ніколи не починається зі сліша (якщо шлях закінчується слішем, то хвіст порожній). Якщо слішів у дорозі немає, то порожньою буде голова.

os.path.splitdrive(path) – розбиває шлях на пару (привід, хвіст).

os.path.splitext(Path) - розбиває шлях на пару (root, ext), де ext починається з точки і містить не більше однієї точки.

os.path.supports_unicode_filenames- Чи підтримує файлова система Unicode.