Python

Работа с модулем pathlib в Python 3

Порой разработчику приходится отойти от мира программной абстракции и обратиться к практическим вещам. Например, к файловой системе. Манипуляции путями файловых систем в Python 3 реализуется несколькими библиотеками. Одна из них — pathlib. И в этой статье мы хотели бы вам рассказать о том, как использовать модуль pathlib и решать практические задачи с помощью него.

Файловая система

Прежде, чем перейти непосредственно к практической части применения библиотеки pathlib, необходимо повторить матчасть и обсудить основную концепцию файловой системы и её терминологию.

Итак, у нас есть файл example.exe расположенный в C:\Program Files (файловая система Windows). У этого файла есть 4 основные характеристики с точки зрения файловой системы:

- Путь — это идентификатор файла, определяющий его местонахождение в файловой системе, исходя из последовательности названий каталогов. Их разделяют на два типа:
    - Абсолютный путь — полный путь файла, начинающийся с корневого каталога. Например, C:\Program Files\example.exe

    - Относительный путь — путь к файлу относительно другого каталога. Например, Program Files\example.exe

- Каталог (C:\Program Files) — это объект в файловой системе, необходимой для иерархической организации файловой системы. Иногда его называют директория или папка. Название «папка» отлично описывает практическое название этого объекта — удобное хранение и быстрое нахождение при необходимости.
- Расширение (.exe) — это формат файла. С помощью расширения пользователь или программы определяют тип данных, хранящихся в файле (видео, текст, музыка и т.д.).

Экземпляры Path

Импортируем модуль pathlib для манипуляции путями файловых систем:

```
from pathlib import *
```

Классы этого модуля можно разбить на два типа: Pure (или «чистые») и Concrete (или «конкретные»). Первые нужны для абстрактной вычислительной работы с путями, при которой не осуществляется непосредственное взаимодействие с файловой системой ОС. Вторые же необходимы для взаимодействия с ней (создания и удаление директорий, чтения файлов и т.д.).

Если вы не уверены, какой класс подходит для выполнения вашей задачи, то, скорее всего, класс Path будет в самый раз. Он позволяет взаимодействовать с файловой системой ОС, под которую сам подстраивается (UNIX или Windows), но и при этом ваша манипуляция путями не страдает от ограничений. В рамках этой статьи мы рассмотрим все классы и их применение, но сосредоточимся конкретно на Path.

Итак, начнем с малого. Создадим переменную с типом данных Path:

```
>>> PathExample = Path('Timeweb', 'Cloud', 'Pathlib')
>>> PathExample
WindowsPath('Timeweb/Cloud/Pathlib')
```

Как видим, модуль сам подстроился под операционную систему — в нашем случае Windows 10. Конструктор Path(args) создает новый экземпляр класса Path и на вход принимает его компоненты (директории, файлы, другие пути).

Путь PathExample — относительный. Чтобы он стал абсолютным необходимо добавить корневой каталог.

С помощью методов Path.home() и Path.cwd() мы можем получить каталог текущего пользователя и текущую рабочую папку:

```
>>> Path.home()
WindowsPath('C:/Users/Blog)
>>> Path.cwd()
WindowsPath('C:/Users/Blog/AppData/Local/Programs/Python/Python310')
```

Сделаем PathExample абсолютным и рассмотрим другие аспекты работы с pathlib:

```
>>> PathExample = Path.home() / PathExample
>>> PathExample
WindowsPath('C:/Users/Blog/Timeweb/Cloud/Pathlib')
```

С помощью оператора «/» мы можем создавать новые пути.

Атрибуты Path

Pathlib предоставляет различные методы и свойства для получения разнообразной информации о путях. Для наглядности добавим введем новую переменную AttributeExample и добавим в неё файл:

```
>>>AttributeExample  = PathExample / 'file.txt'
>>> AttributeExample
WindowsPath('C:/Users/Blog/Timeweb/Cloud/Pathlib/file.txt')
```

Диск

Чтобы узнать букву или имя диска, необходимо использовать свойство.drive:

```
>>> AttributeExample.drive      'C:'
```

Родительские каталоги

Мы можем получить родительские каталоги с помощью двух свойств: parent и parents[n].

Parent возвращает родительский каталог:

```
>>> AttributeExample.parent
WindowsPath('C:/Users/Blog/Timeweb/Cloud/Pathlib')
```

Чтобы получить более «высокие» родительские каталоги, можно использовать parent несколько раз:

```
>>> AttributeExample.parent.parent
WindowsPath('C:/Users/Blog/Timeweb/Cloud')
```

Или воспользоваться свойством parents[n], который возвращает n-го предка:

```
>>> AttributeExample.parents[3]
WindowsPath('C:/Users/Blog)
```

Имя

Для получения имени файла нужно использовать свойство .name:

```
>>> AttributeExample.name
'file.txt'
```

Расширение

Для того, чтобы получить расширение файла, необходимо использовать свойство .suffix или .suffixes (при наличии двойного расширения, например .tar.gz):

```
>>> AttributeExample.suffix
'.txt'
```

```
>>> Path('file.tar.gz').suffixes
['.tar', '.gz']
```

Абсолютный или относительный путь

Мы можем определить, является ли путь абсолютным, с помощью метода .is_absolute():

```
>>> AttributeExample.is_absolute()
True
```

Составные части

Мы можем разложить путь на компоненты с помощью свойства .parts:

```
>>> AttributeExample.parts
('C:\\', 'Users', ‘Blog’, 'Timeweb', 'Cloud', 'Pathlib', 'file.txt')
```

Сравнение путей

Мы можем сравнивать пути как с помощью операторов сравнения, так и с помощью различных методов.

Операторы сравнения

Мы можем узнать, являются ли пути одинаковыми:

```
>>>Path('timeweb') == Path('TIMEWEB')      True
```

Стоит упомянуть, что такое же сравнение для UNIX-систем будет давать результат False:

```
>>> PurePosixPath('timeweb') == PurePosixPath('TIMEWEB')       False
```

Это связано с тем, что в Windows регистр не имеет значения в имени директории, в отличие от файловой системы UNIX-систем.

Методы сравнения

Мы можем узнать, является ли один путь частью другого, с помощью метода .is_relative_to():

```
>>> CompareExample = AttributeExample
>>> CompareExample.is_relative_to('C:')
True
```

```
>>> CompareExample.is_relative_to('D:/')   False
```

Также можно использовать шаблоны для проверки с помощью метода .match():

```
>>> CompareExample.match('*.txt')      True
```

Создание и удаление папок и файлов

Для создания папок с помощью модуля pathlib необходимо использовать метод .mkdir(parents = True/False, exist_ok =True/False). Метод получает на вход (помимо пути, по которому нужно создать папку) 2 логические переменные: parents и exist_ok.

Если значение parents = True, то метод создаст родительские каталоги (при их отсутствии). Если False, то вернет ошибку при их отсутствии.

Exist_ok отвечает за обработку ошибок, возникающих при существовании целевой папки.

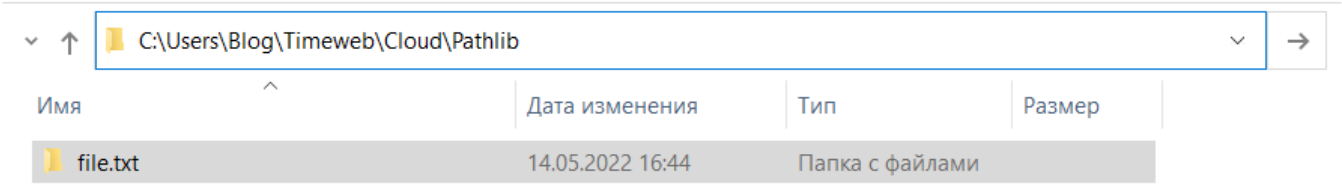
Создадим папку CreateExample, но сначала проверим, есть ли уже такая директория с помощью метода .is_dir():

```
>>> CreateExample = CompareExample
>>> CreateExample.is_dir()
False
```

Теперь попробуем его создать:

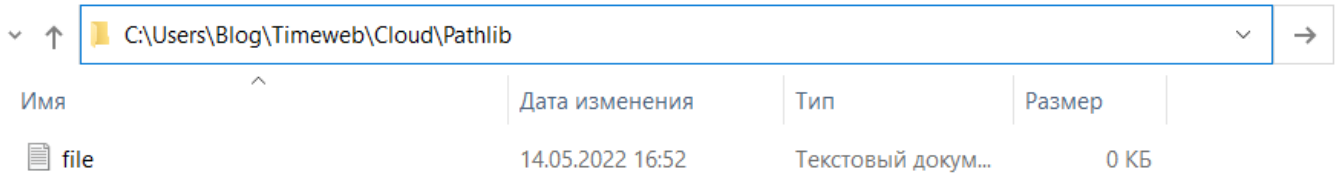
```
>>> CreateExample.mkdir(parents = True, exist_ok = True)          >>> CreateExa
```

Проверим папку в проводнике:



Как видим, появилась именно папка, а не файл. Чтобы создать пустой файл, необходимо использовать метод .touch(). Но сначала удалим папку file.txt с помощью метода .rmdir():

```
>>> CreateExample.rmdir()
>>> CreateExample.touch()
>>> CreateExample.is_file()
True
```



Для удаления файлов существует метод .unlink().

Поиск файлов

На основе имеющегося каталога создадим более сложную структуру:

```
>>> SearchingExample = CreateExample

>>> Hosting = Path(SearchingExample.parents[2],'hosting/host.txt')
>>> Hosting.parent.mkdir(parents=True, exist_ok = True)
>>> Hosting.touch()

>>> Docker = Path(SearchingExample.parents[1],'Docker/desk.txt')
>>> Docker.parent.mkdir(parents=True, exist_ok = True)
>>> Docker.touch()
```

Мы создали и получили такую структуру (начиная с C:\Users\Blog\Timeweb):

```
Cloud
|— Pathlib
|    `— file.txt
`— Docker
       `— desk.txt
Hosting
`— host.txt
```

Использование модуля pathlib для поиска файлов подразумевает применение цикла for и метода .glob():

```
>>> for file_cloud in SearchingExample.parents[2].glob('*.txt'):
    print(file_cloud)
```

Этот код не смог ничего найти, так как он не рассматривал вложенные папки. Чтобы он корректно работал, необходимо его немного изменить:

```
>>> for file_cloud in SearchingExample.parents[2].glob('**/*.txt'):
    print(file_cloud)
...
...
C:\Users\Blog\Timeweb\Cloud\Docker\desk.txt
C:\Users\Blog\Timeweb\Cloud\Pathlib\file.txt
C:\Users\Blog\Timeweb\hosting\host.txt
```

Чтение и запись в файл

Обе операции можно выполнять как в текстовом, так и в бинарном режиме. Мы сосредоточимся на первом варианте. Для работы с содержимым файла pathlib предоставляет 4 метода:

- Чтение: .read_text() и .read_bytes();
- Запись: .write_text() и .write_bytes();

Запишем какую-нибудь важную информацию в файл, которую просто необходимо запомнить. Например, «Timeweb Cloud предлагает очень крутые облачные серверы!». Действительно, такое лучше где-нибудь записать:

```
>>> WRExample = SearchingExample

>>> WRExample.is_file()
True
>>> WRExample.write_text('Timeweb Cloud предлагает очень крутые облачные серве
55 #Длина сообщения
>>> WRExample.read_text()
'Timeweb Cloud предлагает очень крутые облачные серверы!'
```