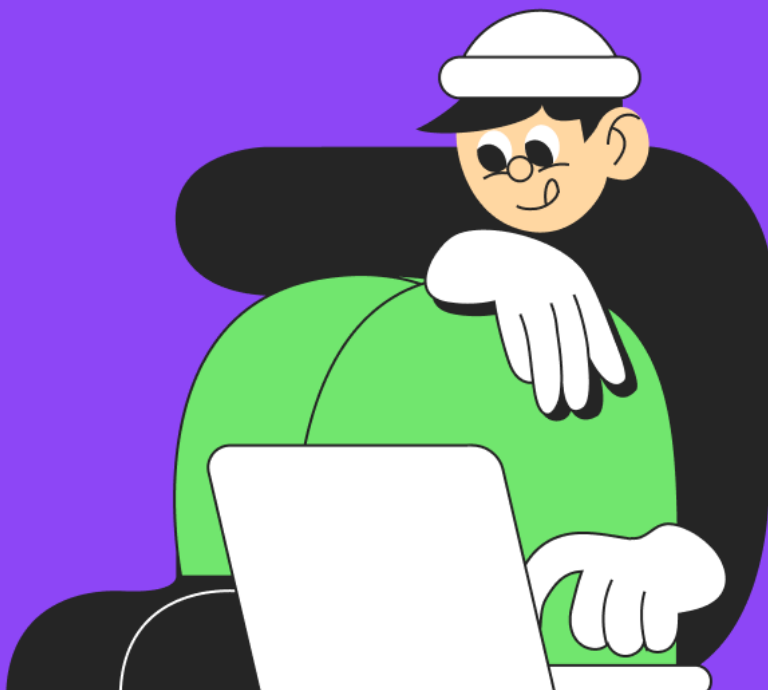
GeekBrains

Лекция 7. Файлы и файловая система

Погружение в Python



Оглавление

На этой лекции мы

1. Разберёмся в особенностях работы с файлами и каталогами в Python
2. Изучим функцию open для работы с содержимым файла
3. Узнаем о возможностях стандартной библиотеки для работы с файлами и каталогами

Дополнительные материалы к лекции

Стандартная библиотека Python. Документация.
https://docs.python.org/3/library/index.html

Краткая выжимка, о чём говорилось в предыдущей лекции

На прошлой лекции мы:

1. Разобрали работу с модулями в Python
2. Изучили особенности импорта объектов в проект
3. Узнали о встроенных модулях и возможностях по созданию своих модулей и пакетов
4. Разобрали модуль random отвечающий за генерацию случайных чисел

Термины лекции

- **Файл (англ. file)** — именованная область данных на носителе информации, используемая как базовый объект взаимодействия с данными в операционных системах.
- **Файловая система (англ. file system)** — порядок, определяющий способ организации, хранения и именования данных на носителях информации.

Подробный текст лекции

1. Файлы

Файл — это область данных на носителе информации (диске, флешке и т.п.). Файл используется как базовый объект взаимодействия с данными. Операционная система обращается к файлу по имени.

Функция open()

В Python для получения доступа файлу используют функцию open().

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
newline=None, closefd=True, opener=None)
```

Функция open имеет один обязательный параметр — file. Обычно file — это объект типа str или bytes, который представляет абсолютный или относительный путь к файлу. Относительный путь считается от каталога запуска программы. Функция возвращает файловый объект или вызывает ошибку OSError, если не получилось вернуть объект.

Рассмотрим пример.

```
f = open('text_data.txt')
print(f)
print(list(f))
```

На выходе получим примерно такой вывод из переменной f:
<_io.TextIOWrapper name='text_data.txt' mode='r' encoding='cp1251'>
Файловый объект для операций текстового ввода выводу файла text_data.txt в режиме чтения в кодировке cp1251.
Для вывода информации воспользовались превращением файлового объекта в список: list(f). Подобный приём удобен для получения всего текстового файла целиком в виде элементов списка. Каждая строка лежит в новой ячейке. Но если файл имеет большие размеры, его сохранение в список может занять много времени. Кроме того будет потрачена оперативная память. Учитывайте это при открытии файлов больше, чем свободная ОЗУ.
Если не указывать режим открытия, файл открывается для чтения текста. При этом в качестве кодировки по умолчанию используется кодировка ОС. И если с режимом чтения понятно, то с кодировкой могут быть проблемы. Вот так выглядит “Привет, мир!”, сохранённый в utf-8, но открытый в cp1251 - РµСЂёРІРµС,, РјРёСЂ!
Чтобы избежать проблем при работе с файлами рекомендуется при открытии указывать как минимум три параметра: название файла, режим и кодировку.

```
f = open('text_data.txt', 'r', encoding='utf-8')
print(f)
print(list(f))
```

🔥 **Важно!** Кодировка UTF-8 является современным стандартом для хранения и передачи текстовой информации. Если вы явно не планируете

работать с другой кодировкой, всегда указывайте encoding='utf-8' при открытии текстовых файлов. Это обеспечит переносимость вашего кода между различными платформами.

- **Режимы работы с файлами**

Рассмотрим доступные режимы работы с файлами.

- ➢ 'r' — открыть для чтения (по умолчанию)
- ➢ 'w' — открыть для записи, предварительно очистив файл
- ➢ 'x' — открыть для эксклюзивного создания. Вернёт ошибку, если файл уже существует
- ➢ 'a' — открыть для записи в конец файла, если он существует
- ➢ 'b' — двоичный режим
- ➢ 't' — текстовый режим (по умолчанию)
- ➢ '+' — открыты для обновления (чтение и запись)

По умолчанию используется режим чтения текста — “rt”. Если не указать режим “t”, он автоматически добавляется к r, w, x или a. Для чтения файла побайтно указывать режим “b” обязательно. Режимы 'w+' и 'w+b' открывают файл для чтения и записи и удаляют старое содержимое. Режимы 'r+' и 'r+b' открывают на чтение и запись без удаления старой информации.

🔥 **Важно!** При открытии файла в режиме “b” информация предаётся в виде байт. Указывать кодировку encoding при этом режиме не нужно, т.к. данные не декодируются.

- **Метод close()**

После завершения работы с файлом необходимо освободить ресурсы. Для этого вызывается метод close().

```
f = open('text_data.txt', 'a', encoding='utf-8')
f.write('Окончание файла\n')
f.close()
```

Закрытие файла гарантирует сохранение информации на носителе.

🔥 **Важно!** Если в коде отсутствует метод close(), то даже при успешном завершении программы не гарантируется сохранение всех данных в файле.

После закрытия файла происходит высвобождение ресурсов и переменная файловый объект (в наших примерах — f) становится недоступна для манипуляций с файлом.

- **Прочие необязательные параметры функции open**

buffering — определяет режим буферизации. При работе с бинарными файлами можно передать 0 — отключить буферизацию. В текстовом режиме можно передать 1 — использовать буферизацию строк. Число больше единицы определяет размер буфера в байтах для двоичных файлов. По умолчанию размер буфера подстраивается под файловую систему и обычно равен 4096 или 8192 байта.

```
f = open('bin_data', 'wb', buffering=64)
f.write(b'X' * 1200)
f.close()
```

errors — используется только в текстовом режиме и определяет поведение в случае ошибок кодирования или декодирования. Рассмотрим несколько возможных вариантов параметра:

- ➢ 'strict' — вызывает исключение ValueError в случае ошибки. Работает как значение по умолчанию.
- ➢ 'ignore' — игнорирует ошибки кодирования. При этом игнорирование ошибок может привести к потере данных.
- ➢ 'replace' — вставляет маркер замены (например, '?') там, где есть некодируемые данные.
- ➢ 'namereplace' — при записи заменяет неподдерживаемые символы последовательностями \N{...}.

```
f = open('data.txt', 'wb')
f.write('Привет, '.encode('utf-8') + 'мир!'.encode('cp1251'))
f.close()

f = open('data.txt', 'r', encoding='utf-8')
print(list(f))  # UnicodeDecodeError: 'utf-8' codec can't decode
byte 0xec in position 14: invalid continuation byte
f.close()

f = open('data.txt', 'r', encoding='utf-8', errors='replace')
print(list(f))
f.close()
```

newline — отвечает за преобразование окончания строки

closefd — указывает оставлять ли файловый дескриптор открытым при закрытии файла
opener — позволяет передать пользовательскую функцию для открытия файла.

Подробнее об этих параметрах можно прочитать в официальной документации, если возникнет необходимость.

Менеджер контекста with open

Если после открытия файла в коде возникнет ошибка, строка f.close() не будет выполнена. В результате ресурсы не освободятся, возможно возникнут проблемы в работе с открываемым файлом. Чтобы избежать подобных ошибок используют менеджер контекста with.

```
with open('text_data.txt', 'r+', encoding='utf-8') as f:
    print(list(f))
print(f.write('Пока'))    # ValueError: I/O operation on closed file.
```

В отличии от прошлых примеров с open вместо присвоения “f =” в начале строки используем “as f:” в конце строки. Вложенный в with блок кода позволяет работать с файлом через файловый дескриптор f. При завершении вложенного блока происходит автоматическое закрытие файла. При этом даже в случае ошибки внутри блока сначала будет закрыт файл, а только потом произведено аварийное завершение программы.

Обычно вариант написания кода с менеджером контекста with предпочтительнее, чем напрямую с open. Но если вам надо одновременно работать с несколькими файлами, вариант с open позволит избежать нескольких уровней вложенности. В этом случае не стоит забывать про закрытие всех файлов.

Однако менеджер контекста позволяет одновременно открыть несколько файлов следующим образом:

```
with open('text_data.txt', 'r+', encoding='utf-8') as f1, \
        open('bin_data', 'rb') as f2, \
                        open('data.txt',   'r',   encoding='utf-8', errors='backslashreplace') as f3:
    print(list(f1))
```

```
    print(list(f2))
    print(list(f3))
```

Обратите внимание на запятые между open. И при переносе кода на новую строку обязательно указывать обратную косую черту (бэкслеш). Так Python поймёт, что это одна длинная строка с переносами, а не несколько отдельных.

Начиная с Python 3.10 менеджер контекста поддерживает круглые скобки для группировки нескольких операторов по строкам:

```
with (
        open('text_data.txt', 'r+', encoding='utf-8') as f1,
        open('bin_data', 'rb') as f2,
                        open('data.txt',   'r',   encoding='utf-8',
errors='backslashreplace') as f3
):
    print(list(f1))
    print(list(f2))
    print(list(f3))
```

Чтение файла: целиком, через генератор

Рассмотрим подробнее варианты чтения информации из файла.

- **Чтение в список**

```
with open('text_data.txt', 'r', encoding='utf-8') as f:
    print(list(f))
```

Мы уже использовали приём чтения всего файла построчно в ячейки списка. Медленная по времени и затратная по памяти операция. Но обратите внимание на окончание каждой строки. Символ “\n” означает перенос строки. Разные ОС для обозначения переноса строки используют “\n”, “\r\n” или даже “\r”. Python берёт работу по конвертации символа окончания строки на себя. При чтении любое окончание заменяется на “\n”. А при записи текстового файла окончание “\n” заменяется на окончание для вашей ОС.

- **Чтение методом read**

Ещё один вариант чтения файла — метод read().

read(n=-1) — читает n символов или n байт информации из файла. Если n отрицательное или не указана, читает весь файл. Попытка чтения будет даже в том случае, когда файл больше оперативной памяти.

```
with open('text_data.txt', 'r', encoding='utf-8') as f:
    res = f.read()
    print(f'Читаем первый раз\n{res}')
    res = f.read()
    print(f'Читаем второй раз\n{res}')
```

Если прочитать файл до конца, повторные попытки чтения не будут вызывать ошибку. Метод будет возвращать пустую строку.
Также при чтении через read не добавляются символы переноса строки. Точнее мы не видим “\n”, а видим перенос строки на новую строчку.

```
SIZE = 100
with open('text_data.txt', 'r', encoding='utf-8') as f:
    while res := f.read(SIZE):
        print(res)
```

При чтении файла блоками фиксированного размера можно воспользоваться циклом while. Дочитав до конца в переменную попадёт пустая строка, которая в цикле будет интерпретирована как ложь и завершит тело цикла.

- **Чтение методом readline**

Для чтения текстового файла построчно используют метод readline.

```
with open('text_data.txt', 'r', encoding='utf-8') as f:
    while res := f.readline():
        print(res)
```

Обратите внимание на вывод информации. Между строками остаётся по пустой строке. При чтении readline возвращает строку с символом переноса на конце. Функция print выводит их на печать и автоматически добавляет переход на следующую строку. Получаем пару “\n\n”.

```
SIZE = 100
with open('text_data.txt', 'r', encoding='utf-8') as f:
    while res := f.readline(SIZE):
        print(res)
```

Передача положительного числа в качестве аргумента задаёт границу для длины строки. Если строка короче границы, она возвращается целиком. А если больше, то возвращается часть строки заданного размера. При этом следующий вызов метода вернёт продолжение строки снова фиксированного размера или остаток до конца, если она короче.

- **Чтение циклом for**

Вместо метода readline без аргумента можно использовать более короткую запись с циклом for

```
with open('text_data.txt', 'r', encoding='utf-8') as f:
    for line in f:
        print(line, end='')
```

Файл построчно попадает в переменную line. А для того чтобы избавиться от пустых строк отключили перенос строки в функции print.

🔥 **Важно!** Символ переноса строки сохранился в конце каждой строки. Если вам необходимо обработать строку без переносов, можно использовать срезы line[:-1] или метод замены line.replace('\n', '')

```
SIZE = 100
with open('text_data.txt', 'r', encoding='utf-8') as f:
    for line in f:
        print(line[:-1])
        print(line.replace('\n', ''))
```

Запись и добавление в файл

С режимами записи мы уже познакомились.

- ➢ w — создаём новый пустой файл для записи. Если файл существует, открываем его для записи и удаляем данные, которые в нём хранились.
- ➢ x — создаём новый пустой файл для записи. Если файл существует, вызываем ошибку.
- ➢ a — открываем существующий файл для записи в конец, добавления данных. Если файл не существует, создаём новый файл и записываем в него.

- **Запись методом write**

Метод write принимает на вход строку или набор байт в зависимости от того как вы открыли файл. После записи метод возвращает количество записанной информации.

```
text = 'Lorem ipsum dolor sit amet, consectetur adipisicing elit.'
with open('new_data.txt', 'a', encoding='utf-8') as f:
    res = f.write(text)
    print(f'{res = }\n{len(text) = }')
```

Метод не добавляет символ перехода на новую строку. Если произвести несколько записей, они “склеиваются” в файле.

```
text = ['Lorem ipsum dolor sit amet, consectetur adipisicing elit.',
        'Consequatur debitis explicabo laboriosam sint suscipit temporibus veniam?',
        'Accusantium alias amet fugit iste neque non odit quia saepe totam velit?', ]
with open('new_data.txt', 'a', encoding='utf-8') as f:
    for line in text:
        res = f.write(line)
        print(f'{res = }\n{len(line) = }')
```

Если каждая строка должна сохранятся в файле с новой строки, необходимо самостоятельно добавить символ переноса - \n

```
text = ['Lorem ipsum dolor sit amet, consectetur adipisicing elit.',
        'Consequatur debitis explicabo laboriosam sint suscipit temporibus veniam?',
        'Accusantium alias amet fugit iste neque non odit quia saepe totam velit?', ]
with open('new_data.txt', 'a', encoding='utf-8') as f:
    for line in text:
        res = f.write(f'{line}\n')
        print(f'{res = }\n{len(line) = }')
```

- **Запись методом writelines**

Метод writelines принимает в качестве аргумента последовательность и записывает каждый элемент в файл. Элементы последовательности должны быть строками или байтами в зависимости от режима записи.

В отличии от write этот метод ничего не возвращает.

```
text = ['Lorem ipsum dolor sit amet, consectetur adipisicing elit.',
          'Consequatur debitis explicabo laboriosam sint suscipit temporibus veniam?',
          'Accusantium alias amet fugit iste neque non odit quia saepe totam velit?', ]
with open('new_data.txt', 'a', encoding='utf-8') as f:
    f.writelines('\n'.join(text))
```

Для того чтобы каждый элемент списка text сохранялся в файле с новой строки воспользовались строковым методом join. writelines не добавляет переноса между элементами последовательности.

- **print в файл**

Функция print по умолчанию выводит информацию в поток вывода. Обычно это консоль. Но можно явно передать файловый объект для печати в файл. Для этого надо воспользоваться ключевым параметром file.

```
text = ['Lorem ipsum dolor sit amet, consectetur adipisicing elit.',
          'Consequatur debitis explicabo laboriosam sint suscipit temporibus veniam?',
          'Accusantium alias amet fugit iste neque non odit quia saepe totam velit?', ]
with open('new_data.txt', 'a', encoding='utf-8') as f:
    for line in text:
        print(line, file=f)
```

В отличии от методов записи в файл, функция print добавляет перенос строки. Кроме того ничто не мешает явно изменить параметр end функции.

```
text = ['Lorem ipsum dolor sit amet, consectetur adipisicing elit.',
          'Consequatur debitis explicabo laboriosam sint suscipit temporibus veniam?',
          'Accusantium alias amet fugit iste neque non odit quia saepe totam velit?', ]
with open('new_data.txt', 'a', encoding='utf-8') as f:
    for line in text:
        print(line, end='***\n##', file=f)
```

Методы перемещения в файле

При работе с файлом можно управлять положением файлового объекта в открытом файле. Действия напоминают движение курсора в строке стрелками влево и вправо.

- **Метод tell**

Метод tell возвращает текущую позицию в файле.

```
text = ['Lorem ipsum dolor sit amet, consectetur adipisicing elit.',
          'Consequatur debitis explicabo laboriosam sint suscipit temporibus veniam?',
          'Accusantium alias amet fugit iste neque non odit quia saepe totam velit?', ]
with open('new_data.txt', 'w', encoding='utf-8') as f:
    print(f.tell())
    for line in text:
        f.write(f'{line}\n')
        print(f.tell())
    print(f.tell())
print(f.tell())  # ValueError: I/O operation on closed file.
```

Для пустого файла возвращается ноль — начало файла. По мере записи или чтения информации позиция сдвигается к концу файла.
Метод использую для определения в каком месте файла будет произведены чтение или запись.

- **Метод seek**

Метод seek позволяет изменить положение “курсора” в файле.
seek(offset, whence=0), где offset — смещение относительно опорной точки, whence - способ выбора опороной точки.

- whence = 0 - отсчёт от начала файла
- whence = 1 - отсчёт от текущей позиции в файле
- whence = 2 - отсчёт от конца файла

🔥 **Важно!** Значения 1 и 2 допустимы только для работы с бинарными файлами. Исключение seek(0, 2) для перехода в конец текстового файла.

Метод возвращает новую позицию “курсора”.

```
last = before = 0
text = ['Lorem ipsum dolor sit amet, consectetur adipisicing elit.',
        'Consequatur debitis explicabo laboriosam sint suscipit temporibus veniam?',
        'Accusantium alias amet fugit iste neque non odit quia saepe totam velit?', ]
with open('new_data.txt', 'r+', encoding='utf-8') as f:
    while line := f.readline():
        last, before = f.tell(), last
    print(f'{last = }, {before = }')
    print(f'{f.seek(before, 0) = }')
    f.write('\n'.join(text))
```

В примере выше мы открыли текстовый файл для одновременного чтения и записи. Переменные last и before хранят позиции двух последних прочитанных строк. Дочитав файл в цикле while до конца изменяем позицию “курсора” на начало последней строки и начинаем запись. Таким образом мы сохранили все строки файла кроме последней и записали новый текст в конец.

- **Метод truncate**

truncate(size=None) — метод изменяет размер файла. Если не передать значение в параметр size будет удалена часть файла от текущей позиции до конца. Метод возвращает позицию после изменения файла.

```
last = before = 0
with open('new_data.txt', 'r+', encoding='utf-8') as f:
    while line := f.readline():
        last, before = f.tell(), last
    print(f.seek(before, 0))
    print(f.truncate())
```

Если передать параметр size метод изменяет размер файла до указанного числа символов или байт от начала файла.

```
size = 64
with open('new_data.txt', 'r+', encoding='utf-8') as f:
    print(f.truncate(size))
```

Если size меньше размера файла, происходит усечение файла. Если size больше размера файла, он увеличивается до указанного размера. При этом добавленный

размер обычно заполняется нулевыми байтами. Заполнение зависит от конкретной ОС.

Задание

Перед вами несколько строк кода. Что будет хранится в файле после завершения работы программы? И что будет выведено на печать? У вас 3 минуты.

```
start = 10
stop = 100
with open('data.bin', 'bw+') as f:
    for i in range(start, stop + 1):
        f.write(str(i).encode('utf-8'))
        if i % 3 == 0:
            f.seek(-2, 1)
    f.truncate(stop)
    f.seek(0)
    res = f.read(start)
    print(res.decode('utf-8'))
```

2. Файловая система

Далее рассмотрим работу с файловой системой средствами Python. В этой части разберём некоторые модули стандартной библиотеки для решения задач работы с файловой системой.

1. Модуль os позволяет использовать функции, зависящие от файловой системы.
2. Для работы с путями используют модуль os.path.
3. Модуль pathlib позволяет работать с путями файловой системы в ООП стиле.
4. Модуль shutil обеспечивает высокоуровневые операции над файлами и группами файлов.

Если задачу можно решить несколькими способами, рассмотрим варианты одновременно.

Работа с каталогами

Операции по работе с файлами и папками должны иметь полный, абсолютный путь к объекту начиная от корневой директории. Или же можно задать относительный путь. Так мы делали в первой части лекции. Файл с кодом и файлы для чтения и записи находились в одном каталоге.

- **Текущий каталог**

Для получения информации о текущем каталоге можно использовать модуль os или pathlib

```
import os
from pathlib import Path

print(os.getcwd())
print(Path.cwd())
```

Обычно текущим является каталог из которого был запущен python.
Для изменения текущего каталога можно воспользоваться функцией os.chdir. Она принимает на вход абсолютный или относительный путь до нового текущего каталога.

```
import os
from pathlib import Path

print(os.getcwd())
print(Path.cwd())
os.chdir('../..')
print(os.getcwd())
print(Path.cwd())
```

- **Создание каталогов**

Для создания каталога снова можно воспользоваться двумя модулями

```
import os
from pathlib import Path
```

```
os.mkdir('new_os_dir')
Path('new_path_dir').mkdir()
```

Представленный код создаёт каталог в текущей директории. А если необходимо создать несколько вложенных друг в друга каталогов, код будет следующим:

```
import os
from pathlib import Path

os.makedirs('dir/other_dir/new_os_dir')
Path('some_dir/dir/new_path_dir').mkdir()  # FileNotFoundError
Path('some_dir/dir/new_path_dir').mkdir(parents=True)
```

Модуль os предоставляет другую функцию — makedirs. Модуль path работает с тем же методом mkdir. Но если в цепочке каталогов будет несуществующий, получим ошибку FileNotFoundError. Дополнительный параметр parents=True указывает на необходимость создать всех недостающих родительских каталогов.

- **Удаление каталогов**

Для удаления одного каталога подойдут следующие функция и метод

```
import os
from pathlib import Path

# os.rmdir('dir')  # OSError
# Path('some_dir').rmdir()  # OSError
os.rmdir('dir/other_dir/new_os_dir')
Path('some_dir/dir/new_path_dir').rmdir()
```

🔥 **Важно!** Удалить можно лишь пустой каталог. Если внутри удаляемого каталога есть другие каталоги или файлы, возникнет ошибка OSError.

Обратите внимание, что при передаче цепочки каталогов удаляется один, последний из перечисленных. Родительские каталоги остаются без изменений.

Если необходимо удалить каталог со всем его содержимым (вложенные каталоги и файлы), подойдёт функция из модуля shutil

```
import shutil

shutil.rmtree('dir/other_dir')
shutil.rmtree('some_dir')
```

В первом случае будет удалена директория other_dir со всем содержимым. Директория dir останется на месте.

Во втором случае удаляется каталог some_dir и его содержимое.

- **Формирование пути**

В операционной системе Windows для указания пути используется обратный слеш \. В Unix системах путь разделяется слешем. Чтобы программа работала одинаково на любой ОС рекомендуется использовать специальную функцию join из os.path для склеивания путей.
Модуль pathlib использует более понятный приём с переопределением операции деления.

```
import os
from pathlib import Path

file_1 = os.path.join(os.getcwd(), 'dir', 'new_file.txt')
print(f'{file_1 = }\n{file_1}')

file_2 = Path().cwd() / 'dir' / 'new_file.txt'
print(f'{file_2 = }\n{file_2}')
```

Оба варианта используют разные способы получения результата и хранения информации. Но результат мы получили один и тот же — путь до файла new_file.txt в каталоге dir текущего каталога.

Чтение данных о каталогах

Для получения информации о том какие директории и файлы находятся в текущем каталоге можно воспользоваться следующими вариантами кода.

```
import os
from pathlib import Path

print(os.listdir())

p = Path(Path().cwd())
for obj in p.iterdir():
    print(obj)
```

Функция listdir возвращает список файлов и каталогов. Метод iterdir у экземпляра класса Path является генератором. В цикле он возвращает объекты из выбранной директории.

- **Проверка на директорию, файл и ссылку**

Получив информацию о содержимом текущего каталога зачастую требуется уточнить что перед нами. В каталогах можно хранить другие каталоги и файлы. В файлах содержатся данные. А символьные ссылки указывают на каталоги и файлы из других мест.
Рассмотрим варианты получения информации об объектах, полученных в примере кода выше.

```
import os
from pathlib import Path

dir_list = os.listdir()
for obj in dir_list:
    print(f'{os.path.isdir(obj) = }', end='\t')
    print(f'{os.path.isfile(obj) = }', end='\t')
    print(f'{os.path.islink(obj) = }', end='\t')
    print(f'{obj = }')

p = Path(Path().cwd())
for obj in p.iterdir():
    print(f'{obj.is_dir() = }', end='\t')
    print(f'{obj.is_file() = }', end='\t')
    print(f'{obj.is_symlink() = }', end='\t')
    print(f'{obj = }')
```

Обратите внимание, что при работе с модулем os мы получаем объекты str типа. Строки передаются в другие функции для получения результата.
При работе с pathlib мы итерируемся по объектам Path. А если быть точным по экземплярам класс Path той операционной системы, в которой работает код. Методы проверки принадлежности являются частью экземпляра.

- **Обход папок через os.walk()**

Функция os.walk рекурсивно обходит каталоги от переданного в качестве аргумента до самого нижнего уровня вложенности.

```
import os

for dir_path, dir_name, file_name in os.walk(os.getcwd()):
   print(f'{dir_path = }\n{dir_name = }\n{file_name = }\n')
```

Функция возвращает кортеж из трёх значений:

- ➢ dir_path — абсолютный путь до каталога
- ➢ dir_names — список с названиями всех каталогов, находящихся в dir_path

➢ dir_names — список с названиями всех файлов, находящихся в dir_path

Вывод продолжается до тех пор пока не будет возвращена информация обо всех директориях, т.е. каждая директория из dir_names передаётся в os.walk и оказывается в dir_path.

Работа с файлами

Рассмотрим базовые операции по работе с файлами как с отдельными объектами.

- **Переименование файлов**

Переименование файла подразумевает сохранение неизменным файла в файловой таблице и в содержимом файла, но изменение его имени. Для переименования можно воспользоваться следующим кодом.

```
import os
from pathlib import Path

os.rename('old_name.py', 'new_name.py')

p = Path('old_file.py')
p.rename('new_file.py')

Path('new_file.py').rename('newest_file.py')
```

Функция rename принимает на вход два обязательных аргумента — исходное имя файла и новое имя. Важно чтобы файл существовал, а новое имя не было занято.
Модуль pathlib позволяет сделать переименование через создание экземпляра класса Path с указанием исходного файла. А затем вызов метода rename задаёт новое имя. При этом операции могут быть объединены в одну строку через точечную нотацию и без явного сохранения экземпляра класса в переменной.

- **Перемещение файлов**

Перемещение файла подразумевает изменение его позиции в каталоге файловой системы. В процессе перемещения файл может быть переименован.

```
import os
from pathlib import Path
```

```
os.replace('newest_file.py',   os.path.join(os.getcwd(),   'dir',
'new_name.py'))

old_file = Path('new_name.py')
new_file = old_file.replace(Path.cwd() / 'new_os_dir' / old_file)
```

Для исходного файла явно не указывали директорию, где он расположен. При переносе была указана текущая директория как отправная точка. Оба варианта имеют одинаковый эффект.
Также при работе через os мы присвоили файлу новое имя. А при использовании модуля pathlib указали, что после переноса файл должен сохранить старое имя.

- **Копирование файлов**

Для копирования файлов лучше всего подходит модуль shutil, который предоставляет ряд высокоуровневых операций.

```
import shutil

shutil.copy('one.txt', 'dir')
shutil.copy2('two.txt', 'dir')
```

Функции copy и copy2 работают схожим образом. Они принимают файл для копирования и целевой каталог. Если такого каталога не существует, функция попытается присвоить копируемому файлу имя “цели”.
Отличие состоит в том, что функция copy2 помимо содержимого файла пытается скопировать и связанные с ним метаданные.

Если стоит задача скопировать каталог со всем его содержимым в новое место, модуль предоставляет функции copytree.

```
import shutil

shutil.copytree('dir', 'one_more_dir')
```

Функция copytree рекурсивно обходит указанный каталог и копирует его содержимое в новое место.

- **Удаление файлов**

Вариант удаления всего каталога целиком с содержимым мы уже рассматривали сегодня.

```
import shutil
```

```
shutil.rmtree('dir')
```

Если же необходимо удалить один файл, можно воспользоваться следующими вариантами:

```
import os
from pathlib import Path

os.remove('one_more_dir/one.txt')
Path('one_more_dir/one_more.txt').unlink()
```

И функция remove и метод unlink пытаются удалить файл по указанному пути.

Задание

Перед вами несколько строчек кода. Какие каталоги и файлы будут созданы после его выполнения? У вас три минуты.

```
import os
import shutil
from pathlib import Path

for i in range(10):
    with open(f'file_{i}.txt', 'w', encoding='utf-8') as f:
        f.write('Hello world!')
os.mkdir('new_dir')
for i in range(2, 10, 2):
    f = Path(f'file_{i}.txt')
    f.replace('new_dir' / f)
shutil.copytree('new_dir', Path.cwd() / 'dir_new')
```

Вывод

На этой лекции мы:

1. Разобрались в особенностях работы с файлами и каталогами в Python

2. Изучили функцию open для работы с содержимым файла
3. Узнали о возможностях стандартной библиотеки для работы с файлами и каталогами

Краткий анонс следующей лекции

1. Разберёмся в сериализации и десериализации данных
2. Изучим самый популярный формат сериализации — JSON
3. Узнаем о чтении и записи таблиц в формате CSV
4. Разберёмся с внутренним сериализатором Python — модулем pickle

Домашнее задание

1. Загляните в документацию к Python и изучите особенности и нюансы работы с файлами и каталогами в рассматриваемых на уроке модулях.