**ДОКЛАД НА ТЕМУ**

**«ФАЙЛЫ КАК ТИП ДАННЫХ**

**В PYTHON»**

**Подготовила доклад студентка 1 курса**

**Икоева Ника**

Файл - именованная ячейка хранилища на компьютере, управляемая операционной системой. Файловые объекты кое-чем необычны:

1. Считаются основным типом, т.к. создаются встроенной функцией, но при этом отличаются от чисел, последовательностей, функций.
2. не реагирует на операции выражений

**ТЕКСТОВЫЕ И БИНАРНЫЕ ФАЙЛЫ**

В Python 3.X и 2.X тип файла определяется вторым аргументом функции open, строкой режима – наличие в ней b означает двоичный (бинарный) файл. В Python всегда поддерживались как текстовые, так и двоичные файлы, но в Python 3.X между ними имеется более строгое разграничение.

• Текстовые файлы представляют содержимое как нормальные строки str с человекочитаемым текстом, автоматически выполняют кодирование и декодирование Unicode. Блокнот и другие стандартные редакторы умеют читать и редактировать этот тип файлов.

Текст может храниться в двух форматах: (.txt) — простой текст и (.rtf) — «формат обогащенного текста».

• В бинарных файлах данные отображаются в закодированной форме (с использованием только нулей (0) и единиц (1) вместо простых символов).

В большинстве случаев это просто последовательности битов. Они хранятся в формате .bin. Любую операцию с файлом можно разбить на три крупных этапа:

1. Открытие файла
2. Выполнение операции (запись, чтение)
3. Закрытие файла

**ОСНОВНЫЕ ОПЕРАЦИИ В ФАЙЛОВЫХ ЭЛЕМЕНТАХ**

**Открытие файла**

Для работы с файлами в Python используется функция open(). Она принимает два аргумента: имя файла и режим открытия файла:

1. Имя файла-абсолютный или относительный путь к файлу. Если файл находится в текущем рабочем каталоге, то нам нужен относительный путь, иначе-абсолютный путь, т.е. диск, файлы, которые нужно открыть, чтобы дойти до нужного файла.
2. Режим обработки-режим, в котором открывается файл. Для работы с файлами в Python используется функция open(). Она принимает два аргумента: имя файла и режим открытия файла. Режимы открытия могут быть следующие:

‘r’ — чтение (по умолчанию)

‘w’ — запись (если файл не существует, он будет создан)

‘a’ — добавление (открывает файл для добавления новых данных)

‘x’ — создание (создает новый файл, если он уже существует, вызывается исключение)

‘b’ — двоичный (бинарный) режим (используется для работы с двоичными файлами, например, изображениями)

Важно понимать, что 2 первых аргумента режима обязательно должны быть строками.

Пример открытия файла для чтения:

|  |  |
| --- | --- |
|  | file = open('example.txt', 'r') |
|  |  |

Также функцию open() можно использовать без второго аргумента. В таком случае режим обработки по умолчанию будет ' r '

**Методы чтения файла**

Для чтения содержимого файла можно использовать несколько методов:

* read(N) - считывает из файла N количество символов

(или весь файл, если N не указана)

* readline() - считывает одну строку из файла
* readlines() - считывает все строки файла в список

Пример чтения всего файла:

|  |  |
| --- | --- |
| 1  2  3  4 | file = open('example.txt', 'r')  for line in file:      print(line)  file.close() |

Пример чтения файла построчно:

|  |  |
| --- | --- |
| 1  2  3  4 | file = open('example.txt', 'r')  content = file.read()  print(content)  file.close() |

**Методы записи в файла**

Для записи данных в файл используется метод write(text). Он принимает один аргумент — текст, который нужно записать в файл.

Если файл открыт в режиме ‘w’, то все существующие данные будут стерты, а новые добавлены.

Пример записи текста в файл:

|  |  |
| --- | --- |
| 1  2  3 | file = open('example.txt', 'w')  file.write('Hello, Python!')  file.close() |

Пример записи целого списка в файл:

|  |  |
| --- | --- |
| 1  2  3 | file = open('example.txt', 'w')  file.writelines(AnyList)  file.close() |

Использование контекстных менеджеров позволяет автоматически закрыть файл после выхода из блока with. Это удобно и позволяет избежать ошибок, связанных с незакрытыми файлами.

Пример чтения файла с использованием контекстного менеджера:

|  |  |
| --- | --- |
| 1  2  3 | with open('example.txt', 'r') as file:      content = file.read()      print(content) |

Использование контекстных менеджеров позволяет автоматически закрыть файл после выхода из блока with. Это удобно и позволяет избежать ошибок, связанных с незакрытыми файлами.

Пример чтения файла с использованием контекстного менеджера:

**Использование контекстного менеджера**

Использование контекстных менеджеров позволяет автоматически закрыть файл после выхода из блока with. Это удобно и позволяет избежать ошибок, связанных с незакрытыми файлами.

Пример чтения файла с использованием контекстного менеджера:

|  |  |
| --- | --- |
| 1  2  3 | with open('example.txt', 'r') as file:      content = file.read()      print(content) |

* Каждая вводимая в файл строка должна заканчиваться символом новой строки \n
* Важно знать, что файловые методы Python сообщают о том, что достигнут конец файла, выводя строки, содержащие только символ новой строки, а не по-настоящему пустые строки.

**ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ ФАЙЛОВ**

После создания файлового объекта посредством встроенной функции open() можно вызывать его методы для чтения и записи для взаимодействия с ним. Во всех случаях текст из файла принимает форму строк в программах Python; методы чтения файла возвращают его содержимое в виде строк.

Ниже приведено несколько замечаний относительно применения файлов:

* Файловые итераторы лучше всего подходят для чтения строк

Наилучший способ чтения строк из текстового файла – не читать файл вообще, использую вместо этого цикл for. У файлов есть итератор, который автоматически читает по одной строке за раз в цикле for.

* Содержимое является строками, а не объектами

Данные, прочитанные из файла, всегда возвращаются в программу в виде строки, поэтому если нужен какой-то другой тип данных, его нужно самостоятельно изменить. По этой причине при работе с файлами будут полезными инструменты, предназначенные для преобразования объектов в и из строк (например, int, float, str и выражение и метод форматирования строк).

Список литературы:

1. Лутц, М. «Изучаем Python» [Том 1, 5-е издание] [2019]
2. <https://sky.pro/media/kak-rabotat-s-fajlami-v-python/>
3. https://habr.com/ru/articles/835216/