Модуль 3

Урок 1. Модули и пакеты, работа с файлами и форматированный вывод

Многие языки программирования предоставляют классы для работы с файлами и директориями проекта. Язык **Python** обладает множеством классов для записи и чтения данных из файлов.

Файл — это всего лишь набор данных, сохраненный в виде последовательности битов на компьютере. Информация хранится в куче данных (структура данных) и имеет название «имя файла» (filename).

В Python существует два типа файлов:

Текстовые

Бинарные

Текстовые файлы

Это файлы с человекочитаемым содержимым. В них хранятся последовательности символов, которые понимает человек. 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».

Бинарные файлы

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.

Работа с файлами

При работе с файлами всегда необходимо помнить две вещи:

1. Перед началом работы с файлом его необходимо открыть;
2. После завершения работы с файлом его необходимо закрыть.

Если файл не открыт или же неверно открыт, то вы не можете полноценно работать с его содержимым.

С закрытием все проще, но и коварнее. Если вы не закроете файл, то программа будет работать верно, тем не менее, чем больше будет открытых файлов, тем больше программа будет перегружена и в какой-то момент она просто зависнет или выключиться.

Исключения и файлы

Поскольку не всегда известно будет ли файл в проекте или на компьютере пользователя, то всегда лучше открывать файлы за счёт использования исключений. Выполняйте открытие файлов в блоке **try except** и тем самым вы обезопасите себя от любых непредвиденных обстоятельств.

Работа с файлами

Для открытия файла существует функция **open**, которая открывает файл разными способами. У функции open много параметров, нам пока важны 3 аргумента: первый, это имя файла. Путь к файлу может быть относительным или абсолютным. Второй аргумент, это режим, в котором мы будем открывать файл.

Синтаксис следующий:

f = open(file_name, access_mode)

Где,

file_name = имя открываемого файла

access_mode = режим открытия файла.

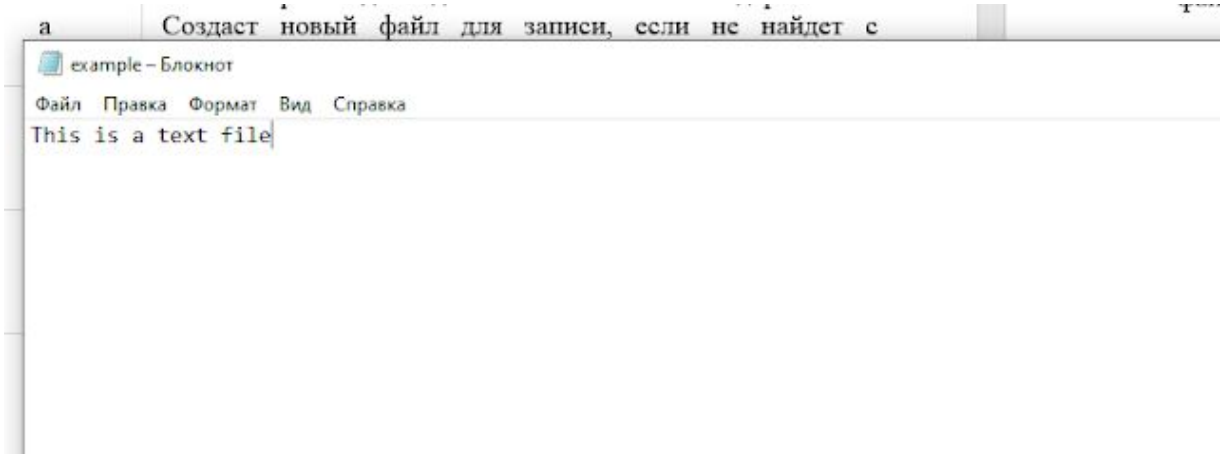
Режим	**Описание**
r	Только для чтения
w	Только для записи. Создаст новый файл, если не найдет с указанным именем
rb	Только для чтения (бинарный)
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем
r+	Для чтения и записи
rb+	Для чтения и записи (бинарный)
w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем
wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем
a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем

a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем
ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем
ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем

По умолчанию режим равен 'rt'.

И последний аргумент, encoding, нужен только в текстовом режиме чтения файла. Этот аргумент задает кодировку.

Создадим текстовый файл example.txt и сохраним его в рабочей директории.



Следующий код используется для его открытия, для вывода содержимого файла и информации о нем.

```
f = open('example.txt','r')  # открыть файл из рабочей директории в режиме чтения
fp = open('D:/example.txt','r')  # открыть файл из выбранного каталога
print(*f) # выводим содержимое файла
print(f) # выводим сведения о файле
```

Результат:

```
This is a text file
<_io.TextIOWrapper name='D:/example.txt' mode='r' encoding='cp1251'>
```

Стоит обратить внимание, что в Windows стандартной кодировкой является cp1252, а в Linux — utf-08.

Закрытие файла

Метод close()

После открытия файла в Python его нужно закрыть. Таким образом освобождаются ресурсы и убирается мусор. Python автоматически закрывает файл, когда объект присваивается другому файлу.

Существуют следующие способы:

Способ №1

Проще всего после открытия файла закрыть его, используя метод close().

```
f = open('example.txt','r')
# работа с файлом
f.close()
```

После закрытия этот файл нельзя будет использовать до тех пор, пока заново его не открыть.

Способ №2

Также можно написать try/finally, которое гарантирует, что если после открытия файла операции с ним приводят к исключениям, он закроется автоматически. Без него программа завершается некорректно.

Вот как сделать это исключение:

```
f = open('example.txt','r')
try:
  # работа с файлом
finally:
  f.close()
```

Файл нужно открыть до инструкции try, потому что если инструкция open сама по себе вызовет ошибку, то файл не будет открываться для последующего закрытия. Этот метод гарантирует, что если операции над файлом вызовут исключения, то он закроется до того, как программа остановится.

Способ №3

Инструкция with

Еще один подход — использовать инструкцию with, которая упрощает обработку исключений с помощью инкапсуляции начальных операций, а также задач по закрытию и очистке. В таком случае инструкция close не нужна, потому что with автоматически закроет файл.

Вот как это реализовать в коде.

```
with open('example.txt') as f:
   # работа с файлом
```

Чтение и запись файлов в Python

В Python файлы можно читать или записывать информацию в них с помощью соответствующих режимов.

Функция read()

Функция read() используется для чтения содержимого файла после открытия его в режиме чтения (r).

```
file.read(size)
```

- file = объект файла
- size = количество символов, которые нужно прочитать. Если не указать, то файл прочитается целиком.

```
f = open('example.txt','r')
f.read(7)  # чтение 7 символов из example.txt
'This is '
```

Интерпретатор прочитал 7 символов файла и если снова использовать функцию read(), то чтение начнется с 8-го символа.

```
f.read(7)  # чтение следующих 7 символов
' a text'
```

Функция readline()

Функция readline() используется для построчного чтения содержимого файла. Она используется для крупных файлов. С ее помощью можно получать доступ к любой строке в любой момент.

Создадим файл example.txt с нескольким строками:

This is line1.
This is line2.
This is line3.

Посмотрим, как функция readline() работает в example.txt.

```
x = open('D://example.txt','r')
print(x.readline()) # прочитать первую строку
z = x.readline()  # прочитать вторую строку
print(z)
z = x.readline(3)  # прочитать три текущих символа строки и записать в переменную
print(z)
x.close()
x = open('D://example.txt','r')
print(x.readlines()) # прочитать все строки
x.close()
```

Результат:

```
This is line1.

This is line2.

Thi
['This is line1.\n', 'This is line2.\n', 'This is line3.\n']
```

Обратите внимание, как в последнем случае строки отделены друг от друга.

Функция write()

Функция write() используется для записи в файлы Python, открытые в режиме записи.

Если пытаться открыть файл, которого не существует, в этом режиме, тогда будет создан новый.

```
file.write(string)
```

Предположим, файла xyz.txt не существует. Он будет создан при попытке открыть его в режиме чтения.

```
f = open(‘example.txt','w')  # открытие в режиме записи
f.write('Hello \n World')  # запись Hello World в файл
f.close()  # закрытие файла
```

Переименование файлов в Python

Функция rename()

Функция rename() используется для переименовывания файлов в Python. Для ее использования сперва нужно импортировать модуль os.

```
import os
os.rename(src,dest)
```

- src = файл, который нужно переименовать
- dest = новое имя файла

```
import os
# переименование xyz.txt в abc.txt
os.rename("xyz.txt","abc.txt")
```

Текущая позиция в файлах Python

В Python возможно узнать текущую позицию в файле с помощью функции tell(). Таким же образом можно изменить текущую позицию командой seek().

```
f = open('example.txt')  # example.txt, который мы создали ранее
```

```
print(f.read(4)) # давайте сначала перейдем к 4-й позиции
print(f.tell())  # возвращает текущую позицию
f.seek(0,0)  # вернем положение на 0 снова
```

Методы файла в Python

file.close()	закрывает открытый файл
file.fileno()	возвращает целочисленный дескриптор файла
file.flush()	очищает внутренний буфер
file.isatty()	возвращает True, если файл привязан к терминалу
file.next()	возвращает следующую строку файла
file.read(n)	чтение первых n символов файла
file.readline()	читает одну строчку строки или файла
file.readlines()	читает и возвращает список всех строк в файле
file.seek(offset[,whene])	устанавливает текущую позицию в файле
file.seekable()	проверяет, поддерживает ли файл случайный доступ. Возвращает True, если да
file.tell()	возвращает текущую позицию в файле
file.truncate(n)	уменьшает размер файл. Если n указала, то файл обрезается до n байт, если нет — до текущей позиции
file.write(str)	добавляет строку str в файл
file.writelines(sequence)	добавляет последовательность строк в файл

Задание:

Создать текстовый файл, записать в него построчно данные, которые вводит пользователь. Окончанием ввода пусть служит символ «n».