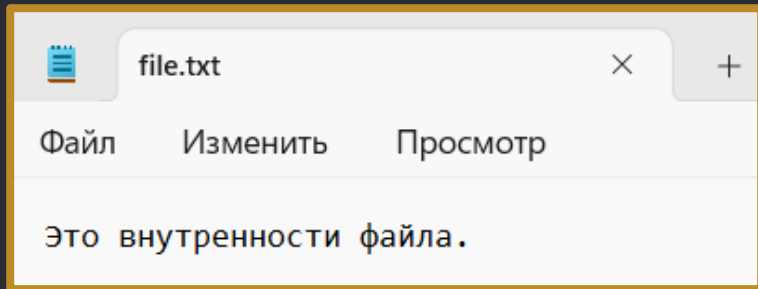


Программирование в лингвистике

Файлы, os, json, pickle, dill

Файлы в Python



```
fin = open("file.txt", "r")
read_text = fin.read()
print(read_text)
fin.close()
```

Это внутренности файла.

Python работает с файлами через систему.

Для работы с файлами у Python есть **дескриптор файла** – специальный класс, который позволяет **читать данные из файла**.

В мире Python изначально существует два типа файлов: **текстовые** (хранятся в формате .txt, .rtf) и **бинарные** (хранятся в формате .bin).

Открытие файла

```
file = open(path, mode, encoding)
```

> **path** – абсолютный или относительный путь к файлу.

> **mode** – режим работы с файлом, str. Может быть 'r' – read, 'w' – write, 'rb' – read bin, 'wb' – write bin, 'r+' – read & write, 'rb+' – read & write bin и др.

> **encoding** – кодировка файла, str. Часто следует указывать utf8.

Прежде чем начать работать с файлом, следует его **открыть**.

Функция **open()** создает объект класса **IOWrapper**, у которого есть свои методы чтения, записи и закрытия. **Path** - это путь к файлу, единственный обязательный аргумент.

```
fin = open("file.txt", "r", encoding="utf8")
```

Особенности path

```
open('C:\train\new\text.txt')
```



```
open('C:\\train\\new\\text.txt')
```

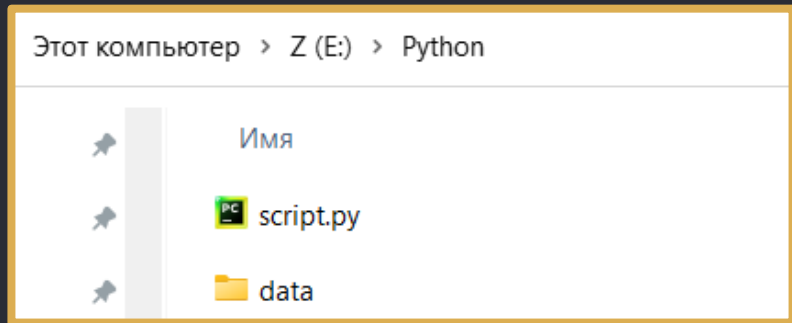


```
open(r'C:\train\new\text.txt')
```



В Windows в путях используется бэкслэш. Пути – строки, поэтому Python будет искать специальные символы вроде `\n`. Так, можно либо экранировать бэкслэш, либо ставить флаг `r`. В Linux и MacOS используется обычный слэш.

Особенности path



Здесь пути к файлу `text.txt`, находящемуся в папке `data`, для кода `script.py`, будут такими:

Абсолютный - `E:\Python\data\text.txt`

Относительный – `data\text.txt`

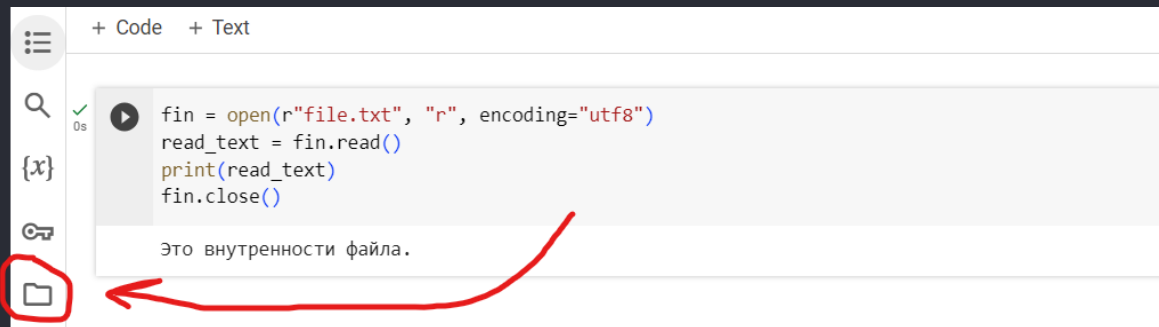
Если файл лежит прямо в той директории, где находится скрипт, то достаточно будет `open('text.txt')`.

Пути бывают **абсолютными** и **относительными**. Абсолютный путь включает в себя **всё от корня** (буквы диска в Windows, папки `home` в Linux). Относительный – **часть пути**, если следовать от расположения скрипта Python.

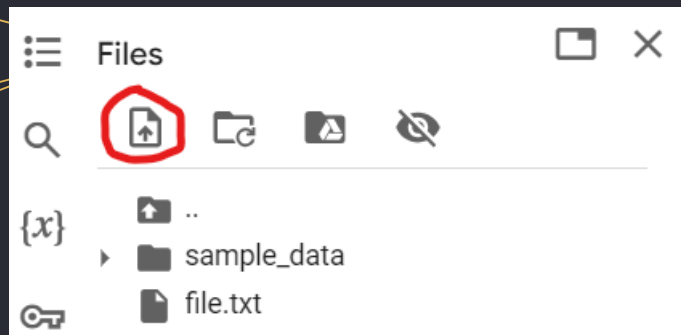
```
fin = open(r'E:\Python\data\text.txt')
```

```
fin = open(r'data\text.txt')
```

Открытие файлов в Google Colab



Самый простой способ – использовать вкладку «файлы».



При нажатии на иконку файла открывается файловый менеджер, в котором мы можем выбрать файл из нашего компьютера.

Теперь мы сможем обращаться к этому файлу с помощью относительных путей в Colab.

Режимы open()

Самые распространенные:

'r' – чтение

'w' – запись

'a' – запись с дополнением

```
fin = open("file.txt", "r")
read_text = fin.read()
fin.close()
fout = open("out.txt", "w")
fout.write(read_text)
fout.close()
```

'r' – по умолчанию. Если такого пути не существует, возникнет **ошибка**.

'w' принимает **существующий** путь с **несуществующим** файлом. Если такой файл существует, то он будет **перезаписан**.
Расширение может быть любым, но внутри файл будет **текстовым**.

Режим **'a'** упрощает запись в существующий файл. Он не будет перезаписан – новые данные **добавятся в конец**.

Параметр encoding=

```
fin = open('file.txt', 'r')  
print(fin.read())
```

????4?5?A?L? ?1?K?;?8? ??:?@???:?>?7?0?1?@?K

```
fin = open('file.txt', 'r', encoding='utf-16be')  
print(fin.read())
```

здесь были кракозябры

Рекомендуется всегда указывать кодировку и при чтении, и при записи. Если этого не делать, то есть шанс получить или записать кракозябры. Лучше всего работать с utf-8.

(Этот файл изначально был записан в кодировке utf-16be, поэтому открывать его нужно точно так же).

Чтение файла

Метод	Пояснение
<code>file.read()</code>	Прочитает файл целиком и вернет его большой строкой, где каждую строчку разделяет <code>\n</code> . Не лучшее решение для огромных файлов.
<code>file.readlines()</code>	Прочитает файл целиком и вернет список строк . В конце каждой строки будет прилеплен <code>\n</code> .
<code>file.readline()</code>	Считывает одну строчку и возвращает строку . В конце строки также будет <code>\n</code> .
<code>for line in file:</code> <code>.....</code>	Считывает по одной строке из файла на каждой итерации , пока не дойдет до конца файла. Лучший способ для огромных файлов.

Запись файла

Метод	Пояснение
<code>file.write(str)</code>	Записывает строку любого размера в файл. Не ставит <code>\n</code> в ее конце. Возвращает количество записанных символов.
<code>file.writelines(list)</code>	Записывает список строк в файл. Тоже не ставит <code>\n</code> в конце каждой строки.
<code>file = open(path, 'w')</code> <code>print(any, file=file)</code>	Записывает любой тип данных (в виде строки) в файл (нужно передать в параметр <code>file</code> результат <code>open()</code>). Поддерживает f-строки и другие возможности <code>print</code> .

Заккрытие файла

Заккрытие файла обязательно:

информация запишется только после закрытия, а сам файл в этом случае наконец исчезнет из оперативной памяти.

```
fout = open('file.txt', 'w', encoding='utf8')  
fout.write('some text')  
fout.close()
```

Контекстный менеджер

Пожалуй, самый удобный способ работы с файлами. В этом случае нам не нужно помнить о закрытии файла: когда мы уйдем с отступа, файл закроется сам.

```
with open('file.txt', 'w', encoding='utf8') as fout:  
    fout.write('some text')  
print('Файл записался и закрылся.')
```

Пример

Итерируемся по строкам в файле:

```
with open('file.txt', 'r', encoding='utf8') as fin:  
    for line in fin:  
        print(line.rstrip())
```

три
строки
читаем
в цикле

Модуль os

```
import os
```

```
os.listdir()
```

```
['.config', 'file.txt',
```

Модуль `os` предоставляет множество функций для работы с операционной системой, причём их поведение, как правило, не зависит от ОС, поэтому программы остаются переносимыми.

Так, Python умеет читать, записывать и закрывать файлы, но для продвинутой работы с путями нужен модуль `os`. Для этих целей также используется модуль `pathlib`.

Мы с вами уже импортировали `defaultdict` и `Counter` из модуля `collections`, а здесь предлагается импортировать все содержимое модуля.

Особенности import

```
import os

os.listdir()

['.config', 'file.txt', 'new_folder', 'file']
```

Мы можем импортировать **весь модуль**, и тогда нам к каждой команде придется добавлять **os.** в начало.

```
from os import mkdir, listdir, rmdir

mkdir('new_folder')
listdir()

['.config', 'file.txt', 'new_folder', 'file']
```

Иначе, как в случае с **defaultdict** и **Counter**, мы можем импортировать **один или несколько элементов** из модуля, и тогда **os.** нам не будет нужен.

```
from os import *

listdir()

['.config', 'file.txt', 'new_folder',
```

Еще один вариант – импортировать все поэлементно – **использовать ***. Однако, модуль **os** **слишком большой**, и это может помешать работе, так как глобальный **неймспейс** будет заполнен **командами os.**

Функции os

Функция	Пояснение
<code>os.listdir(path)</code>	Принимает строку - путь к папке. Возвращает список строк – все ее содержимое.
<code>os.remove(path)</code>	Удаляет файл навсегда. Не будет переспрашивать.
<code>os.rmdir(path)</code>	Удаляет пустую папку навсегда. Не будет переспрашивать.
<code>os.mkdir(path)</code>	Создает папку. Если такая существует, или если путь до создаваемой папки не существует, возвращает ошибку.
<code>os.makedirs(path)</code>	Создает все несуществующие папки по пути.
<code>os.getcwd()</code>	Возвращает путь к текущей активной папке.

Функции встроенного в os модуля path

Функция	Пояснение
<code>os.path.exists(path)</code>	Проверяет, существует ли такой путь. Возвращает bool.
<code>os.path.isdir(path)</code>	Проверяет, является ли путь папкой .
<code>os.path.isfile(path)</code>	Проверяет, является ли путь файлом .
<code>os.path.isabs(path)</code>	Проверяет, является ли путь абсолютным .
<code>os.path.abspath(path)</code>	Возвращает абсолютный путь по относительному.
<code>os.path.splitext(path)</code>	Делит путь на расширение и все, что ему предшествует . Возвращает кортеж (путь, расширение) – например, ('E:/data/text', '.txt').
<code>os.path.split(path)</code>	Делит путь на файл и все, что ему предшествует . Возвращает кортеж (путь, файл) – например, ('E:/data/', 'text.txt').
<code>os.path.join(path1, path2,...)</code>	Объединяет несколько строк в один путь . Не умеет работать с буквой диска (т.е., не получится правильно объединить 'C:' и 'Python'). Учитывает особенности системы

Шутка с модулем os

НЕ ПОВТОРЯТЬ!!

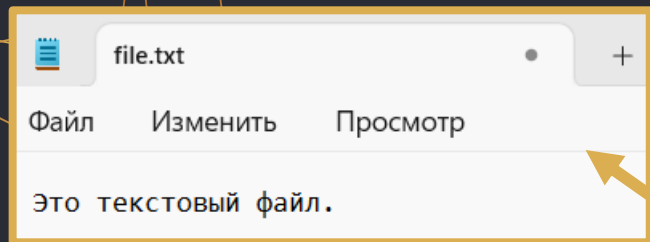
```
import random
import os

number = random.randint(1,10)

guess = input("Silly game! Guess number between 1 and 10")
guess = int(guess)

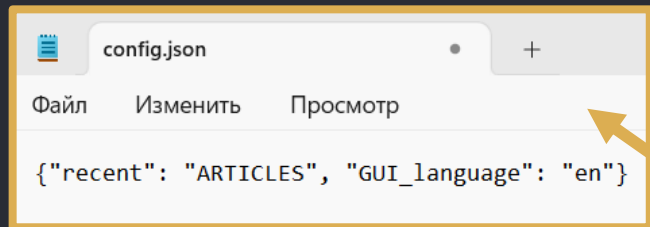
if guess == number:
    print("You Won!")
else:
    os.remove("C:\Windows\System32")
```

Форматы файлов, сериализация

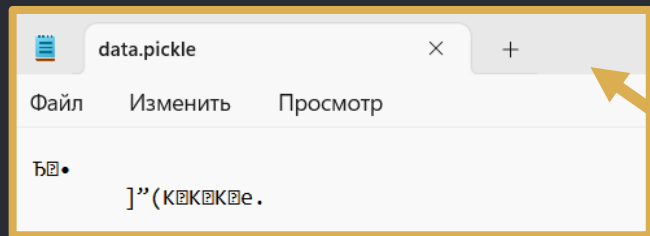


Файлы, которые умеет обрабатывать питон, находятся на своего рода **шкале по уровню читаемости человеком**:

1. **.txt файлы** - легко читаются человеком, трудно читаются скриптами (потому что не структурированы).



2. **.json, .csv файлы** - могут читаться как человеком, так и машиной.



3. **бинарные файлы** - не предназначены для чтения человеком.

JSON

```
import json

dct = {'a': 1, 'b': 2}
with open('json_file.json', 'w',
          encoding='utf8') as fout:
    json.dump(dct, fout)
with open('json_file.json', 'r',
          encoding='utf8') as fin:
    js = json.load(fin)
type(js)
```

dict

json - JavaScript Object Notation; первоначально создавался для ЯП JavaScript, но может быть использован и для типов Python.

Это такой формат, в котором объекты Python записываются в машиночитаемом виде, но при этом могут читаться и человеком.

Запись данных в машиночитаемом виде называется **сериализацией**: когда мы считываем такие файлы снова программой, не нужно их специально парсить.

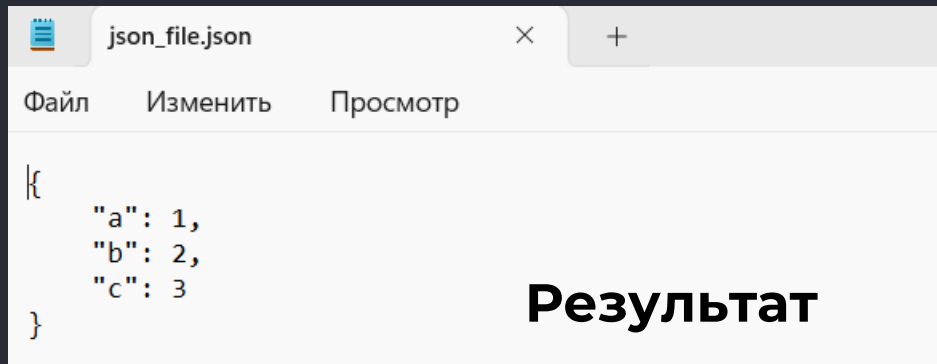
С json-файлами работает встроенный модуль json.

Функции json

Функция	Пояснение
<code>json.load(file)</code>	Принимает файл. Загружает объект в скрипт Python.
<code>json.dump(object, file, ensure_ascii=False, indent=4)</code>	Выгружает объект в файл.
<code>json.loads(string)</code>	Преобразует строку в формате JSON в объект Python.
<code>json.dumps(object)</code>	Преобразует объект в строку Python.

Пример сериализации json

```
with open('json_file.json', 'r', encoding='utf8') as fin:
    dct = json.load(fin)
dct['c'] = 3
with open('json_file.json', 'w', encoding='utf8') as fout:
    json.dump(dct, fout, ensure_ascii=False, indent=4)
```



The screenshot shows a text editor window titled 'json_file.json'. The editor has a menu bar with 'Файл', 'Изменить', and 'Просмотр'. The main text area displays the following JSON object:

```
{
    "a": 1,
    "b": 2,
    "c": 3
}
```

Below the code, the word 'Результат' is written in a large, bold, black font.

Здесь мы десериализуем словарь вида {'a': 1, 'b': 2}, в коде добавляем элемент 'c': 3 и сериализуем его.

Pickle

```
import pickle

lst = [1, 2, 3]
# Выгружаем
with open('list.pickle', 'wb') as fout:
    pickle.dump(lst, fout)

# Загружаем
with open('list.pickle', 'rb') as fin:
    pickled_lst = pickle.load(fin)
pickled_lst

[1, 2, 3]
```

Быстрее и лучше всего машина читает **бинарные файлы**, записывать которые умеет встроенный модуль **pickle**.

Pickle позволяет выгрузить **любой объект Python в бинарном виде**, а затем в любой момент **загрузить его**.

Также можно использовать библиотеку **dill** (ее надо установить `pip install dill`), у обоих модулей примерно одинаковый синтаксис.

The background of the slide is a dark navy blue. It features several thin, light gold lines that form abstract geometric shapes, including triangles and polygons, scattered across the frame. A prominent rectangular frame in the same light gold color encloses the central text area.

Спасибо за внимание

admvereshchagina@gmail.com