



Стандартные функции Input-Output



Файлы

Файлы - это просто массив байт на жёстком диске. В файлах мы можем хранить абсолютно любую информацию: текст, аудио, видео, изображения, исполняемый код и т.д. Логически принято делить файлы на:

1. Текстовые
2. Бинарные (двоичные)



Типы текстовых файлов

- .txt
- .csv
- .xml
- .json
- .yaml



ТХТ

ТХТ — это формат файлов, который содержит текст, упорядоченный по строкам. Текстовые файлы отличаются от двоичных файлов, содержащих данные, не предназначенные для интерпретирования в качестве текста (закодированный звук или изображение).



Файл настроек Raspberry Pi config.txt

```
# Set stdv mode to PAL (as used in Europe)
sdtv_mode=2
# Force the monitor to HDMI mode so that sound will be sent
over HDMI cable
hdmi_drive=2
# Set monitor mode to DMT
hdmi_group=2
# Set monitor resolution to 1024x768 XGA 60Hz
(HDMI_DMT_XGA_60)
hdmi_mode=16
# Make display smaller to stop text spilling off the screen
overscan_left=20
overscan_right=12
overscan_top=10
overscan_bottom=10
```



CSV

CSV (comma-separated value) - это формат представления табличных данных (например, это могут быть данные из таблицы или данные из БД).

В этом формате каждая строка файла - это строка таблицы. Несмотря на название формата, разделителем может быть не только запятая.

И хотя у форматов с другим разделителем может быть и собственное название, например, TSV (tab separated values), тем не менее, под форматом CSV понимают, как правило, любые разделители.



Пример файла cvs

1.04.2022,1 апреля,13.00,"Открытие выставки"

1.04.2022,1 апреля,18.00,"Показ музыкальной комедии"

2.04.2022,2 апреля,16.00,"Концертная программа «Мелодия любви»"

5.04.2022,5 апреля,18.00,"Показ боевика"



.xml

XML — текстовый формат, предназначенный для хранения структурированных данных . XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов как программами, так и человеком, с акцентом на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

Файл workspace.xml в папки .idea

```
▼<component name="TaskManager">
  ▼<task active="true" id="Default" summary="Default task">
    <changelist id="809dbb00-5478-42ba-ae09-75e83407f4b8" name="Ch
    <created>1649147023499</created>
    <option name="number" value="Default"/>
    <option name="presentableId" value="Default"/>
    <updated>1649147023499</updated>
  </task>
  ▼<task id="LOCAL-00001" summary="Добавилена диаграмма Mermaid">
    <created>1649153680257</created>
    <option name="number" value="00001"/>
    <option name="presentableId" value="LOCAL-00001"/>
    <option name="project" value="LOCAL"/>
    <updated>1649153680257</updated>
  </task>
  ▼<task id="LOCAL-00002" summary="Добавилена диаграмма Mermaid">
    <created>1649153785124</created>
    <option name="number" value="00002"/>
    <option name="presentableId" value="LOCAL-00002"/>
    <option name="project" value="LOCAL"/>
    <updated>1649153785124</updated>
  </task>
  ▼<task id="LOCAL-00003" summary="Hotfix">
    <created>1649154138050</created>
    <option name="number" value="00003"/>
    <option name="presentableId" value="LOCAL-00003"/>
    <option name="project" value="LOCAL"/>
    <updated>1649154138050</updated>
  </task>
  ▼<task id="LOCAL-00004" summary="Hotfix">
    <created>1649154793842</created>
    <option name="number" value="00004"/>
```



.json

JSON (JavaScript Object Notation) - простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript, определенного в стандарте ECMA-262 3rd Edition - December 1999. ... Эти свойства делают JSON идеальным языком обмена данными. JSON основан на двух структурах данных: Коллекция пар ключ/значение.

Файл package.json для nrm

```
{
  "name": "cdpo",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "nuxt",
    "build": "nuxt build",
    "start": "nuxt start",
    "generate": "nuxt generate"
  },
  "dependencies": {
    "amqplib": "^0.8.0",
    "core-js": "^3.19.3",
    "nuxt": "^2.15.8",
    "vue": "^2.6.14",
    "vue-server-renderer": "^2.6.14",
    "vue-template-compiler": "^2.6.14",
    "vuetify": "^2.6.1",
    "webpack": "^4.46.0"
  },
  "devDependencies": {
    "@nuxtjs/moment": "^1.6.1",
    "@nuxtjs/vuetify": "^1.12.3"
  }
}
```



.yaml

YAML — это формат файла, обычно используемый для сериализации данных. Существует множество проектов, использующих файлы YAML для настройки, таких как Docker-compose, pre-commit, TravisCI, AWS Cloudformation, ESLint, Kubernetes, Ansible и многие другие.



Конфигурационный файл swagger

```
1. definitions:
2.   CatalogItem:
3.     type: object
4.     properties:
5.       id:
6.         type: integer
7.         example: 38
8.       title:
9.         type: string
10.        example: T-shirt
11.     required:
12.       - id
13.       - title
```



Открытие файла

Работать с файлами можно тремя способами:

1. Читать из файла
2. Записывать в файл
3. Дозаписывать в файл



Открытие файла

Прежде, чем работать с файлом, его надо открыть. Для этой задачи есть встроенная функция `open`:

```
f = open("text.txt", encoding="utf-8")
```

Результатом работы функция `open` возвращает специальный объект, который позволяет работать с файлом (файловый дескриптор)

Можно ли указать `"utf-8"` без **`encoding=`** ?

Синтаксис функции open()

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

Параметры:

file – абсолютное или относительное значение пути к файлу или файловый дескриптор открываемого файла.

mode – необязательно, строка, которая указывает режим, в котором открывается файл. По умолчанию 'r'.

buffering – необязательно, целое число, используемое для установки политики буферизации.


encoding – необязательно, кодировка, используемая для декодирования или кодирования файла.

errors – необязательно, строка, которая указывает, как должны обрабатываться ошибки кодирования и декодирования. Не используется в бинарном режиме

newline – необязательно, режим перевода строк. Варианты: None, '\n', '\r' и '\r\n'. Следует использовать только для текстовых файлов.

closefd – необязательно, bool, флаг закрытия файлового дескриптора.

opener – необязательно, пользовательский объект, возвращающий открытый дескриптор файла.



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, *mode*, в котором мы будем открывать файл. Режим обычно состоит из двух букв, первой является тип файла – текстовый или бинарный, в котором мы хотим открыть файл, а второй указывает, что именно мы хотим сделать с файлом.

Третий аргумент – кодировка файла

Первая буква режима:

"b" – открытие в двоичном режиме.

"t" – открытие в текстовом режиме (является значением по умолчанию).

Второй буква режима:

"r" – открытие на чтение (является значением по умолчанию).

"w" – открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.

"x" – эксклюзивное создание (открытие на запись), бросается исключение `FileExistsError`, если файл уже существует.

"a" – открытие на дозапись, информация добавляется в конец файла.

"+" – открытие на чтение и запись



Примеры

Режим "w" открывает файл только для записи. Перезаписывает файл, если файл существует. Если файл не существует, создает новый файл для записи.

```
f = open("text.txt", mode="w" encoding="utf-8")
```

Открывает файл в бинарном режиме для записи и чтения. Перезаписывает существующий файл, если файл существует. Если файл не существует, создается новый файл для чтения и записи.

```
f = open("music.mp3", mode="wb+")
```

По всем режимам см. [документацию open\(\)](#)



Заккрыть файл

После того как вы сделали всю необходимую работу с файлом - его следует закрыть.

```
f = open("text.txt", encoding="utf-8")  
# какие-то действия  
f.close()
```

Кто ограничивает максимальное количество открытых файловых дескрипторов ?

```
ulimit -Hn
```



Чтение файла

Теперь мы хотим прочесть из него информацию. Для этого есть несколько способов, но большого интереса заслуживают лишь два из них. Первый - метод **read**, читающий весь файл целиком, если был вызван без аргументов, и *n* символов, если был вызван с аргументом (целым числом *n*).

```
f = open("text.txt", "rt")
```

```
print(f.read(5))  
print(f.read(5))  
print(f.read(4))  
print(f.read())
```

```
f.close()
```



Чтение файла

Теперь мы хотим прочитать из него информацию. Для этого есть несколько способов, но большого интереса заслуживают лишь два из них. Первый - метод **read**, читающий весь файл целиком, если был вызван без аргументов, и *n* символов, если был вызван с аргументом (целым числом *n*).

```
f = open("text.txt", "rt")
```

```
print(f.read(5))  
print(f.read(5))  
print(f.read(4))  
print(f.read())
```

```
f.close()
```



Функция `readlines()`

Файлы можно читать не только целиком или посимвольно, но и построчно. Для этого у объекта файла есть метод `readlines` который возвращает список из строк файла.

```
f = open("text.txt", "rt")  
print(f.readlines())  
f.close()
```

Обратите внимания, что каждая строка в списке имеет в конце символ `\n`.



Функция `readline()`

Функция ``readlines`` загружает все строки целиком и хранит их в оперативной памяти, что может быть очень накладно, если файл занимает много места на жёстком диске. Можно читать файл построчно с помощью функции ``readline``

```
f = open("text.txt", "rt")  
print(f.readline())  
print(f.readline())  
f.close()
```

Также обратите внимание, что возвращённые строки имеют в конце символ ``\\n``.



Итерирование файла

Ещё один способ прочитать файл построчно – использовать файл как итератор. Такой вариант считается самым оптимизированным

```
f = open("text.txt")
```

```
for line in f:
```

```
    print(line)
```

```
f.close()
```




Запись

(*) rec

Теперь рассмотрим запись в файл. Для того чтобы можно было записывать информацию в файл, нужно открыть файл в режиме записи. Для записи в файл используется функция `write`. При открытии файла на запись из него полностью удаляется предыдущая информация.

```
f = open("text.txt", "wt")  
f.write("New string")  
f.write("Another string")  
f.close()
```

Если вы откроете файл в текстовом редакторе, то увидите, что строки `"New string"` и `"Another string"` склеились. Так произошло, потому что между ними нет символа перевода строки.



Также в файлах, открытых на запись, есть метод `writelines`, который позволяет записать несколько строк в файл

```
f = open("text.txt", "wt")
lines = [
    "New string\n",
    "Another string\n",
]
f.writelines(lines)
f.close()
```



Дозапись

Если нужно записать в конец файла какую-то информацию, то можно сделать это открыв файл в режиме дозаписи. Все методы, доступные в режиме записи также доступны в режиме дозаписи.

```
f = open("text.txt", "at")
```

```
f.write("First string\n")
```

```
lines = [  
    "Second string\n",  
    "Third string\n",  
]
```

```
f.writelines(lines)
```

```
f.close()
```



Запись с возможностью чтения

Иногда нужно открыть файл с возможностью и записи, и чтения. В Python есть два режима:

- * Запись с возможностью чтения ("w+")
- * Чтение с возможностью записи ("r+")

На первый взгляд кажется, что они ничем не отличаются, но это не так.

При открытии файла на запись с возможностью чтения из файла полностью удаляется вся информация. Вы можете записывать и читать из файла одновременно.



Пример.

(*) rec

```
f = open("text.txt", "w+t")
```

```
print(f.read())
```

```
f.write("Hello\n")
```

```
print(f.read())
```

```
f.close()
```



Чтение с возможностью записи

При открытии файла на чтение с возможностью записи файл не перезаписывается.

```
f = open("text.txt", "r+t")  
  
print(f.read(1))  
f.write("A")  
f.read()  
  
f.close()
```

При записи символы в файле затирают символы, идущие следом, как если бы вы в текстовом редакторе перевели указатель в середину текста, нажали insert и начали бы печатать.



PEP8

В PEP8 описано, что в конце файла с кодом всегда нужно оставлять пустую строку. Это правило кажется надуманным, но сейчас мы знаем, что в любой файл, в котором последним символом стоит перевод строки можно программно дозаписать любую строку и она не склеится с последней строкой в файле.



Управляющие символы

<code>\n</code>	(newline) перевод каретки на следующую строку
<code>\r</code>	(return) перевод каретки на в начало текущей строки
<code>\t</code>	(tab) табуляция (отступ, красная строка)
<code>\b</code>	(backspace) перевод каретки на один символ назад



Указатель позиции

При чтении файла функция ``read`` читает символы друг за другом, а при записи в файл все строки (строки байт) записываются последовательно друг за другом. Это поведение объясняется тем, что python хранит специальный указатель, позиция этого указателя говорит, с какого места читать из файла или писать в файл.

Независимо от того в каком режиме открыт файл у каждого объекта файла есть методы ``tell`` и ``seek``. Метод ``tell`` возвращает целое число – позицию, где сейчас находится указатель. Метод ``seek`` принимает целое число и переносит указатель в указанную позицию. Например, передвинуть указатель на две позиции вперёд можно следующим образом

```
position = f.tell()  
f.seek(position + 2)
```



file.seek(offset[, whence])

Параметры:

`file` - объект файла

`offset` - `int` байтов, смещение указателя чтения/записи файла.

`whence` - `int`, абсолютное позиционирование указателя.

Возвращаемое значение:

целое число `int`, новая позиция указателя.

Описание:

Метод файла `file.seek()` устанавливает текущую позицию в байтах `offset` для указателя чтения/записи в файле `file`.

Аргумент `whence` является необязательным и по умолчанию равен 0.
Может принимать другие значения:

0 - означает, что нужно сместить указатель на `offset` относительно начала файла.

1 - означает, что нужно сместить указатель на `offset` относительно текущей позиции.

2 - означает, что нужно сместить указатель на `offset` относительно конца файла.



Пример 1

```
# Начать чтение с 3 символа строки  
f = open('testFile.txt', 'r')  
f.seek(3)  
print(f.read())
```



Пример 2

```
>>> text = b'This is 1st line\nThis is 2nd line\nThis is 3rd line\n'
>>> fp = open('foo.txt', 'bw+')
>>> fp.write(text)
# 51

>>> fp.seek(20, 0)
# 20
>>> fp.read(10)
# b's is 2nd l'

>>> fp.seek(10, 1)
# 40
>>> fp.read(10)
# b's 3rd line'

>>> fp.seek(-11, 2)
# 40
>>> fp.read(10)
# b's 3rd line'

>>> fp.close()
```



Модуль OS

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

`os.chdir(path)` – смена текущей директории.

`os.listdir(path=".")` – список файлов и директорий в папке.

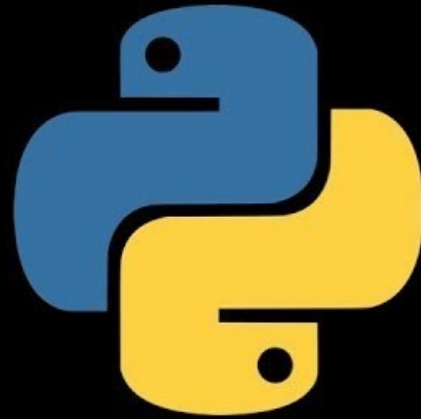
`os.mkdir(path, mode=0o777, *, dir_fd=None)` – создаёт директорию. `OSError`, если директория существует.

`os.rmdir(path, *, dir_fd=None)` – удаляет пустую директорию.

Полный перечень функций:

<https://docs.python.org/3/library/os.html>

<https://all-python.ru/osnovy/os.html>



PYTHON PROGRAMMING