

< Teach
Me
Skills >




09

Работа с текстом, сериализация и файловая система



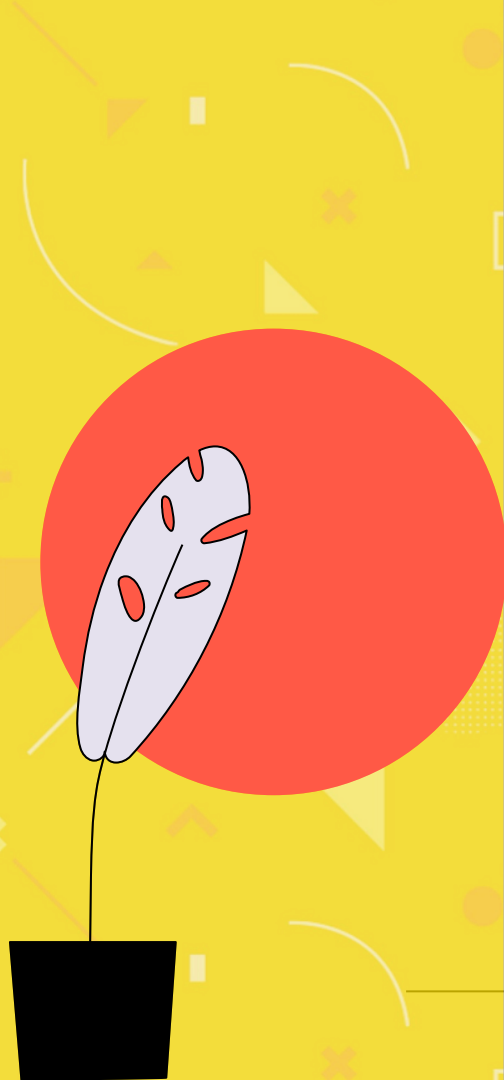
План занятия

- Кодировки
 - Взаимодействие с файловой системой: модуль `os`
 - Работа с файлами
 - Сериализация и десериализация
 - Работа с внешними данными: JSON, CSV
 - Регулярные выражения
- 

Кодировки



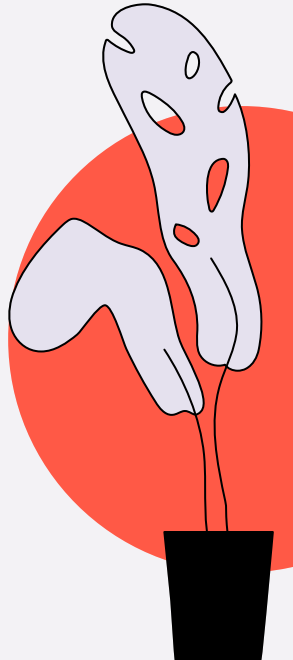
Кодировка – это правила перевода одного набора символов в другой. В отношении компьютерных программ речь идёт о правилах кодирования последовательностей из нулей и единиц в текст, число или что-то другое.



Кодировки

На самом деле компьютер оперирует исключительно числами 0 и 1. Обычные привычные нам числа в десятичной системе счисления закодированы с помощью двоичных чисел:

- 0 – 0
- 1 – 1
- 2 – 10
- 3 – 11
- 4 – 100
- 5 – 101
- 6 – 110
- 7 – 111
- и так далее




Кодировки

Но как быть с текстом? На самом деле компьютер ничего не знает о буквах, знаках пунктуации и других текстовых символах.

Можно взять английский алфавит и поставить каждой букве в соответствие число:

- a – 1
- b – 2
- c – 3
- d – 4
- ...

В этом и есть смысл кодировок.



Кодировки

Во время работы, программы используют кодировки для преобразования чисел в символы и наоборот. Причём сама программа не имеет представления о смысле этих символов.

Таблицы, в которых сопоставляются символы и числа называются **таблицами кодировок**. Кроме букв алфавита, в таблицу кодировки входят знаки препинания и другие полезные символы.

Разные кодировки содержат разное количество символов. Изначально небольших таблиц вроде ASCII было вполне достаточно для решения большинства задач.



Кодировки

В ASCII входят только латинские буквы в верхнем и нижнем регистрах, несколько простых символов вроде % и ?, и специальные управляющие символы (например, перевод строки).

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
SPC	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
Ђ	Ѓ	Ѕ	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї
ђ	ѓ	ѕ	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї
Ў	ў	Ј	ј	Љ	љ	Ћ	ћ	Ќ	ќ	Ў	ў	Ј	ј	Љ	љ
°	±	І	і	Г	г	μ	μ	·	ё	№	е	»	Ј	ѕ	ї
А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я



0 1 2 3 4 5 6 7 8 9 A B C D E F

0 1 2 3 4 5 6 7 8 9 A B C D E F

ASCII

Кодировки

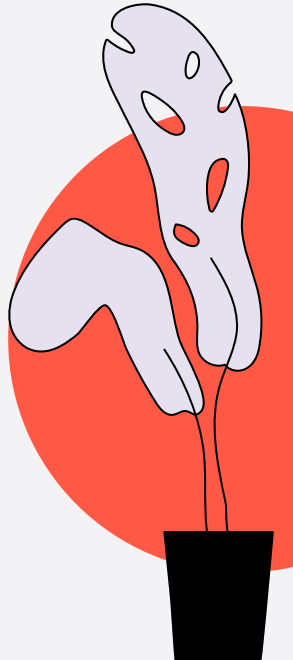
Так как первые ЭВМ были малоёмкими, для представления в их памяти всего набора требуемых знаком хватало 7 бит (128 символов).

```
ASCII

1 import string
2
3 print(string.ascii_letters)
4 print(string.digits)
5 print(string.punctuation)
6
7 # Output:
8 # abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
9 # 0123456789
10 # !"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
```


Кодировки

С течением времени общество становилось всё более компьютеризированным, 128 символов стало не хватать. Появилось большое количество различных кодировок – немецкая, кириллическая и т.п. Такая ситуация повлекла за собой множество проблем – уже в то время люди из разных стран, обмениваясь электронными письмами, могли увидеть не знакомые символы, а набор непонятных закорючек, что было связано с отсутствием корректного перевода символов из одной кодировки в другую. Потребовалось указание кодировок в заголовках документов.

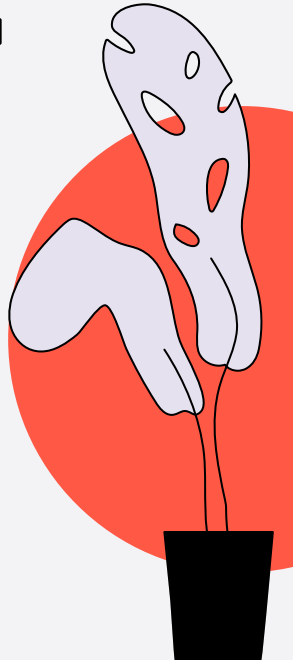




Кодировки

Всего в Unicode на сегодняшний день есть место более, чем для 100 000 символов.

- **UTF-8** – кодировка символов Unicode, использующая от 1 до 4 байт. Наиболее часто используемые символы занимают 1 байт, чем менее популярен символ, тем больше байт он занимает.
- **UTF-16** – кодировка, использующая 2 или 4 байта. Считается удобной для пользователей из азиатских стран с иероглифическим письмом.
- **UTF-32** – представляет все символы при помощи 4 байт. Редко используется, так как потребляет много памяти, но при наличии необходимых мощностей работает весьма быстро.

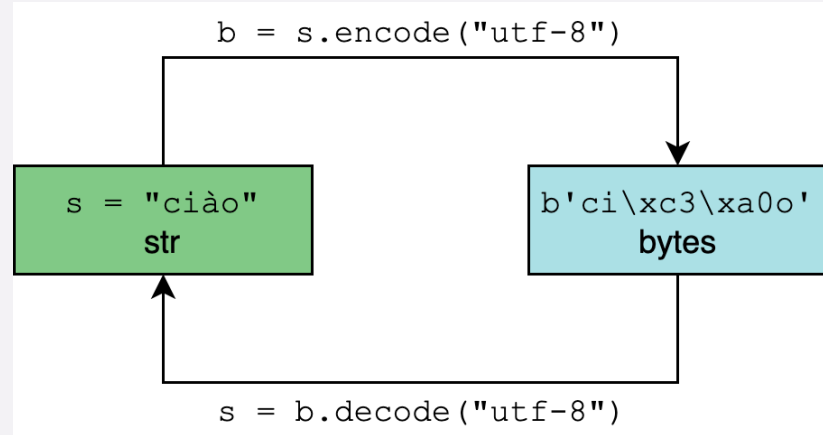


Кодировки

Тип данных **str** в Python рассчитан на представление текста и может содержать любые символы юникода.

Тип **bytes**, напротив, представляет последовательность байт без указания на кодировку.

Кодирование и декодирование – это переход данных из одной формы в другую.

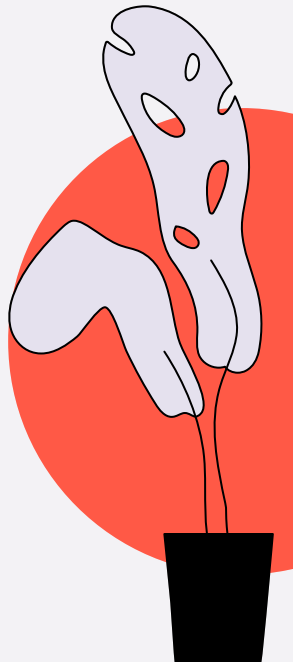


Кодировки

Для использования кодировок в Python есть методы `decode()` и `encode()`. По умолчанию они работают с кодировкой `utf-8`, однако для наглядности её можно указывать явно.

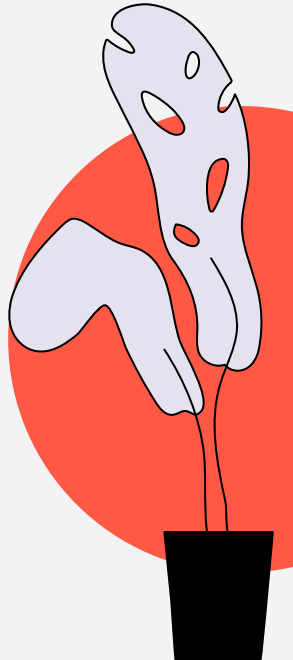
```
encode - decode

1 some_str = 'привет'
2 s_bytes = some_str.encode('utf-8')
3 print(s_bytes)
4 # Output:
5 # b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'
6
7 some_str_decoded = s_bytes.decode('utf-8')
8 print(some_str_decoded)
9 # Output: привет
```



Кодировки

decode(), encode()	кодируют/декодируют строку в нужной кодировке
ascii()	представляет строку в формате ASCII. Все не-ascii символы будут экранированы
chr()	позволяет увидеть юникод символ, соответствующий введённому числовому значению
ord()	позволяет увидеть соответствующее введённому символу число



Кодировки

При работе с кодировками важно помнить:

- Если вы кодируете строку в байты кодировкой utf-8, то и декодировать её из байт нужно той же кодировкой. Некоторые кодировки совместимы, но в большинстве случаев применение разных кодировок может привести к потере данных
- Декодирование байтовых данных в строки лучше производить сразу после их получения, а кодирование в байты – только перед отправкой. В коде лучше работать с привычными типами данных, а не с байтами
- По возможности используйте Unicode (например, utf-8). Utf-8 по умолчанию используется практически везде, это снижает риск ошибок

Кодировки

encoding features

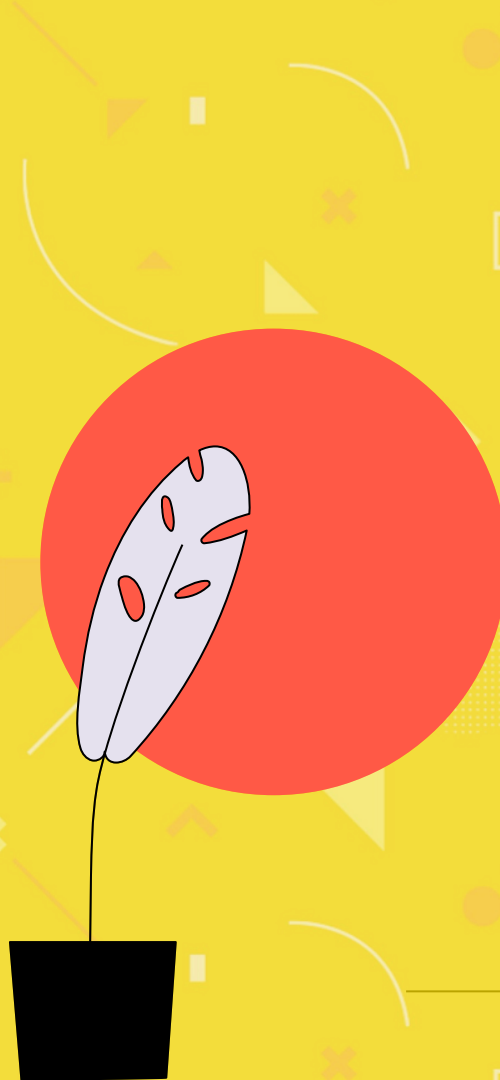
```
1 print(ascii('cat'))
2 # Output: 'cat'
3
4 print(ascii('кошка'))
5 # Output: '\u043a\u043e\u0448\u043a\u0430'
6
7 print(bytes('Python', 'utf16'))
8 # Output: b'\xff\xfeP\x00y\x00t\x00h\x00o\x00n\x00'
9
10 print(bytes('Python', 'utf32'))
11 # Output:
12 # b'\xff\xfe\x00\x00P\x00\x00\x00y\x00\x00\x00t\x00\x00\x00h\x00\x00\x00o\x00\x00\x00n\x00\x00\x00'
13
14 print(chr(63))
15 # Output: ?
16
17 print(ord('?'))
18 # Output: 63
```


Взаимодействие с файловой системой



Обработка файлов с помощью модуля **os** включает создание, переименование, удаление файлов и папок, а также получение списка всех файлов и каталогов и многое другое.

Модуль **os** используется не только для работы с файлами, он включает в себя массу методов и инструментов для других операций: обработки переменных среды, управления системными процессами, а также аргументы командной строки и расширенные атрибуты файлов Linux.



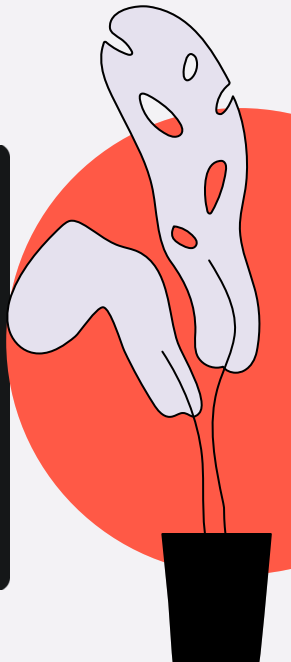
Взаимодействие с файловой системой

Самая простая и вместе с тем одна из самых важных команд для Python-разработчика – показать текущий каталог. Она нужна, потому что разработчики чаще всего имеют дело с относительными путями, но в некоторых случаях важно знать, где мы находимся.



Get Current Directory

```
1 import os
2
3 # Get Current Working Directory
4 print("Current directory:", os.getcwd())
```



Взаимодействие с файловой системой

Прежде чем задействовать команду по созданию файла или каталога, стоит убедиться, что аналогичных элементов ещё нет. Это поможет избежать ряда ошибок, включая перезапись существующих элементов с данными.



Check If Exists

```
1 import os
2
3 # Check if file or directory exists
4 os.path.exists('sample_data')
```

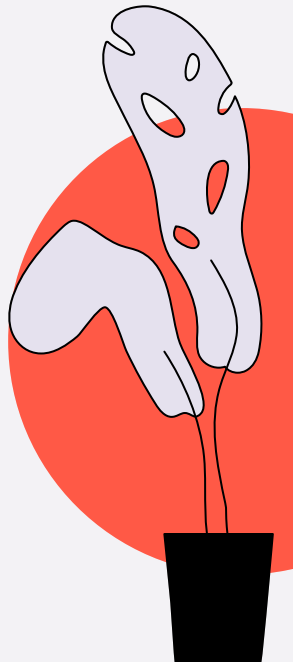
Взаимодействие с файловой системой

Если вы хотите, чтобы ваше приложение было кроссплатформенным, не стоит использовать “/” как разделитель пути, некоторые старые версии *Windows* распознают только “\” в качестве разделителя. Python решает эту проблему – **`os.path.join()`**



Path Join

```
1 import os
2
3 # Create path data/README.md
4 path = os.path.join('data', 'README.md')
5
6 # And check if it exists
7 os.path.exists(path)
```



Взаимодействие с файловой системой

Создать новую директорию можно командой **mkdir**. Если мы попытаемся создать директорию, которая уже существует, то получим исключение, поэтому лучше всегда проверять наличие одноимённого каталога перед созданием

```
Make Directory

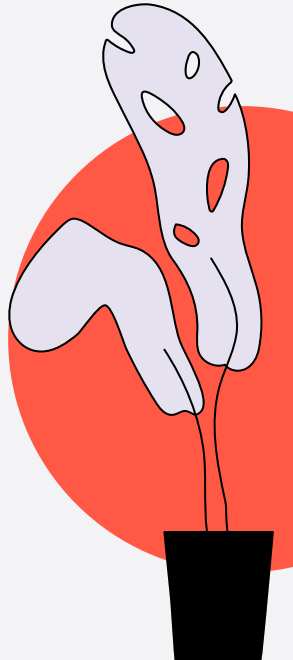
1 import os
2
3
4 print(f"test_dir exists {os.path.exists('test_dir')}")
5 # False, there is no test_dir yet
6
7 # Make directory with name test_dir
8 os.mkdir('test_dir')
9
10 print(f"test_dir exists {os.path.exists('test_dir')}")
11 # True, now the directory exists
```

Взаимодействие с файловой системой

Иногда нам нужно создать подкаталоги с уровнем вложенности 2 и более. Если мы всё ещё используем **os.mkdir()**, нам нужно будет вызвать её несколько раз. Для таких ситуаций можно использовать **os.makedirs()**, она позволит создать все необходимые промежуточные каталоги.

```
1 import os
2
3
4 os.makedirs(os.path.join('test_dir', 'level_1', 'level_2', 'level_3'))
```

Make Directories



Взаимодействие с файловой системой

Ещё одна полезная команда – **os.listdir()**, она показывает всё содержимое каталога.

С помощью модуля **os** достаточно просто переименовать файл:

```
List Dirs

1 import os
2
3
4 os.listdir('sample_data')
```

```
Rename File

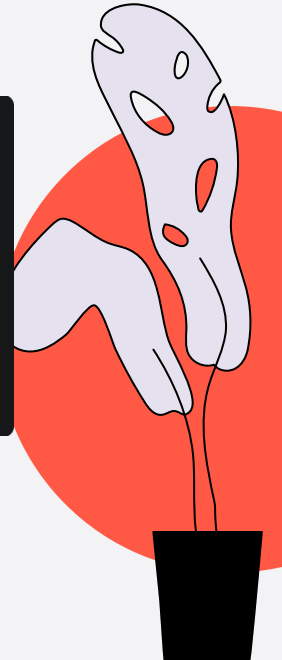
1 import os
2
3
4 os.rename("text.txt", "renamed-text.txt")
```

Взаимодействие с файловой системой

Функцию `os.replace()` можно использовать для перемещения файлов или каталогов. Стоит обратить внимание, что если в папке *folder* уже есть файл с таким именем, он будет полностью перезаписан.

```
Replace File

1 import os
2
3
4 os.replace("renamed-text.txt", "folder/renamed-text.txt")
```



Взаимодействие с файловой системой

Если вывод `os.listdir()` оказывается недостаточным, и нам нужно узнать состав всех вложенных каталогов, можно использовать `os.walk()` – генератор дерева каталогов. "." здесь – корень дерева.

Recursion File Searching

```
1 import os
2
3
4 for dirpath, dirnames, filenames in os.walk("."):
5
6     for dirname in dirnames:
7         print("Каталог:", os.path.join(dirpath, dirname))
8
9     for filename in filenames:
10         print("Файл:", os.path.join(dirpath, filename))
11
12 # Каталог: .\folder
13 # Каталог: .\handling-files
14 # Каталог: .\nested1
15 # Файл: .\text.txt
16 # Файл: .\handling-files\listing_files.py
17 # Файл: .\handling-files\README.md
18 # Каталог: .\nested1\nested2
19 # Каталог: .\nested1\nested2\nested3
```

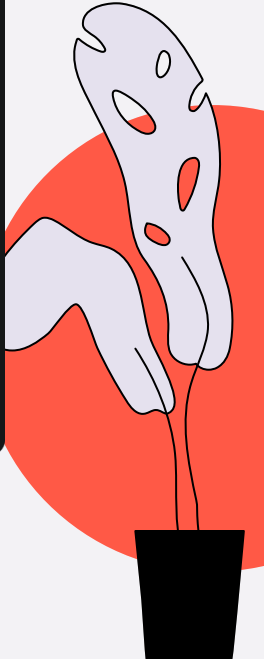
Взаимодействие с файловой системой

Удалить существующий файл (не каталог) можно командой **`os.remove()`**



Delete File

```
1 import os
2
3
4 os.remove("folder/renamed-text.txt")
```



Взаимодействие с файловой системой

С помощью функции `os.rmdir()` можно удалить указанную папку. Для удаления каталогов рекурсивно стоит использовать `os.removedirs()`



Delete Directory

```
1 import os
2
3
4 os.rmdir("folder")
```

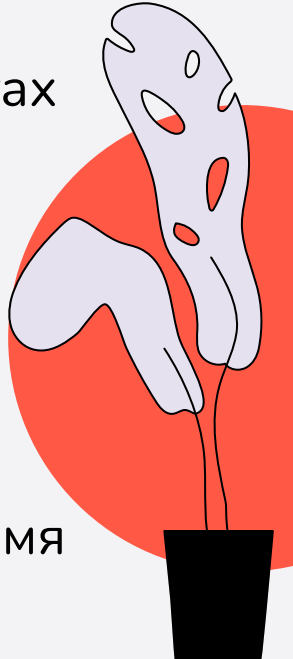
Взаимодействие с файловой системой

Для получения информации о файле используется функция **os.stat()**, которая вернёт кортеж с несколькими метриками.

- **st_size** – размер файла в байтах
- **st_atime** – время последнего доступа
- **st_mtime** – время последнего изменения
- **st_ctime** – в Linux это время последнего изменения метаданных, в Windows – время создания файла

```
Delete Directory

1 import os
2
3
4 open("text.txt", "w").write("Some data")
5
6 print(os.stat("text.txt"))
7 # os.stat_result(
8 # st_mode=33206,
9 # st_ino=14355223812608232,
10 # st_dev=1558443184,
11 # st_nlink=1,
12 # st_uid=0,
13 # st_gid=0,
14 # st_size=19,
15 # st_atime=1575967618,
16 # st_mtime=1575967618,
17 # st_ctime=1575966941)
```

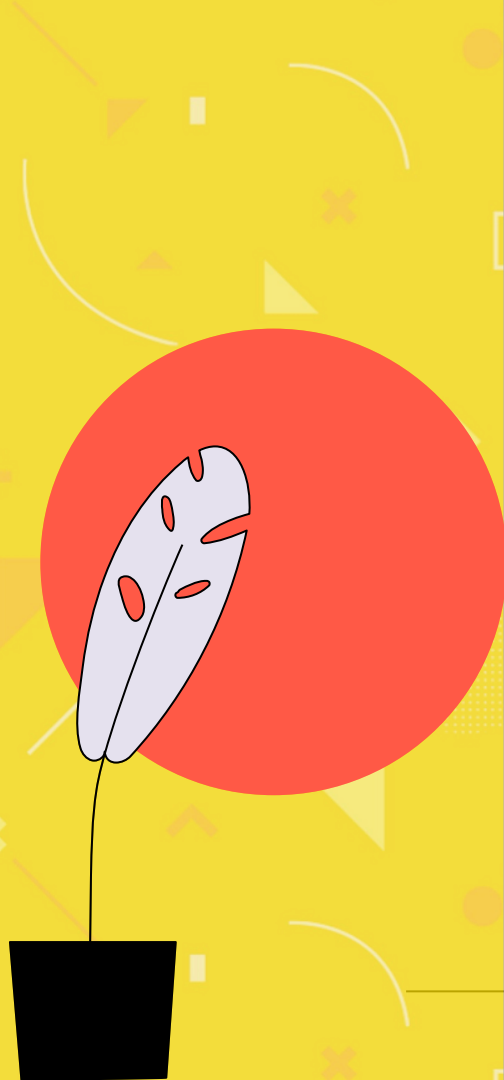


Работа с файлами



Файлы используются программами для долговременного хранения информации, необходимой непосредственно для работы программы (например, настройки), а также для хранения информации, полученной во время выполнения программы (например, результаты вычислений).

Подавляющее большинство программ в том или ином виде использует файлы, сохраняя результаты работы между сеансами запуска.



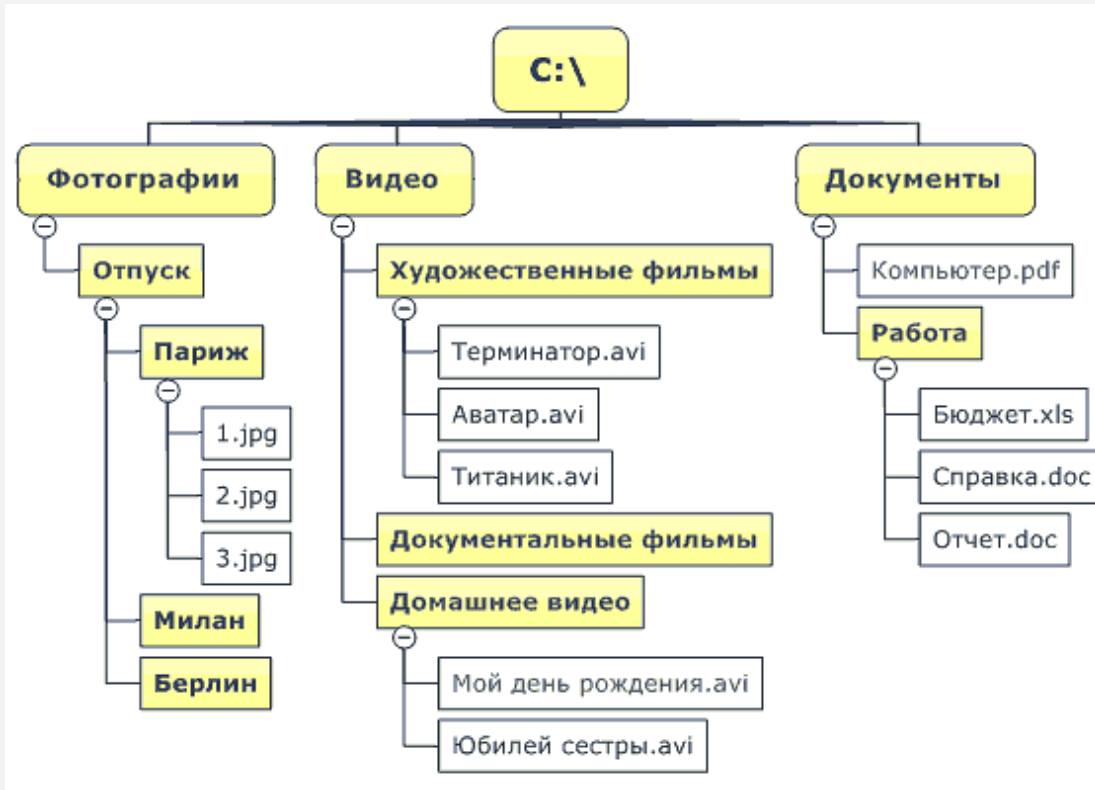
Работа с файлами

Файл – именованная область данных на носителе информации.

Файлы хранятся в **файловой системе** – каталоге, определяющем способы организации, хранения и именования данных, а также задающем ограничения на формат и доступ к данным.

На сегодняшний день наиболее популярными являются древовидные каталоги (директории, папки) – файлы, содержащие информацию о входящих в них файлах.

Работа с файлами



Работа с файлами

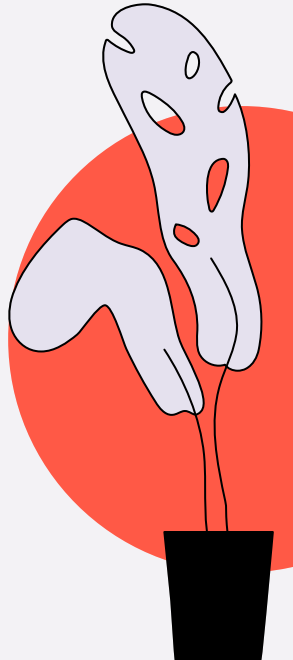
Файловая система связывает носитель информации с программным интерфейсом для доступа к файлам. Когда прикладная программа обращается к файлу, она не имеет никакого представления о том, каким образом представлена информация в конкретном файле, а также не имеет информации, на каком носителе вообще записан файл. Всё, что знает программа, - имя файла, его размер и атрибуты (атрибуты получаются от драйвера файловой системы).

Именно файловая система устанавливает, как и где будет записан файл на физическом носителе.

Работа с файлами

Файл может обладать различным набором свойств в зависимости от типа используемой файловой системы. В основном файл имеет следующие свойства:

- Имя и расширение
- Дата и время создания (опционально - дата и время последнего доступа к файлу)
- Владелец
- Атрибуты (скрытый/системный/др.) и права доступа



Работа с файлами

Для того, чтобы найти файл в файловой системе, необходимо знать к нему **путь** – узлы дерева файловой системы, которые нужно пройти, чтобы до него добраться.

Путь может быть:

- Абсолютным – указывающим на одно и то же место файловой системы вне зависимости от текущей рабочей директории или других обстоятельств
- Относительным – путём по отношению к текущему рабочему каталогу

Работа с файлами

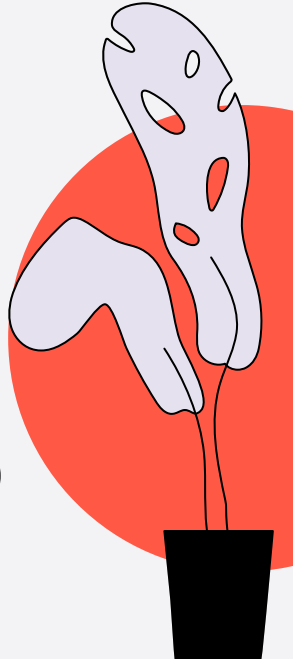
Примеры путей:

ОС Windows:

- Абсолютный: `C:\user\python\example1.py`
- Относительные:
 - `example1.py` (текущий каталог `C:\user\python\`)
 - `python\example1.py` (текущий каталог `C:\user\`)

ОС UNIX:

- Абсолютный: `/home/user/python/example1.py`
- Относительные:
 - `example1.py` (текущий каталог `/home/user/python/`)
 - `user/python/example1.py` (текущий каталог `/home/`)



Работа с файлами

При открытии файла, как правило, указывается имя и права доступа, после чего ОС возвращает специальный **дескриптор файла** – идентификатор, однозначно определяющий, с каким файлом дальше будут выполняться операции.

После открытия файла становится доступен **файловый указатель** – число, определяющее текущую позицию относительно начала файла.

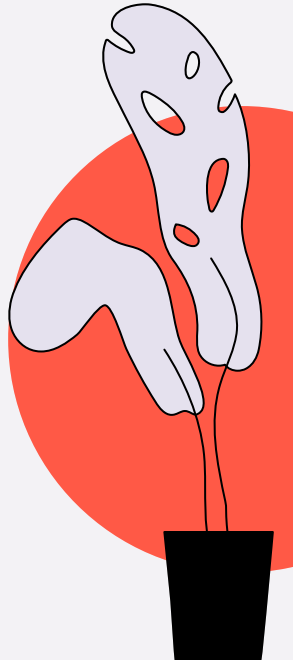


Работа с файлами

При работе с файлами нужно соблюдать некоторую последовательность операций:

- Открыть файл
- Прочитать содержимое файла или записать в него новые данные
- Закрыть файл

Рассмотрим подробно каждую из операций.

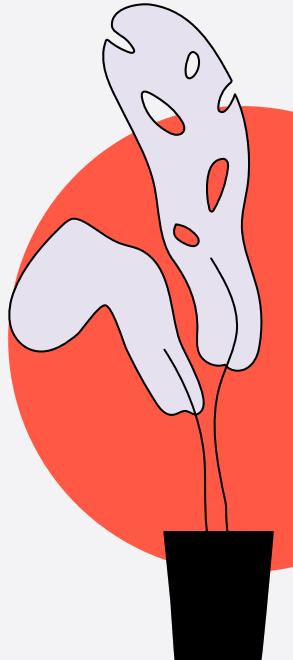


Открытие файла

Чтобы начать работу с файлом, его нужно открыть с помощью функции **open()**. Функции **open()** нужно передать два аргумента:

- Путь к файлу
- Режим доступа к файлу

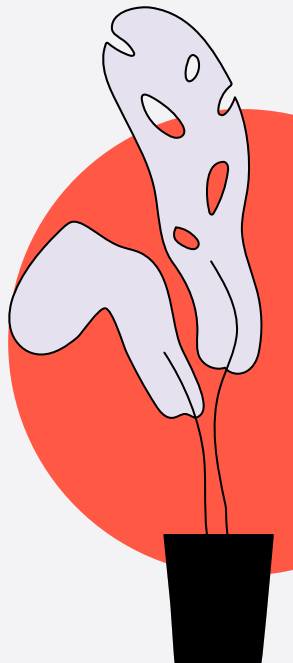
Путь к файлу может быть абсолютным и относительным. В случае с относительным путём поиск файла будет начинаться относительно расположения файла запускаемой программы на Python.



Открытие файла

Второй передаваемый аргумент устанавливает режим доступа к файлу в зависимости от того, что мы собираемся делать с файлом.

'r'	файл открывается для чтения. Если файл не найден, генерируется исключение
'w'	файл открывается для записи. Если файл не найден, он создаётся. Если файл найден, его содержимое будет стёрто
'a'	файл открывается для дозаписи в его конец. Если файл не найден, он создаётся. Если файл найден, новые данные записываются в его конец



Открытие файла

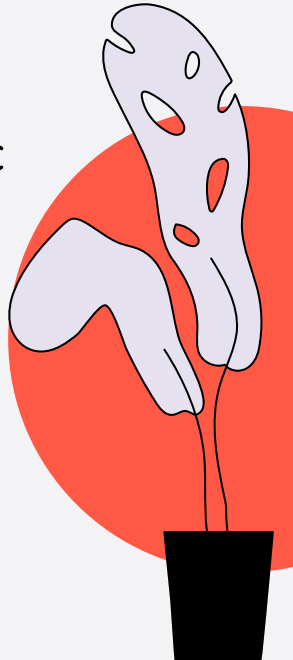
'rb'	бинарный файл открывается для чтения
'wb'	бинарный файл открывается для записи
'x'	файл открывается для записи только если файла не существует, иначе - исключение
't'	файл открывается в текстовом режиме (значение по умолчанию)
'r+'	файл открывается для чтения и записи
'w+'	файл открывается для чтения и записи, если файл не найден – он создаётся



Заккрытие файла


После того, как над содержимым файла были произведены какие-либо действия (например, было считано содержимое файла или же наоборот в файл записано новое содержимое), файл необходимо закрыть.

Заккрытие файла осуществляется с помощью метода **close()**. Вызов данного метода освободит все связанные с файлом занимаемые ресурсы.



Работа с файлами

При работе с файлами можно столкнуться с различными исключениями: нет доступа к файлу, некорректная обработка содержимого файла и т.д. В этом случае программа прекратит своё выполнение на моменте возникновения исключения, так и не дойдя до закрытия файла. Для того, чтобы избежать подобных ситуаций, следует использовать контекстный менеджер **with** при работе с файлами. При использовании оператора **with** не нужно закрывать файл вручную вызовом метода **close()**, **with** автоматически закроет файл сразу после того, как мы закончим работу над этим файлом.



Работа с файлами

Для примера создадим текстовый файл и заполним его некоторым содержимым

Прочитать содержимое файла можно таким способом:

some_text_file.txt

```
1 First Line
2 Second Line
3 Third Line
4 Some information
5 More information here
6 End Line
```

Open file and read data

```
1 with open("some_text_file.txt", "r") as text_file:
2     data = text_file.read()
3
4 print(data)
```

Работа с файлами

Метод **read()** считывает всё содержимое файла. Если нам нужно считать только определённое количество символов из файла, методу **read()** нужно передать аргумент – число, которое указывает, сколько конкретно символов нужно прочитать.



Open file and read data

```
1 with open("some_text_file.txt", "r") as text_file:
2     data = text_file.read(5)
3
4 print(data)
5 # Output: First
```

Работа с файлами

А чтобы получить список всех строк файла, используют метод **readlines()**



Open file and read data

```
1 with open("some_text_file.txt", "r") as text_file:
2     file_lines = text_file.readlines()
3
4 print(file_lines)
5 # Output:
6 # ['First Line\n', 'Second Line\n',
7 # 'Third Line\n', 'Some information\n',
8 # 'More information here\n', 'End Line']
```

Работа с файлами

Для того, чтобы записать какие-либо данные в файл, нужно открыть его с помощью конструкции **with ... as**, но уже указать режим доступа к файлу “**w**” (write).



Open file and write data

```
1 data_for_write = "first line to write\nsecond line to write\nend the writing"
2
3 with open("some_file_to_write.txt", "w") as text_file
4     text_file.write(data_for_write)
```

Работа с файлами

В файл можно записывать не только однострочную информацию, записать в файл строки из списка строк можно так:

Open file and write data

```
1 lines_to_write = ["first line to write", "second line to write", "end the writing"]
2 with open("some_file_to_write.txt", "w") as text_file:
3     text_file.writelines(line + '\n' for line in lines_to_write)
```

И также можно дозаписать оставшуюся информацию в конец файла – режим доступа “a”

Open file and write data

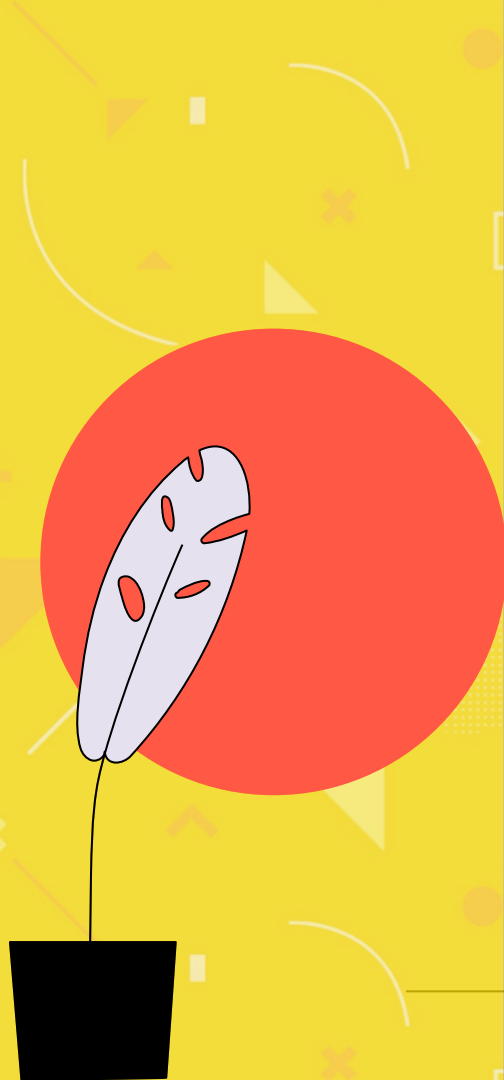
```
1 lines_to_write = ["add one line", "and one more line"]
2 with open("some_file_to_write.txt", "a") as text_file:
3     text_file.writelines(line + '\n' for line in lines_to_write)
```

Работа с внешними данными: JSON



JSON (JavaScript Object Notation) – это легковесный формат обмена данными. Человеку легко читать и вести записи в таком формате, а компьютеры хорошо справляются с его синтаксическим анализом и генерацией.

JSON основан на ЯП JavaScript, но текстовый формат не зависит от языка и может быть использован в том числе и для программ на Python. В основном JSON применяется для передачи данных между программами.



Работа с внешними данными: JSON

Процесс записи внутренних данных программы в JSON-формат называется **сериализация**. Этот термин обозначает трансформацию данных в серию байтов (серийные данные -> сериализация) для хранения или передачи по сети.

Соответственно, **десериализация** – обратная операция, это трансформация данных из JSON-формата во внутреннее представление данных конкретного языка программирования.



Работа с внешними данными: JSON

Допустим, у нас есть **.json** файл со следующим содержимым

```
capitals.json  
  
1 {  
2     "Russia": "Moscow",  
3     "Belarus": "Minsk",  
4     "Kazakhstan": "Nur-Sultan"  
5 }
```

Преобразовать данные из JSON-формата к внутреннему представлению Python можно таким образом:

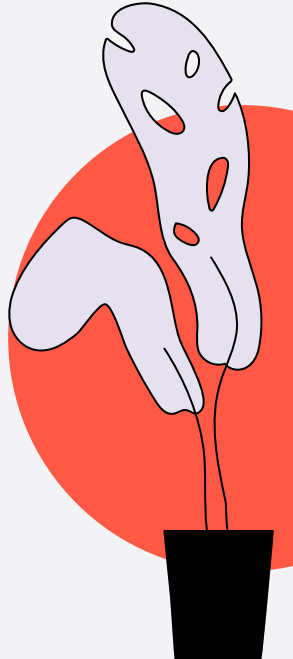
```
Read JSON data  
  
1 import json  
2  
3 with open("capitals.json", "r") as json_file:  
4     data = json.load(json_file)  
5  
6 print(type(data), data)  
7 # Output:  
8 # <class 'dict'> {'Russia': 'Moscow', 'Belarus': 'Minsk', 'Kazakhstan': 'Nur-Sultan'}
```

Работа с внешними данными: JSON

```
Write JSON data

1 import json
2
3 data = {"employees":
4     [
5         {
6             "name": "John",
7             "salary": 1000
8         },
9         {
10            "name": "Bob",
11            "salary": 1200
12        },
13        {
14            "name": "Jessica",
15            "salary": 1100
16        }
17    ]
18 }
19
20 with open("employees.json", "w") as json_file:
21     json.dump(data, json_file, indent=4)
```

Для сериализации
данных используется
метод **dump()**

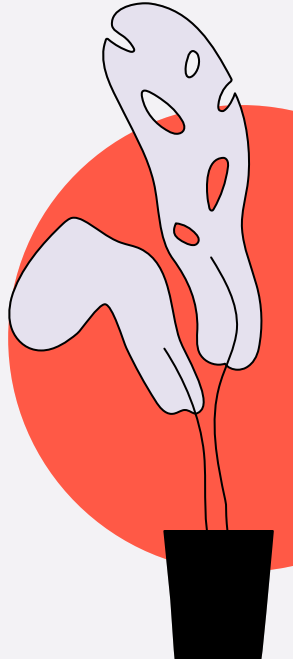


Работа с внешними данными: JSON

```
employees.json

1 {
2     "employees": [
3         {
4             "name": "John",
5             "salary": 1000
6         },
7         {
8             "name": "Bob",
9             "salary": 1200
10        },
11        {
12            "name": "Jessica",
13            "salary": 1100
14        }
15    ]
16 }
```

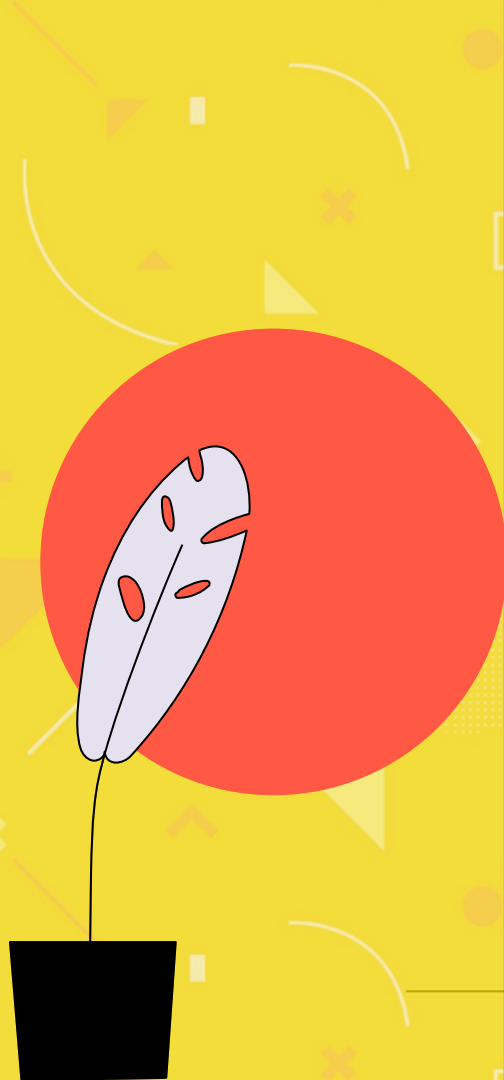
После выполнения сериализации, появится файл с расширением .json, который будет содержать данные, взятые из словаря Python



Работа с внешними данными: CSV

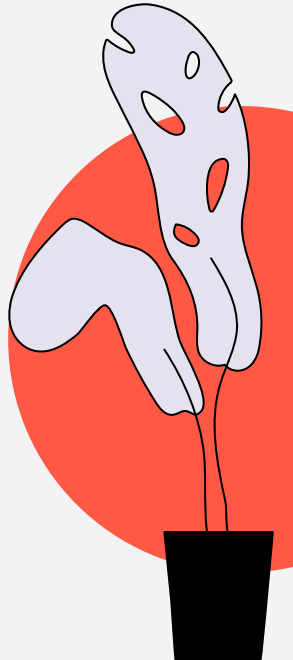


Разработчики часто сталкиваются с задачей обработки больших объёмов структурированных данных. Python имеет библиотеку csv, с помощью которой можно работать со специальными csv-файлами (это своего рода электронные таблицы).



Работа с внешними данными: CSV

CSV-файл – это особый вид файла, который позволяет структурировать большие объёмы данных. По сути, он является обычным текстовым файлом, но каждый новый элемент отделён от другого запятой (**Comma-Separated Value -> CSV**) или другим разделителем. Обычно каждая запись начинается с новой строки. Данные CSV можно легко экспортировать в электронные таблицы или базы данных.



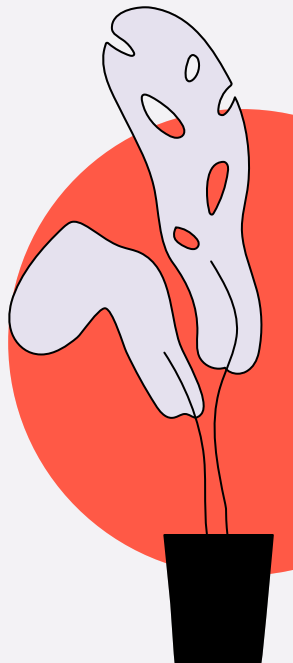
Работа с внешними данными: CSV

В первой строке csv-файла обычно указывается, какая информация будет находиться в каждом столбце. В конце строки данных разделитель (запятая) не ставится, новая запись начинается с новой строки.



CSV Data

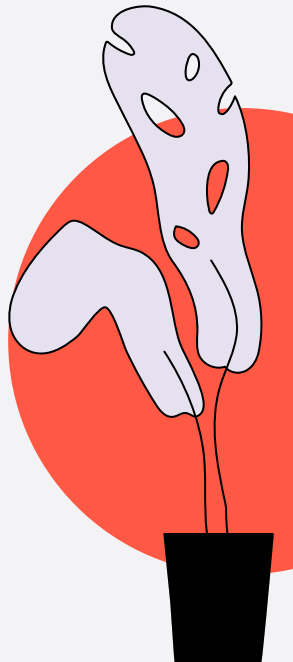
```
1 Имя,Профессия,Год рождения
2 Виктор,Токарь, 1995
3 Сергей,Сварщик,1983
```



Работа с внешними данными: CSV

```
CSV Data

1 import csv
2
3 with open("some_data.csv", encoding='utf-8') as r_file:
4     # create a reader, set separator ","
5     file_reader = csv.reader(r_file, delimiter=",")
6     # counter for counting the number of lines
7     count = 0
8     # read data from csv file
9     for row in file_reader:
10         if count == 0:
11             print(f'Файл содержит столбцы: {"", ".join(row)}')
12         else:
13             print(f'    {row[0]} - {row[1]} и он родился в {row[2]} году.')
14         count += 1
15     print(f'Всего в файле {count} строк.')
16
17 # Output:
18 # Файл содержит столбцы: Имя, Профессия, Год рождения
19 #     Виктор - Токарь и он родился в 1995 году.
20 #     Сергей - Сварщик и он родился в 1983 году.
21 # Всего в файле 3 строк.
```

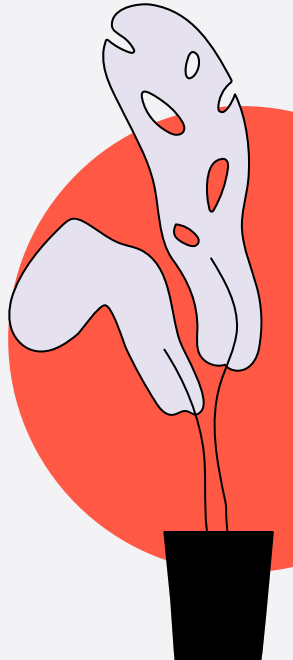


Работа с внешними данными: CSV

CSV Data

```
1 import csv
2 with open("some_data.csv", encoding='utf-8') as r_file:
3     file_reader = csv.DictReader(r_file, delimiter = ",")
4     count = 0
5     for row in file_reader:
6         if count == 0:
7             print(f'Файл содержит столбцы: {", ".join(row)}')
8             print(f' {row["Имя"]} - {row["Профессия"]}', end='')
9             print(f' и он родился в {row["Год рождения"]} году.')
10            count += 1
11        print(f'Всего в файле {count + 1} строк.')
12
13 # Output:
14 # Файл содержит столбцы: Имя, Профессия, Год рождения
15 #     Виктор - Токарь и он родился в 1995 году.
16 #     Сергей - Сварщик и он родился в 1983 году.
17 # Всего в файле 3 строк.
```

Библиотека **csv** позволяет работать с файлами как со словарями. Для этого нужно вместо **reader** создать объект **DictReader**. Обращаться к элементам можно будет по имени столбцов, а не по индексам.



Работа с внешними данными: CSV

Для записи в csv-файл необходимо создать объект **writer** и использовать метод **writerow()** для записи одной строки данных.

```
CSV Data

1 import csv
2 with open("classmates.csv", mode="w", encoding='utf-8') as w_file:
3     file_writer = csv.writer(w_file, delimiter = ",", lineterminator="\r")
4     file_writer.writerow(["Имя", "Класс", "Возраст"])
5     file_writer.writerow(["Женя", "3", "10"])
6     file_writer.writerow(["Саша", "5", "12"])
7     file_writer.writerow(["Маша", "11", "18"])
```

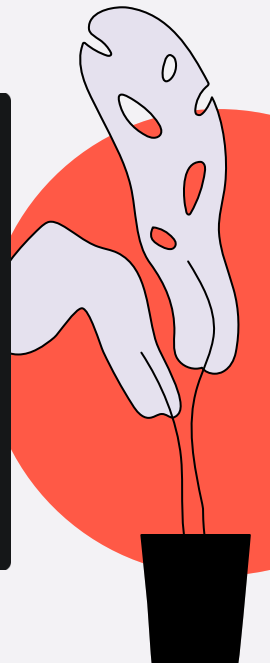


Работа с внешними данными: CSV

Запись в файл также может быть осуществлена с помощью объекта **DictWriter**, он требует явного указания названий столбцов (fieldnames)

```
CSV Data

1 import csv
2 with open("classmates.csv", mode="w", encoding='utf-8') as w_file:
3     names = ["Имя", "Класс", "Возраст"]
4     file_writer = csv.DictWriter(w_file, delimiter=",",
5                                 lineterminator="\r", fieldnames=names)
6     file_writer.writeheader()
7     file_writer.writerow({"Имя": "Саша", "Класс": "3", "Возраст": "6"})
8     file_writer.writerow({"Имя": "Маша", "Класс": "5", "Возраст": "15"})
9     file_writer.writerow({"Имя": "Вова", "Класс": "11", "Возраст": "14"})
```

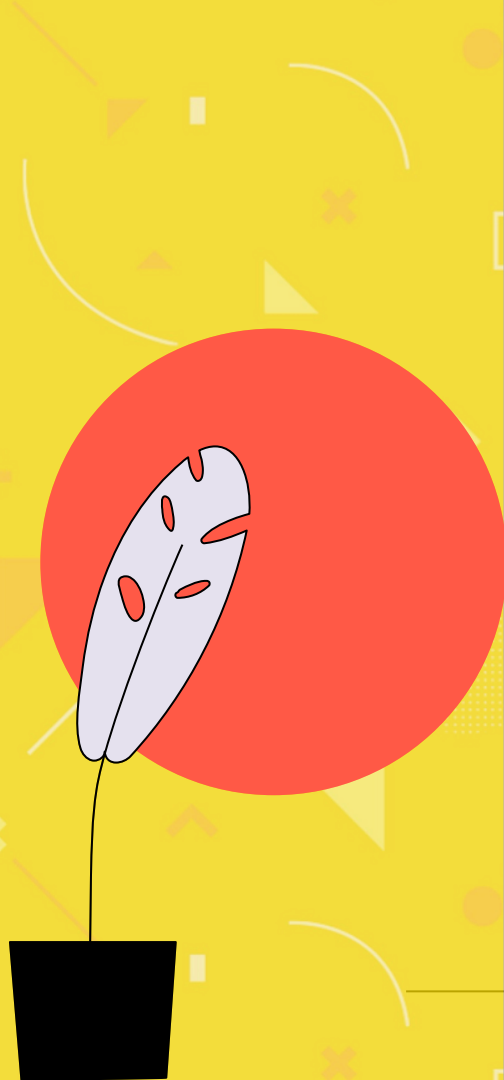


Регулярные выражения



Регулярные выражения (Regex) – это строки, задающие шаблон для поиска определённых фрагментов в тексте. Помимо поиска, с помощью специальных Regex-шаблонов можно манипулировать текстовыми фрагментами – удалять и изменять подстроки частично или полностью.


Регулярки применяются для обработки текстовых данных, в том числе в скриптах для веб-скраппинга.



Регулярные выражения

Регулярные выражения – хороший инструмент для анализа данных. Они эффективно идентифицируют текст (строку, подстроку) в куче текста, проверяя его по заранее заданному шаблону. Некоторые распространённые сценарии – определение адреса электронной почты, телефона или url-адреса из кучи текста.

Также регулярные выражения подходят для таких задач, как проверка данных, например, валидация номера телефона (начинается с '+', содержит в себе только цифры и только определённое количество цифр, содержит ли код и так далее).



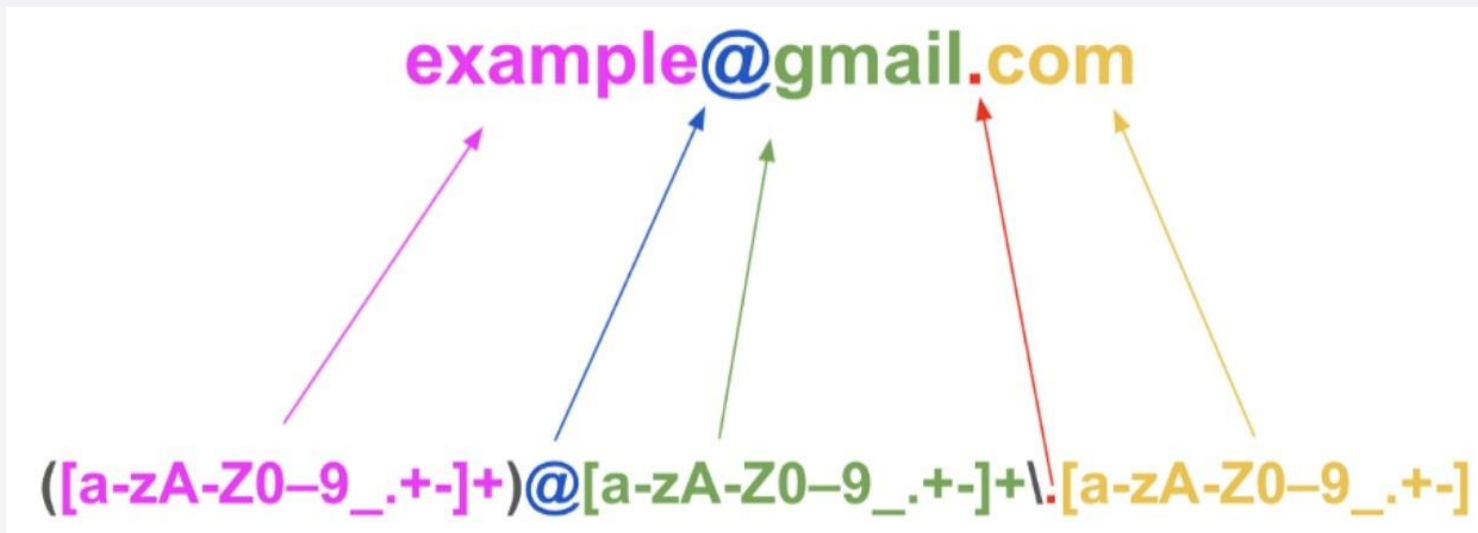
Регулярные выражения

Пример регулярного выражения для проверки email:



Регулярные выражения

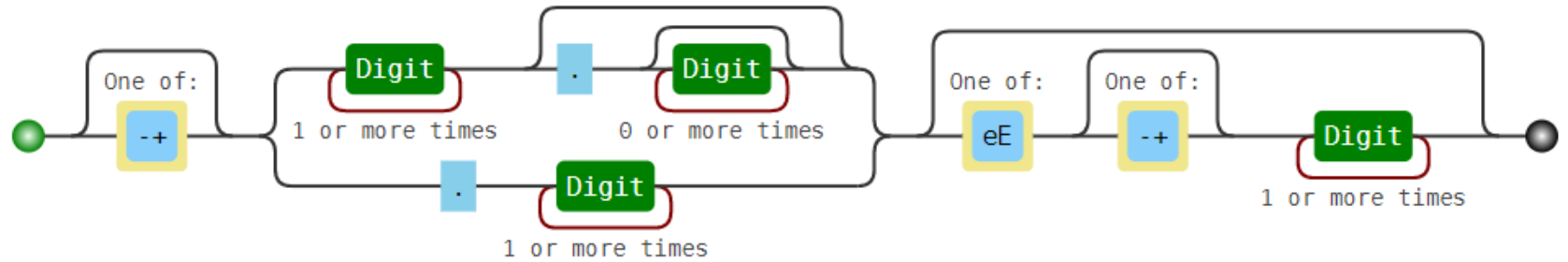
Ещё один пример с почтой:



Регулярные выражения

Схема регулярного выражения:

RegExp: `/[-+]?(?:\d+(?:\.\d*)?|\.\d+)(?:[eE][-+]?\d+)?/`



Основы синтаксиса

Шаблон	Описание	Пример	В тексте
.	один любой символ, кроме \n	м.л.ко	<u>Молоко</u> , <u>малако</u> , <u>Им0л0ко</u> Ихлеб
\d	любая цифра	С\d\d	<u>СУ35</u> , <u>СУ1111</u> , АЛ <u>СУ14</u>
\D	любой символ, кроме цифры	926\D123	<u>926(123, 1926-1234</u>
\s	любой пробельный символ (пробел, табуляция, конец строки и т.п.)	бор\sода	<u>бор ода</u> , <u>бор ода</u>

Основы синтаксиса

Шаблон	Описание	Пример	В тексте
\s	любой непробельный символ	\s123	<u>X123</u> , <u>!12345</u>
\w	любая буква, цифра или _	\w\w\w	<u>Год</u> , <u>f_3</u> , <u>qwert</u>
\W	любая не-буква, не-цифра и не-подчёркивание	com\W	<u>com</u> !, <u>com</u>)
[...]	1 из символов в скобках или из диапазона	[0-9][0-9A-Fa-f]	<u>12</u> , <u>1F</u> , <u>4B</u>

Квантификаторы

Шаблон	Описание	Пример	В тексте
<code>{n}</code>	ровно n повторений	<code>\d{4}</code>	1, 12, <u>1234</u> , 12345
<code>{m, n}</code>	от m до n повторений включительно	<code>\d{2, 4}</code>	1, <u>12</u> , <u>123</u> , <u>1234</u> , 12345
<code>{m, }</code>	не менее m повторений	<code>\d{3, }</code>	1, 12, <u>123</u> , <u>1234</u> , <u>12345</u>
<code>{, n}</code>	не более n повторений	<code>\d{, 2}</code>	<u>1</u> , <u>12</u> , <u>123</u>

Квантификаторы

Шаблон	Описание	Пример	В тексте
?	0 или 1 вхождение (синоним {0, 1})	валы?	<u>вал</u> , <u>валы</u> , <u>валов</u>
*	0 или более (синоним {0, })	СУ\d*	<u>СУ</u> , <u>СУ1</u> , <u>СУ12</u>
+	1 или более вхождений (синоним {1, })	a\)+	<u>a</u>), <u>a</u>)), <u>a</u>))), <u>ba</u>)))]

Жадные квантификаторы

По умолчанию квантификаторы *жадные*, то есть захватывают максимально возможное число символов. Добавление знака ? Делает их *ленивыми*, тогда они захватывают минимально возможное число символов.

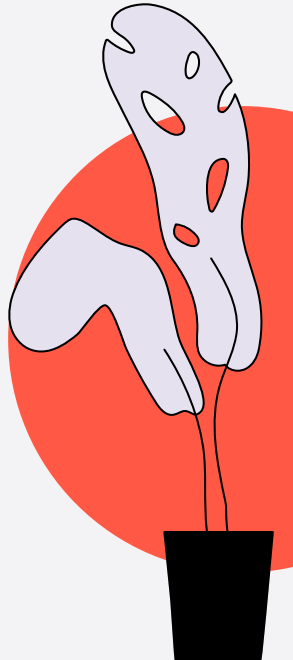
Например, шаблону `\(.*\)` будет подходить весь такой текст: `(a + b) * (c + d) * (e + f)`

Но если мы сделаем квантификатор ленивым, то есть вида `\(.*?\)`, то он захватит из этого текста только `(a + b)`

Границы шаблона

Подход с жадностью квантификаторов решает очень важную проблему – проблему границы шаблона. Скажем, шаблон `\d+` захватывает максимально возможное количество цифр. Поэтому можно быть уверенными, что перед найденным шаблоном не идёт цифра и после найденного шаблона цифр тоже больше нет – квантификатор захватит всё, что только можно.

Однако если в шаблоне есть не жадные части, например, явный текст, то подстрока может быть найдена неудачно. В тех случаях, когда это важно, условие на границу шаблона нужно обязательно добавлять в регулярку.




Пересечение подстрок

В обычной ситуации регулярки позволяют найти только непересекающиеся шаблоны. Вместе с проблемой границы слова это делает их использование в некоторых случаях более сложным. Например, если мы решим искать email адреса при помощи неудачной регулярки `\w+@\w+`, то в неудачном случае найдём вот что:

foo@boo@goo@moo@roo@zoo

То есть, это, с одной стороны, и не email, а с другой – это и не все подстроки вида текст-собака-текст, так как варианты `boo@goo` и `moo@roo` пропущены.



Регулярки в Python – модуль re

Функция	Назначение
<code>re.search(pattern, string)</code>	Найти в строке string первую строчку, подходящую под шаблон pattern
<code>re.fullmatch(pattern, string)</code>	Проверить, подходит ли вся строка string под шаблон pattern
<code>re.split(pattern, string, maxsplit=0)</code>	Аналог <code>str.split()</code> , только разделение происходит по подстрокам, подходящим под шаблон pattern
<code>re.findall(pattern, string)</code>	Найти в строке string все непересекающиеся шаблоны pattern

Регулярки в Python – модуль re

Функция	Назначение
<code>re.finditer(pattern, string)</code>	Итератор по всем непересекающимся шаблонам <code>pattern</code> в строке <code>string</code>
<code>re.sub(pattern, repl, string, count=0)</code>	Заменить в строке <code>string</code> все непересекающиеся шаблоны <code>pattern</code> на <code>repl</code>
<code>re.compile(pattern)</code>	Собирает регулярное выражение в объект для последующего использования в других re-функциях
<code>re.match(pattern, string)</code>	Найти в начале строки <code>string</code> шаблон <code>pattern</code>

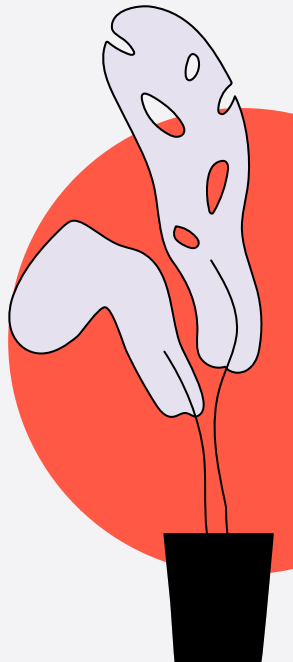
Регулярки в Python – модуль re

`re.match(pattern, string)` находит совпадение только в том случае, если соответствующая шаблону подстрока находится в начале строки, по которой ведётся поиск



`re.match(pattern, string)`

```
1 import re
2
3
4 print(re.match(r'Мама', 'Мама мыла раму'))
5 # Output: <re.Match object; span=(0, 4), match='Мама'>
6
7 print(re.match(r'мыла', 'Мама мыла раму'))
8 # Output: None
```



Регулярки в Python – модуль re

re.search(pattern, string) ищет совпадения по всей строке, при этом возвращает только первое совпадение, даже если в строке, по которой ведётся поиск, их больше



re.search(pattern, string)

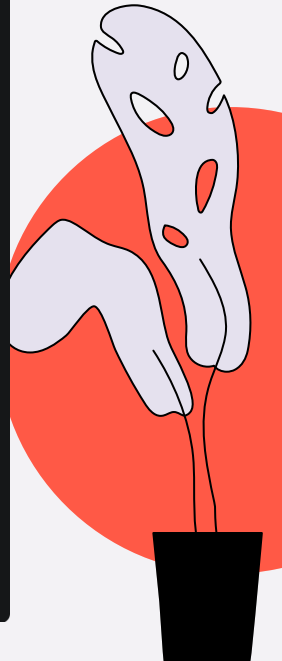
```
1 import re
2
3
4 print(re.search(r'Мама', 'Мама мыла раму'))
5 # Output: <re.Match object; span=(0, 4), match='Мама'>
6
7 print(re.search(r'мыла', 'Мама мыла раму'))
8 # Output: <re.Match object; span=(5, 9), match='мыла'>
```

Регулярки в Python – модуль re

`re.finditer(pattern, string)` возвращает итератор с объектами, к которым можно обращаться через цикл

```
re.finditer(pattern, string)

1 import re
2
3
4 some_str = 'Мама мыла раму, а потом ещё раз мыла, потому что не домыла'
5 results = re.finditer(r'мыла', some_str)
6 print(results)
7 # Output: <callable_iterator object at 0x000001C4CDE446D0>
8
9 for match in results:
10     print(match)
11 # Output: <re.Match object; span=(5, 9), match='мыла'>
12 # Output: <re.Match object; span=(32, 36), match='мыла'>
13 # Output: <re.Match object; span=(54, 58), match='мыла'>
```



Как работать с Match-объектами

В match-объектах находится много интересного.
Посмотрим на match-объект из первого примера:

```
<re.Match object; span=(0, 4), match='Мама'>
```

span – это индексы начала и конца найденной подстроки в тексте

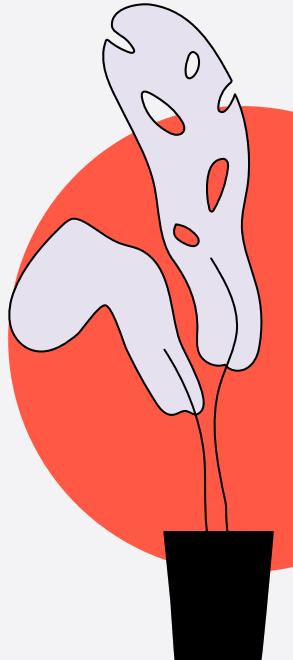
match – это, собственно, найденная подстрока

Как работать с Match-объектами

group

```
1 import re
2
3
4 match = re.match(r'(M)(ама)', 'Мама мыла раму')
5
6 print(match.group(1))
7 # Output: M
8
9 print(match.group(2))
10 # Output: ама
11
12 print(match[1])
13 # Output: M
14
15 print(match[2])
16 # Output: ама
17
18 print(match.group(1,2))
19 # Output: ('M', 'ама')
```

Если регулярное выражение поделено на группы, то, начиная с единицы, можно обращаться не к целой найденной подстроке, а к группе.



Как работать с Match-объектами

Если группы поименованы, то в качестве аргумента метода **group** можно передавать название группы, которую нужно получить

```
group

1 import re
2
3
4 match = re.match(r'(?P<first_letter>M)(?P<rest_letters>ама)', 'Мама мыла раму')
5
6 print(match.group('first_letter'))
7 # Output: Мама
8
9 print(match.group('rest_letters'))
10 # Output: ама
```

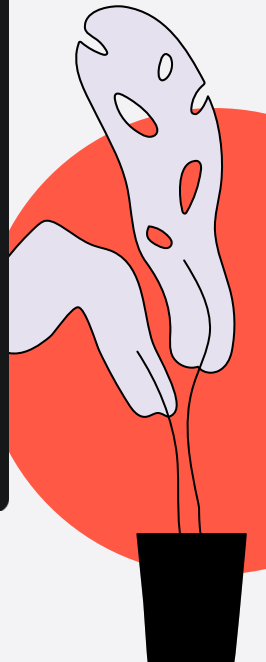
Как работать с Match-объектами

Метод **groups** возвращает кортеж с группами



groups

```
1 import re
2
3
4 match = re.match(r'(M)(ама)', 'Мама мыла раму')
5
6 print(match.groups())
7 # Output: ('M', 'ама')
```



Как работать с Match-объектами

span возвращает кортеж с индексами начала и конца подстроки в исходном тексте. Если нужно получить только первый индекс, можно использовать метод **start**, последний – **end**

```
span, start, end

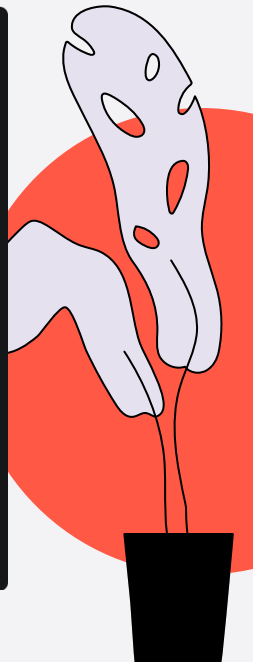
1 import re
2
3
4 match = re.search(r'мыла', 'Мама мыла раму')
5
6 print(match.span())
7 # Output: (5, 9)
8
9 print(match.start())
10 # Output: 5
11
12 print(match.end())
13 # Output: 9
```

re-функции без Match-объекта

`re.findall(pattern, string)` возвращает просто список совпадений. Если в регулярке есть деление на группы, вернёт список кортежей с группами.

```
findall

1 import re
2
3
4 text = 'Мама мыла раму, а папа был на пилораме, потому что работает на лесопилке.'
5
6 match_list = re.findall(r'\b\w{4}\b', text)
7
8 print(match_list)
9 # Output: ['Мама', 'мыла', 'раму', 'папа']
10
11 match_list = re.findall(r'\b(\w{1})(\w{3})\b', text)
12 print(match_list)
13 # Output: [('М', 'ама'), ('м', 'ыла'), ('р', 'аму'), ('п', 'апа')]
```



re-функции без Match-объекта

`re.split(pattern, string, [maxsplit=0])` – аналог `str.split()`. Аргумент `maxsplit` указывает, на сколько частей максимально можно поделить строку. По умолчанию – 0, то есть не устанавливает никаких ограничений.

```
split

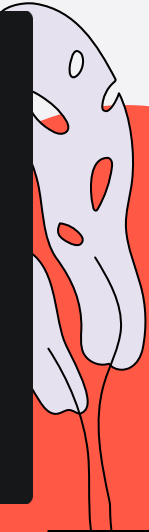
1 import re
2
3
4 text = 'Мама мыла раму, а папа был на пилораме, потому что работает на лесопилке.'
5
6 split_string = re.split(r', ', text)
7 print(split_string)
8 # Output:
9 # ['Мама мыла раму',
10 # 'а папа был на пилораме',
11 # 'потому что работает на лесопилке.']
12
13
14 split_string = re.split(r', ', text, maxsplit=1)
15 print(split_string)
16 # Output:
17 # ['Мама мыла раму',
18 # 'а папа был на пилораме,
19 # потому что работает на лесопилке.']
```

re-функции без Match-объекта

`re.sub(pattern, repl, string)` требует дополнительного указания аргумента в виде строки, на которую и будет заменять найденные совпадения

```
sub

1 import re
2
3
4 text = 'Мама мыла раму, а папа был на пилораме, потому что работает на лесопилке.'
5
6 new_string = re.sub(r'Мама', 'Дочка', text)
7
8 print(new_string)
9 # Output: Дочка мыла раму, а папа был на пилораме, потому что работает на лесопилке.
```



re-функции без Match-объекта

Дополнительные возможности у функции появляются при применении групп. В качестве аргумента замены можно передавать не только строку, но и ссылку на номер группы в виде \n, где n – номер группы. Тогда на нужное место будет подставлена соответствующая группа из шаблона.

```
sub

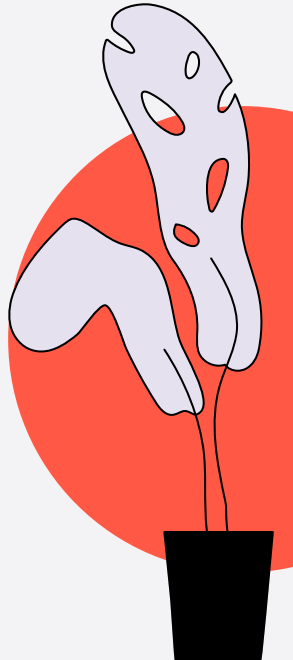
1 import re
2
3
4 text = 'Бендер Остап Ибрагимович, директор 000 "Рога и копыта"'
5
6 new_string = re.sub(r'(\w+) (\w+) (\w+)',',', r'\2 \3 \1 -', text)
7
8 print(new_string)
9 # Output: Остап Ибрагимович Бендер – директор 000 "Рога и копыта"
```

re-функции без Match-объекта

re.compile(pattern) используется для ускорения и упрощения кода, когда одно и то же регулярное выражение используется в коде несколько раз

```
compile

1 import re
2
3
4 pattern = re.compile(r'Мама')
5
6 print(pattern.search('Мама мыла раму'))
7 # Output: <re.Match object; span=(0, 4), match='Мама'>
8
9 print(pattern.sub('Дочка', 'Мама мыла раму'))
10 # Output: Дочка мыла раму
```



Деление r-строк

Нередко в регулярных выражениях нужно учесть сразу много вариантов, из-за чего их структура усложняется. А regex даже простые и короткие читать нелегко, не говоря уже о длинных.

Чтобы облегчить чтение регулярок, в Python r-строки можно делить также, как и обычные.

```
r-strings

1 print(re.findall(r'(?:\d{4})'
2                 r'|'
3                 r'(?:[IVX]+ век)', text))
```

Задания

1. Работа с модулем os

Есть папка, в которой лежат файлы с разными расширениями. Программа должна:

- Вывести имя вашей ОС
- Вывести путь до папки, в которой вы находитесь
- Рассортировать файлы по расширениям, например, для текстовых файлов создается папка, в неё перемещаются все файлы с расширением .txt, то же самое для остальных расширений
- После рассортировки выводится сообщение типа «в папке с текстовыми файлами перемещено 5 файлов, их суммарный размер - 50 гигабайт»

Продолжение ->

Задания

- Как минимум один файл в любой из получившихся поддиректорий переименовать. Сделать вывод сообщения типа «Файл data.txt был переименован в some_data.txt»
- Программа должна быть кроссплатформенной – никаких хардкодов с именем диска и слэшами.

2. Замена имён в судебном решении

Написать программу, которая заменит в тексте ФИО подсудимого на N. Каждое слово в ФИО начинается с заглавной буквы, фамилия может быть двойная.

Продолжение ->

Задания

Вводимый текст:

Подсудимая Эверт-Колокольцева Елизавета Александровна в судебном заседании вину инкриминируемого правонарушения признала в полном объёме и суду показала, что 14 сентября 1876 года, будучи в состоянии алкогольного опьянения от безысходности, в связи с состоянием здоровья позвонила со своего стационарного телефона в полицию, сообщив о том, что у неё в квартире якобы заложена бомба. После чего приехали сотрудники полиции, скорая и пожарные, которым она сообщила, что бомба — это она.

Вывод:

«Подсудимая N в судебном заседании» и далее по тексту.

Задания

3. Напишите программу, которая считывает текст из файла (в файле может быть больше одной строки) и выводит в новый файл самое часто встречаемое слово в каждой строке и число – счётчик количества повторений этого слова в строке.

4. Напишите программу, которая получает на вход строку с названием текстового файла и выводит на экран содержимое этого файла, заменяя все запрещённые слова звездочками. Запрещённые слова, разделённые символом пробела, должны храниться в файле `stop_words.txt`.

Продолжение ->

Задания

4 (Продолжение). Программа должна находить запрещённые слова в любом месте файла, даже в середине другого слова. Замена независима от регистра: если в списке запрещённых есть слово `exam`, то замениться должны `exam`, `eXam`, `EXAm` и другие вариации.

Пример: в `stop_words.txt` записаны слова: `hello email python the exam wor is`

Текст файла для цензуры выглядит так: **Hello, World! Python IS the programming language of thE future. My EMAIL is... PYTHON as AwESOME!**

Тогда итоговый текст: *******, ***ld! ***** ** *** programming language of *** future. My ***** **... ***** ** awesome!!!!**

Задания

5. В текстовый файл построчно записаны фамилия и имя учащихся класса и оценка за контрольную. Вывести на экран всех учащихся, чья оценка меньше трёх баллов.

6. В файл записано некоторое содержимое (буквы, цифры, пробелы, специальные символы и т.д.). Числом назовём последовательность цифр, идущих подряд. Вывести сумму всех чисел, записанных в файле.

Пример:

Входные данные: **123 aaa456 1x2y3z 4 5 6**

Выходные данные: **600**

Задания

7. Дан текстовый файл с несколькими строками.
Зашифровать шифром Цезаря, при этом шаг зависит от
номера строки: для первой строки шаг 1, для второй – 2 и т.д.
Пример:

Входные данные:

Hello
Hello
Hello
Hello

Выходные данные:

Ifmmp
Jgnnq
Khoor
Lipps

Задания

8. JSON и CSV.

Исходные данные:

<https://drive.google.com/drive/folders/1KH3pJewo3QKl3mua2XnJDv9xN2LxusbE?usp=sharing>

Пункты задания:

0. Есть данные в формате JSON – взять с диска с исходными данными.
1. Реализовать функцию, которая считывает данные из исходного JSON-файла и преобразует их в формат CSV

Задания

2. Реализовать функцию, которая сохранит данные в CSV-файл (данные должны сохраняться независимо от их количества – если добавить в исходный JSON-файл ещё одного сотрудника, работа программы не должна нарушаться).

3. Реализовать функцию, которая добавит информацию о новом сотруднике в JSON-файл. Пошагово вводятся все необходимые данные о сотруднике, формируются данные для записи.

Задания

4. Такая же функция для добавления информации о новом сотруднике в CSV-файл.

5. Реализовать функцию, которая выведет информацию об одном сотруднике по имени. Имя для поиска вводится с клавиатуры.

6. Реализовать функцию фильтра по языку: с клавиатуры вводится язык программирования, выводится список всех сотрудников, кто владеет этим языком программирования.

7. Реализовать функцию фильтра по году: ввести с клавиатуры год рождения, вывести средний рост всех сотрудников, у которых год рождения меньше заданного.

Задания

8. Программа должна представлять собой интерактив – пользовательское меню с возможностью выбора определённого действия (действия – функции из предыдущих пунктов + выход из программы). Пока пользователь не выберет выход из программы, должен предлагаться выбор следующего действия.

Контрольные вопросы

1. Что такое кодировка?
2. Как работать с файлами в Python?
3. Зачем нужна конструкция `with ... as`?
4. Что такое контекстный менеджер?
5. Что такое файловый дескриптор?
6. Разница между относительным и абсолютным путями?
7. Что такое JSON?
8. Что такое CSV?
9. Что такое регулярные выражения?
10. Зачем нужны регулярные выражения?