



Лекция 8.

Сериализация

Погружение в Python



Оглавление

На этой лекции мы	2
Краткая выжимка, о чём говорилось в предыдущей лекции	3
Термины лекции	3
Подробный текст лекции	
Сериализация данных	3
1. JSON	4
Преобразование JSON в Python	6
Преобразование Python в JSON	7
Дополнительные параметры dump и dumps	9
Задание	10
2. CSV	11
Формат CSV	11
Модуль CSV	12
Чтение CSV	12
Запись CSV	13
Чтение в словарь	15
Запись из словаря	15
Задание	17
3. Pickle	17
Допустимые типы данных для преобразования	18
Десериализация	20
Задание	21
Вывод	21

На этой лекции мы

1. Разберёмся в сериализации и десериализации данных
2. Изучим самый популярный формат сериализации - JSON
3. Узнаем о чтении и записи таблиц в формате CSV
4. Разберёмся с внутренним сериализатором Python - модулем pickle

Краткая выжимка, о чём говорилось в предыдущей лекции

На прошлой лекции мы:

1. Разобрались в особенностях работы с файлами и каталогами в Python
2. Изучили функцию `open` для работы с содержимым файла
3. Узнали о возможностях стандартной библиотеки для работы с файлами и каталогами

Термины лекции

- **Сериализация** — это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл.
- **Десериализация** — восстановление объектов из байт, сохранение которых было произведено ранее. Процедура выгрузки «зафиксированной» информации пользователем.

Подробный текст лекции

Сериализация данных

Под термином сериализация принято понимать преобразование объектов, а точнее хранящихся в них данных в набор байт для пересылки или хранения. При этом сериализованные данные можно подвергнуть десериализации, т.е. восстановлению данных из байт в исходные объекты. Особенностью сериализации является превращение сложных многоуровневых объектов различной степени вложенности в линейный набор байт. Такой поток легко передавать по каналам связи, хранить в файле и т.п.

Далее на лекции рассмотрим несколько популярных форматов данных, которые используются не только в Python, но и в современном программировании. Отдельно коснёмся сериализации специфичной только для Python.

1. JSON

JSON (JavaScript Object Notation) — это популярный формат текстовых данных, который используется для обмена данными в современных веб- и мобильных приложениях. Кроме того, JSON используется для хранения неструктурированных данных в файлах журналов или базах данных NoSQL.

Не смотря на JavaScript в названии формат JSON не привязывается к конкретному ЯП. Созданный на Python JSON может быть прочитан приложением под Андроид на Java и под iPhone на Swift. Как результат JSON стал одним из самых популярных форматов передачи данных.

Формат JSON

Рассмотрим пример JSON файла

```
{  
  "id": 2,  
  "name": "Ervin Howell",
```

```

"username": "Antonette",
"email": [
    "Shanna@melissa.tv",
    "antonette@howel.com"
],
"address": {
    "street": "Victor Plains",
    "suite": "Suite 879",
    "city": "Wisokyburgh",
    "zipcode": "90566-7771",
    "geo": {
        "lat": "-43.9509",
        "lng": "-34.4618"
    }
},
"phone": "010-692-6593 x09125",
"website": "anastasia.net",
"company": {
    "name": "Deckow-Crist",
    "catchPhrase": "Proactive didactic contingency",
    "bs": "synergize scalable supply-chains"
}
}

```

Как несложно заметить JSON похож на словарь Python. В целом верно, но есть небольшие отличия. Некоторые типы данных Python не совпадают с типами в формате JSON. Поэтому при конвертации словаря в JSON и последующей конвертации в словарь типы данных могут оказаться иными.

Python	JSON	Python
dict	object	dict
list, tuple	array	list
str	string	str
int	number (int)	int
float	number (real)	float
True	true	True
False	false	False
None	null	None

Обратите внимание, что list и tuple конвертируется в массив array, а при обратной конвертации получаем только list.

Если мы храним несколько JSON объектов, в Python обычно используют list. Для JSON в этом случае используется array — в квадратных скобках записываются JSON объекты разделённые запятыми.

Модуль JSON

Для работы с форматом в Python есть встроенный модуль json. Для его использования достаточно импорта в начале файла:

```
import json
```

Рассмотрим четыре основных функции модуля.

- **Преобразование JSON в Python**

Договоримся, что представленный выше объект JSON сохранён в виде текстового файла user.json в кодировке UTF-8 в той же директории, что и исполняемый код.

```
import json

with open('user.json', 'r', encoding='utf-8') as f:
    json_file = json.load(f)

print(f'{type(json_file) = }\n{json_file = }')
print(f'{json_file["name"] = }')
print(f'{json_file["address"]["geo"] = }')
print(f'{json_file["email"] = }')
```

Функция load загрузила объект из файла и произвела его десериализацию — превращение в словарь dict. Дальнейшие манипуляции со словарём не вызовут затруднений у Python разработчиков.

А теперь представим, что мы подготовили информацию в виде многострочного str в python и хотим превратить его из JSON строки в объекты Python.

```
import json

json_text = """
[
    {
        "userId": 1,
```

```

        "id": 9,
        "title": "nesciunt iure omnis dolorem tempora et accusantium",
        "body": "consectetur animi nesciunt iure dolore"
    },
    {
        "userId": 1,
        "id": 10,
        "title": "optio molestias id quia eum",
        "body": "quo et expedita modi cum officia vel magni"
    },
    {
        "userId": 2,
        "id": 11,
        "title": "et ea vero quia laudantium autem",
        "body": "delectus reiciendis molestiae occaecati non minima eveniet qui voluptatibus"
    },
    {
        "userId": 2,
        "id": 12,
        "title": "in quibusdam tempore odit est dolorem",
        "body": "praesentium quia et ea odit et ea voluptas et"
    }
]"""

print(f'{type(json_text) = }\n{json_text = }')
json_list = json.loads(json_text)
print(f'{type(json_list) = }\t{len(json_list) = }\n{json_list = }')

```

Функция `loads` принимает на вход строку хранящуюся как структуру JSON и преобразует её к нужным типам. В нашем примере получили список `list` с четырьмя словарями внутри.

Запомнить различия между функциями просто. Окончание `s` у `loads` намекает на строку. А `load` требует объект с методом `read` для чтения информации. Напомним, что файловый дескриптор имеет метод `read` для чтения информации из файла.



Важно! При открытии файлов важно учитывать их размер. Огромные JSON объекты даёт высокую нагрузку на процессор и оперативную память.

• Преобразование Python в JSON

Что делать, если мы хотим превратить Словарь Python в JSON объект? Для этого используем функции сериализации `dump` и `dumps`. Смысл окончания `s` у `dumps` такой же как и у `loads`.

```
import json

my_dict = {
    "first_name": "Джон",
    "last_name": "Смит",
    "hobbies": ["кузнечное дело", "программирование",
"путешествия"],
    "age": 35,
    "children": [
        {
            "first_name": "Алиса",
            "age": 5
        },
        {
            "first_name": "Маруся",
            "age": 3
        }
    ]
}

print(f'{type(my_dict) = }\n{my_dict = }')
with open('new_user.json', 'w') as f:
    json.dump(my_dict, f)
```

Создаём словарь, открываем файл для записи и сохраняем содержимое функцией `dump`. В качестве первого аргумента функция получает словарь или список, вторым передаем файловый дескриптор либо другой объект, поддерживающий метод `write`. Сама функция `dump` ничего не возвращает.

Если мы откроем файл `new_user.json`, увидим одну длинную строку:

```
{ "first_name":      "\u0414\u0436\u043e\u043d",      "last_name":
"\u0421\u043c\u0438\u0442",      "hobbies":
["\u043a\u0443\u0437\u043d\u0435\u0447\u043d\u043e\u0435 \u0434\u0435\u043b\u043e",
"\u043f\u0440\u043e\u0433\u0440\u0430\u043c\u043c\u0438\u0440\u043e\u0432\u0430\u043d\u0438\u0435",
"\u043f\u0443\u0442\u0435\u0448\u0435\u0441\u0442\u0432\u0438\u044f"],
"age":      35,      "children":      [{"first_name":
"\u0410\u043b\u0438\u0441\u0430",      "age":      5},
{"first_name":
```



```
"\u041c\u0430\u0440\u0443\u0441\u044f", "age": 3}}}
```

Как вы видите символы отличные от ASCII были заменены на специальные коды. Это не означает, что файл повреждён или что-то пошло не так. Проведём десериализацию уже знакомым способом и проверим целостность данных.

```
import json

with open('new_user.json', 'r', encoding='utf-8') as f:
    new_dict = json.load(f)
print(f'{new_dict = }')
```

Если же мы хотим отказаться от символов экранирования в JSON файле, следует установить дополнительный параметр `ensure_ascii` в значение `ложь`.

```
with open('new_user.json', 'w', encoding='utf-8') as f:
    json.dump(my_dict, f, ensure_ascii=False)
```

Воспользуемся словарём `my_dict` ещё раз для проверки функции `dumps`

```
import json

my_dict = {
    "first_name": "Джон",
    "last_name": "Смит",
    "hobbies": ["кузнечное дело", "программирование", "путешествия"],
    "age": 35,
    "children": [
        {
            "first_name": "Алиса",
            "age": 5
        },
        {
            "first_name": "Маруся",
            "age": 3
        }
    ]
}

dict_to_json_text = json.dumps(my_dict)
print(f'{type(dict_to_json_text) = }\n{dict_to_json_text = }')
```

На выходе получаем объект типа `str` хранящий структуру `json`. Подобные данные мы видели в сохранённом файле.

- **Дополнительные параметры dump и dumps**

Функции для сериализации объектов в JSON поддерживают несколько дополнительных параметров. Они позволяют сделать полученные объекты более удобочитаемыми для пользователя. Разберём на примере функции dumps. Но стоит помнить, что функция dump обладает такими же параметрами с тем же смыслом.

```
import json

my_dict = {
    "id": 123,
    "name": "Clementine Bauche",
    "username": "Cleba",
    "email": "cleba@corp.mail.ru",
    "address": {
        "street": "Central",
        "city": "Metropolis",
        "zipcode": "123456"
    },
    "phone": "+7-999-123-45-67"
}

res = json.dumps(my_dict, indent=2, separators=(',', ':'),
sort_keys=True)
print(res)
```

- Параметр indent отвечает за форматирование с отступами. Теперь JSON выводится не в одну строку, а в несколько. Читать стало удобнее, но размер увеличился.
- Параметр separators принимает на вход кортеж из двух строковых элементов. Первый — символ разделитель элементов. По умолчанию это запятая и пробел. Второй — разделитель ключа и значения. По умолчанию это двоеточие и пробел. Передав запятую и двоеточие без пробела JSON стал компактнее.
- Параметр sort_keys отвечает за сортировку ключей по алфавиту. Нужна сортировка или нет, решать только вам.

Задание

Перед вами несколько строк кода. Какой объект будет получен после его выполнения? У вас три минуты.

```
import json

a = 'Hello world!'
b = {key: value for key, value in enumerate(a)}

c = json.dumps(b, indent=3, separators=('; ', '= '))
print(c)
```

2. CSV

CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделённых запятыми.

CSV часто используют там, где удобно хранить информацию в таблицах. Выгрузки из баз данных, из электронных таблиц для анализа данных.

Формат CSV

При работе с CSV стоит помнить о том, что формат не до конца стандартизирован. Например запятая как символ разделитель может являться частью текста. Чтобы не учитывать такие запятые, можно использовать кавычки. Но тогда кавычки не могут быть частью строки. Кроме того десятичная запятая используется для записи вещественных чисел в некоторых странах. Все эти особенности необходимо учитывать при работе с CSV файлами.

Рассмотрим тестовый CSV файл biostats.csv

```
"Name","Sex","Age","Height (in)","Weight (lbs)"
"Alex","M",41,74,170
"Bert","M",42,68,166
"Carl","M",32,70,155
"Dave","M",39,72,167
"Elly","F",30,66,124
"Fran","F",33,66,115
```

```
"Gwen","F",26,64,121
"Hank","M",30,71,158
"Ivan","M",53,72,175
"Jake","M",32,69,143
"Kate","F",47,69,139
"Luke","M",34,72,163
"Myra","F",23,62,98
"Neil","M",36,75,160
"Omar","M",38,70,145
"Page","F",31,67,135
"Quin","M",29,71,176
"Ruth","F",28,65,131
```

В первой строке могут содержаться заголовки столбцов как в нашем случае. Строки со второй и до конца файла представляют записи. Одна строка — одна запись. Текстовая информация заключена в кавычки, а числа указаны без них.

Подобные текстовые CSV файлы легко получить выгрузив данные из Excel или другой электронной таблицы указав нужный формат. По сути CSV является промежуточным файлом между Excel и Python.

Модуль CSV

Для работы с форматом в Python есть встроенный модуль csv. Для его использования достаточно импорта в начале файла:

```
import csv
```

Рассмотрим основные функции модуля.


- **Чтение CSV**

Функция csv.reader принимает на вход файловый дескриптор и построчно читает информацию.

```
import csv

with open('biostats.csv', 'r', newline='') as f:
    csv_file = csv.reader(f)
    for line in csv_file:
        print(line)
```

```
print(type(line))
```

 **Важно!** При работе с CSV необходимо указывать параметр `newline=""` во время открытия файла.

Кроме файлового дескриптора можно передать любой объект поддерживающий итерацию и возвращающий строки. Также функция `reader` может принимать диалект отличный от заданного по умолчанию — `"excel"`. А при необходимости и дополнительные параметры форматирования, если файл имеет свои особенности. Файл `biostats_tab.csv` хранит те же данные, что и файл выше, но вместо разделителя используется символ табуляции. По сути это разновидность TSV — файл с разделителем табуляцией.

"Name"	"Sex"	"Age"	"Height (in)"	"Weight (lbs)"
"Alex"	"M"	41	74	170
"Bert"	"M"	42	68	166
"Carl"	"M"	32	70	155
"Dave"	"M"	39	72	167
"Elly"	"F"	30	66	124
"Fran"	"F"	33	66	115
"Gwen"	"F"	26	64	121
"Hank"	"M"	30	71	158
"Ivan"	"M"	53	72	175
"Jake"	"M"	32	69	143
"Kate"	"F"	47	69	139
"Luke"	"M"	34	72	163
"Myra"	"F"	23	62	98
"Neil"	"M"	36	75	160
"Omar"	"M"	38	70	145
"Page"	"F"	31	67	135
"Quin"	"M"	29	71	176
"Ruth"	"F"	28	65	131

Добавим несколько параметров для его чтения

```
import csv

with open('biostats_tab.csv', 'r', newline='') as f:
    csv_file = csv.reader(f, dialect='excel-tab',
quoting=csv.QUOTE_NONNUMERIC)
    for line in csv_file:
        print(line)
    print(type(line))
```

- `dialect='excel-tab'` — указали диалект с табуляцией в качестве разделителя
- `quoting=csv.QUOTE_NONNUMERIC` — передали встроенную константу, указывающую функции, что числа без кавычек необходимо преобразовать к типу `float`.

• Запись CSV

Для записи данных в файл используют функцию `writer`, которая возвращает объект конвертирующий данные в формат CSV. Функция `writer` принимает файловый дескриптор и дополнительные параметры записи аналогичные параметрам функции `reader`. При этом данные в файл не записываются пока у возвращённого объекта не будет вызван метод `writerow` для записи одной строки или `writerows` для записи нескольких строк.

Рассмотри на примере.

```
import csv

with (
    open('biostats_tab.csv', 'r', newline='') as f_read,
    open('new_biostats.csv', 'w', newline='', encoding='utf-8')
as f_write
):
    csv_read = csv.reader(f_read, dialect='excel-tab',
quoting=csv.QUOTE_NONNUMERIC)
    csv_write = csv.writer(f_write, dialect='excel', delimiter='
', quotechar='|', quoting=csv.QUOTE_MINIMAL)
    all_data = []
    for i, line in enumerate(csv_read):
        if i == 0:
            csv_write.writerow(line)
        else:
            line[2] += 1
            for j in range(2, 4 + 1):
                line[j] = int(line[j])
            all_data.append(line)
    csv_write.writerows(all_data)
```

1. Используя менеджер контекста `with` открыли два файла. Из первого читаем информацию, а второй создаём для записи.
2. Функция `reader` возвращает объект `csv_read` для чтения как в пример выше.
3. Функция `writer` возвращает объект `csv_write` для записи. Мы указали:
 - a. диалект “excel”
 - b. в качестве разделителя столбцов будем использовать пробел

- c. если символ разделитель (пробел) есть в данных, экранируем их вертикальной чертой
 - d. символ экранирования используем по минимуму, только там где он необходим для разрешения конфликта с разделителем
4. В цикле читаем все строки из исходного файл. При этом строку с заголовком сразу записываем методом `writerow`.
5. Для остальных строк увеличиваем возраст на единицу, преобразуем вещественные числа в целые и сохраняем список в матрицу `all_data`
6. Одним запросом `writerows(all_data)` сохраняем матрицу в файл.

• Чтение в словарь

Помимо сохранения таблицы в список можно использовать для хранения словарь. Ключи словаря — названия столбцов, значения — очередная строка файла CSV. Прочитаем файл `biostats_tab.csv` из примера выше, но не в список, а в словарь.

Воспользуемся классом `DictReader`.

```
import csv

with open('biostats_tab.csv', 'r', newline='') as f:
    csv_file = csv.DictReader(f, fieldnames=["name", "sex",
"age", "height", "weight", "office"],
                                restkey="new", restval="Main
Office", dialect='excel-tab', quoting=csv.QUOTE_NONNUMERIC)
    for line in csv_file:
        print(f'{line = }')
        print(f'{line["name"] = }\t{line["age"] = }')
```

Если передать список строк в параметр `fieldnames`, они будут использоваться для ключей словаря, а не первая строка файла. В нашем примере передан “лишний” ключ `count`. Т.к. в таблице нет шестого столбца, ему было присвоено значение из параметра `restval`.

```
import csv

with open('biostats_tab.csv', 'r', newline='') as f:
    csv_file = csv.DictReader(f, fieldnames=["name", "sex",
"age", ], restkey="new", restval="Main Office",
                                dialect='excel-tab',
quoting=csv.QUOTE_NONNUMERIC)
    for line in csv_file:
        print(f'{line = }')
        print(f'{line["name"] = }\t{line["age"] = }')
```

Если количество ключей оказывается меньше, чем столбцов, недостающий ключ берётся из параметра `restkey`. При этом все столбцы без ключа сохраняются как элементы списка в `restkey` ключ.

- **Запись из словаря**

Для записи содержимого словаря в CSV используют класс `DictWriter`. Его параметры схожи с рассмотренными выше параметрами `DictReader`.

```
import csv
from typing import Iterator

with (
    open('biostats_tab.csv', 'r', newline='') as f_read,
    open('biostats_new.csv', 'w', newline='', encoding='utf-8')
as f_write
):
    csv_read: Iterator[dict] = csv.DictReader(f_read,
fieldnames=["name", "sex", "age", "height", "weight", "office"],
                                                    restval="Main
Office", dialect='excel-tab', quoting=csv.QUOTE_NONNUMERIC)
    csv_write = csv.DictWriter(f_write, fieldnames=["id", "name",
"office", "sex", "age", "height", "weight"],
                                                    dialect='excel-tab',
quoting=csv.QUOTE_ALL)
    csv_write.writeheader()
    all_data = []
    for i, dict_row in enumerate(csv_read):
        if i != 0:
            dict_row['id'] = i
            dict_row['age'] += 1
            all_data.append(dict_row)
    csv_write.writerows(all_data)
```

Класс `DictWriter` получил список полей для записи, где добавлено новое поле `id`. В качестве диалекта выбран `excel` с табуляцией. В параметре `quoting` указали, что все значения стоит заключать в кавычки.

Новый для нас метод `writeheader` сохранил первую строку с заголовками в том порядке, в котором мы их перечислили в параметре `fieldnames`. Далее мы работаем с элементами словаря и формируем список словарей для одноразовой записи в файл.



Важно! Обратите внимание на импорт объекта `Iterator` из модуля `typing`. При написании кода IDE подсвечивала возможные ошибки, так как не понимала что за объект `csv_read`. Запись `csv_read: Iterator[dict] = ...` сообщает, что мы используем объект итератор, который возвращает словари. После уточнения типа IDE исключила подсветку “ошибок”.

Задание

Перед вами несколько строк кода. Что будет записано в файл после его выполнения? У вас три минуты.

```
import csv

with open('quest.csv', 'w', newline='', encoding='utf-8') as f_write:
    csv_write = csv.DictWriter(f_write, fieldnames=["number",
"number", "data"], restval='Hello world!', dialect='excel',
delimiter='#', quotechar='=', quoting=csv.QUOTE_NONNUMERIC)
    csv_write.writeheader()
    dict_row = {}
    for i in range(10):
        dict_row['number'] = i
        dict_row['name'] = str(i)
        csv_write.writerow(dict_row)
```

3. Pickle

Рассмотренные выше сериализаторы универсальны. Они не привязаны к конкретному языку программирования. А следовательно прочитать и понять файл сможет любой разработчик. Однако сложные структуры данных Python не всегда возможно сохранить в таблице CSV или JSON объекте.

Python предлагает модуль `pickle` для сериализации и десериализации своих структур в поток байт. Преобразования возможны в любом месте и в любое время, если вы используете Python. Но данные окажутся бесполезными, если вы передаёте их для обработки другим ЯП.



Крайне важно! Модуль `pickle` не занимается проверкой потока байт на безопасность перед распаковкой. Не используйте его с тем набором байт, безопасность которого не можете гарантировать.

```
import pickle

res = pickle.loads(b"cos\nsystem\n(S'echo Hello world!'\ntr.")
print(res)
```

В этом безобидном примере модуль получил доступ к консоли и вывел сообщение “Привет, мир!” прежде чем десериализовать поток байт в число ноль.

Допустимые типы данных для преобразования

Модуль `pickle` может обработать следующие структуры Python:

- `None`, `True` и `False`;
- `int`, `float`, `complex`;
- `str`, `bytes`, `bytearrays`;
- `tuple`, `list`, `set`, `dict` если они содержат объекты, обрабатываемые `pickle`;
- встроенные функции и функции созданные разработчиком и доступные из верхнего уровня модуля, кроме `lambda` функций;
- классы доступные из верхнего уровня модуля;
- экземпляры классов, если `pickle` смог обработать их дандер `__dict__` или результат вызова метода `__getstate__()`.

Список достаточно большой, чтобы позволить сериализовывать большую часть Python структур.

Сериализация

Преобразуем словарь из главы про JSON в набор байт средствами модуля `pickle`.

```
import pickle
```

```

my_dict = {
    "first_name": "Джон",
    "last_name": "Смит",
    "hobbies": ["кузнечное дело", "программирование",
"путешествия"],
    "age": 35,
    "children": [
        {
            "first_name": "Алиса",
            "age": 5
        },
        {
            "first_name": "Маруся",
            "age": 3
        }
    ]
}
print(my_dict)
res = pickle.dumps(my_dict, protocol=pickle.DEFAULT_PROTOCOL)
print(f'{res = }')

```

Вызываем функцию `dumps` для преобразования всей структуры в строку байт. Отдельно указали протокол по умолчанию. Модуль `pickle` имеет несколько протоколов, который не гарантируют совместимость с более старыми версиями. Сейчас протоколом по умолчанию является версия 4. Она появилась в Python 3.4, а стала дефолтным протоколом с Python 3.8. Если вы не занимаетесь поддержкой старого кода, можно смело использовать четвёртую версию протокола. Попробуем сохранить объекты, неподдерживаемые JSON в бинарный файл.

```

import pickle

def func(a, b, c):
    return a + b + c

my_dict = {
    "numbers": [42, 4.1415, 7+3j],
    "functions": (func, sum, max),
    "others": {True, False, 'Hello world!'},
}

with open('my_dict.pickle', 'wb') as f:
    pickle.dump(my_dict, f)

```

Функция `dump` принимает на вход объект для сериализации и файловый дескриптор. При этом файл должен быть открыт для записи как бинарный. В целом мы можем заменить файл на любой объект потоковой записи, который реализует метод `write`.

Обратите внимание на то, что мы добавили в словарь функцию `func`, которая принимает на вход три значения и возвращает их сумму. К ней мы вернемся ниже.

Десериализация

Проведём обратные операции. Набор байт мы уже восстанавливали, когда разбирались с безопасностью данных. Уточним детали.

```
import pickle

data = b'\x80\x04\x95\xe3\x00\x00\x00\x00\x00\x00\x00}\x94(\x8c\nfirst_name\x94\x8c\x08\xd0\x94\xd0\xb6\xd0\xbe\xd0xbd\x94\x8c\tlast_name\x94\x8c\x08\xd0\xa1\xd0xbc\xd0\xb8\xd1\x82\x94\x8c\x07hobbies\x94]\x94(\x8c\x1b\xd0\xba\xd1\x83\xd0\xb7\xd0\xbd\xd0\xb5\xd1\x87\xd0\xbd\xd0\xbe\xd0\xb5 \xd0\xb4\xd0\xb5\xd0\xbb\xd0\xbe\x94\x8c\xd0\xbf\xd1\x80\xd0\xbe\xd0\xb3\xd1\x80\xd0\xb0\xd0xbc\xd0\xbc\xd0\xb8\xd1\x80\xd0\xbe\xd0\xb2\xd0\xb0\xd0\xbd\xd0\xb8\xd0\xb5\x94\x8c\x16\xd0\xbf\xd1\x83\xd1\x82\xd0\xb5\xd1\x88\xd0\xb5\xd1\x81\xd1\x82\xd0\xb2\xd0\xb8\xd1\x8f\x94e\x8c\x03age\x94K#\x8c\x08children\x94]\x94(\x94(h\x01\x8c\n\xd0\x90\xd0\xbb\xd0\xb8\xd1\x81\xd0\xb0\x94h\nK\x05u}\x94(h\x01\x8c\x0c\xd0\x9c\xd0\xb0\xd1\x80\xd1\x83\xd1\x81\xd1\x8f\x94h\nK\x03ueu.'
```

```
new_dict = pickle.loads(data)
print(f'{new_dict = }')
```

Функция получила на вход набор байт и восстановила из них исходный словарь. Уточним, что `loads` имеет ряд дополнительных параметров: `fix_imports=True`, `encoding='ASCII'`, `errors='strict'`. Они нужны для десериализации объектов созданных в Python 2. А так как поддержка второй версии Python завершена в 2020 году, нет смысла разбирать назначение параметров.

В финале загрузим данные из файла `my_dict.pickle`, который создали ранее.

```
import pickle
```

```
def func(a, b, c):
    return a * b * c

with open('my_dict.pickle', 'rb') as f:
    new_dict = pickle.load(f)
print(f'{new_dict = }')
print(f'{new_dict["functions"][0](2, 3, 4) = }')
```

Содержимое словаря в точности соответствует исходному. Но есть одно но. При вызове функции func, которая лежит в нулевой ячейке кортежа по ключу functions мы получили не сумму трёх чисел, а произведение. В файле, где произведена десериализация есть функция func, которая умножает числа. Модуль pickle указал в словаре её, а не исходную. Более того, если бы функции с нужным именем не было, десериализация завершилась бы ошибкой.

Задание

Перед вами несколько строк кода. Что будет записано в файл после его выполнения? У вас три минуты.

```
import pickle

my_dict = {'numbers': [42, 4.1415, (7 + 3j)],
           'functions': (sum, max),
           'others': {False, True, 'Hello world!'}}

res = pickle.dumps(my_dict)
with open('quest.pickle', 'wb') as f:
    pickle.dump(f'{res = }', f)
```

Вывод

На этой лекции мы:

1. Разобрались в сериализации и десериализации данных
2. Изучили самый популярный формат сериализации — JSON
3. Узнали о чтении и записи таблиц в формате CSV
4. Разобрались с внутренним сериализатором Python — модулем pickle

Краткий анонс следующей лекции

1. Разберём замыкания в программировании
2. Изучим возможности Python по созданию декораторов
3. Узнаем как создавать декораторы с параметрами
4. Разберём работу некоторых декораторов из модуля functools

Домашнее задание

Возьмите код прошлых уроков и попытайтесь сериализовать используемые в нём объекты на основе знаний с сегодняшнего занятия. Попробуйте все варианты сериализации и десериализации в разных форматах.