PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

СЕРИАЛИЗАЦИЯ ДАННЫХ

СЕРИАЛИЗАЦИЯ ДАННЫХ

Сериализация данных - это сохранение данных в каком-то формате. Чаще всего, это сохранение в каком-то структурированном формате.

Например, это могут быть:

- файлы в формате YAML или JSON
- файлы в формате CSV
- база данных

СЕРИАЛИЗАЦИЯ ДАННЫХ

Для чего могут пригодится форматы YAML, JSON, CSV:

- у вас могут быть данные о IP-адресах и подобной информации, которую нужно обработать, в таблицах
  - таблицу можно экспортировать в формат CSV и обрабатывать её с помощью Python
- управляющий софт может возвращать данные в JSON. Соответственно, преобразовав эти данные в объект Python, с ними можно работать и делать, что угодно
- YAML очень удобно использовать для описания параметров
  - например, это могут быть параметры настройки различных объектов (IP-адреса, VLAN и др)

РАБОТА С ФАЙЛАМИ В ФОРМАТЕ CSV

РАБОТА С ФАЙЛАМИ В ФОРМАТЕ 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.

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.

РАБОТА С ФАЙЛАМИ В ФОРМАТЕ CSV

Пример файла в формате CSV (sw_data.csv):

```
hostname,vendor,model,location
sw1,Cisco,3750,London
sw2,Cisco,3850,Liverpool
sw3,Cisco,3650,Liverpool
sw4,Cisco,3650,London
```

В стандартной библиотеке Python есть модуль csv, который позволяет работать с файлами в CSV формате.

ЧТЕНИЕ ФАЙЛОВ В ФОРМАТЕ CSV

Пример использования модуля csv (файл csv_read.py):

```
import csv

with open('sw_data.csv') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Вывод будет таким:

```
$ python csv_read.py
['hostname', 'vendor', 'model', 'location']
['sw1', 'Cisco', '3750', 'London']
['sw2', 'Cisco', '3850', 'Liverpool']
['sw3', 'Cisco', '3650', 'Liverpool']
['sw4', 'Cisco', '3650', 'London']
```

ЧТЕНИЕ ФАЙЛОВ В ФОРМАТЕ CSV

reader - это итератор:

```
In [1]: import csv

In [2]: with open('sw_data.csv') as f:
   ...:     reader = csv.reader(f)
   ...:     print reader
   ...:
<_csv.reader object at 0x10385b050>
```

ЧТЕНИЕ ФАЙЛОВ В ФОРМАТЕ CSV

Заголовки столбцов удобней получить отдельным объектом (файл csv_read_headers.py):

```
import csv

with open('sw_data.csv') as f:
    reader = csv.reader(f)
    headers = next(reader)
    print('Headers: ', headers)
    for row in reader:
        print(row)
```

ЧТЕНИЕ ФАЙЛОВ В ФОРМАТЕ CSV

DictReader позволяет получить словари, в которых ключи - это названия столбцов, а значения - значения столбцов (файл csv_read_dict.py):

```
import csv

with open('sw_data.csv') as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)
        print(row['hostname'], row['model'])
```

ЧТЕНИЕ ФАЙЛОВ В ФОРМАТЕ CSV

Вывод будет таким:

```
$ python csv_read_dict.py
OrderedDict([('hostname', 'sw1'), ('vendor', 'Cisco'), ('model', '3750'), ('location', 'London')])
sw1 3750
OrderedDict([('hostname', 'sw2'), ('vendor', 'Cisco'), ('model', '3850'), ('location', 'Liverpool')])
sw2 3850
OrderedDict([('hostname', 'sw3'), ('vendor', 'Cisco'), ('model', '3650'), ('location', 'Liverpool')])
sw3 3650
OrderedDict([('hostname', 'sw4'), ('vendor', 'Cisco'), ('model', '3650'), ('location', 'London')])
sw4 3650
```

ЗАПИСЬ ФАЙЛОВ В ФОРМАТЕ CSV

Аналогичным образом, с помощью модуля csv, можно и записать файл в формате CSV (файл csv_write.py):

```
import csv

data = [['hostname', 'vendor', 'model', 'location'],
        ['sw1', 'Cisco', '3750', 'London, Best str'],
        ['sw2', 'Cisco', '3850', 'Liverpool, Better str'],
        ['sw3', 'Cisco', '3650', 'Liverpool, Better str'],
        ['sw4', 'Cisco', '3650', 'London, Best str']]

with open('sw_data_new.csv', 'w') as f:
    writer = csv.writer(f)
    for row in data:
        writer.writerow(row)

with open('sw_data_new.csv') as f:
    print(f.read())
```

ЗАПИСЬ ФАЙЛОВ В ФОРМАТЕ CSV

Вывод будет таким:

```
$ python csv_write.py
hostname,vendor,model,location
sw1,Cisco,3750,"London, Best str"
sw2,Cisco,3850,"Liverpool, Better str"
sw3,Cisco,3650,"Liverpool, Better str"
sw4,Cisco,3650,"London, Best str"
```

Обратите внимание: последнее значение, взято в кавычки, а остальные строки - нет.

ЗАПИСЬ ФАЙЛОВ В ФОРМАТЕ CSV

Для того, чтобы все строки записывались в файл csv с кавычками, надо изменить скрипт таким образом (файл csv_write_quoting.py):

```
import csv

data = [['hostname', 'vendor', 'model', 'location'],
        ['sw1', 'Cisco', '3750', 'London, Best str'],
        ['sw2', 'Cisco', '3850', 'Liverpool, Better str'],
        ['sw3', 'Cisco', '3650', 'Liverpool, Better str'],
        ['sw4', 'Cisco', '3650', 'London, Best str']]

with open('sw_data_new.csv', 'w') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC)
    for row in data:
        writer.writerow(row)

with open('sw_data_new.csv') as f:
    print(f.read())
```

ЗАПИСЬ ФАЙЛОВ В ФОРМАТЕ CSV

Теперь вывод будет таким:

```
$ python csv_write_quoting.py
"hostname","vendor","model","location"
"sw1","Cisco","3750","London, Best str"
"sw2","Cisco","3850","Liverpool, Better str"
"sw3","Cisco","3650","Liverpool, Better str"
"sw4","Cisco","3650","London, Best str"
```

Теперь все значения с кавычками. И, так как номер модели задан как строка, в изначальном списке, тут он тоже в кавычках.

ЗАПИСЬ ФАЙЛОВ В ФОРМАТЕ CSV

Кроме метода writerow, поддерживается метод writerows (файл csv_writerows.py):

```
import csv

data = [['hostname', 'vendor', 'model', 'location'],
        ['sw1', 'Cisco', '3750', 'London, Best str'],
        ['sw2', 'Cisco', '3850', 'Liverpool, Better str'],
        ['sw3', 'Cisco', '3650', 'Liverpool, Better str'],
        ['sw4', 'Cisco', '3650', 'London, Best str']]

with open('sw_data_new.csv', 'w') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC)
    writer.writerows(data)

with open('sw_data_new.csv') as f:
    print(f.read())
```

DICTWRITER

С помощью DictWriter можно записать словари в формат csv.

В целом DictWriter работает так же, как writer, но так как словари не упорядочены, надо указывать явно в каком порядке будут идти столбцы в файле. Для этого используется параметр fieldnames.

DICTWRITER

Файл csv_write_dict.py:

```
import csv

data = [{'hostname': 'sw1',
         'location': 'London',
         'model': '3750',
         'vendor': 'Cisco'},
        {'hostname': 'sw2',
         'location': 'Liverpool',
         'model': '3850',
         'vendor': 'Cisco'},
        {'hostname': 'sw3',
         'location': 'Liverpool',
         'model': '3650',
         'vendor': 'Cisco'},
        {'hostname': 'sw4',
         'location': 'London',
         'model': '3650',
         'vendor': 'Cisco'}]

with open('csv_write_dictwriter.csv', 'w') as f:
    writer = csv.DictWriter(f, fieldnames=list(data[0].keys()),
                            quoting=csv.QUOTE_NONNUMERIC)
    writer.writeheader()
    for d in data:
        writer.writerow(d)
```

УКАЗАНИЕ РАЗДЕЛИТЕЛЯ

Например, если в файле используется разделитель ; (файл sw_data2.csv):

```
hostname;vendor;model;location
sw1;Cisco;3750;London
sw2;Cisco;3850;Liverpool
sw3;Cisco;3650;Liverpool
sw4;Cisco;3650;London
```

Достаточно просто указать какой разделитель используется в reader (файл csv_read_delimiter.py):

```
import csv

with open('sw_data2.csv') as f:
    reader = csv.reader(f, delimiter=';')
    for row in reader:
        print(row)
```

РАБОТА С ФАЙЛАМИ В ФОРМАТЕ JSON

РАБОТА С ФАЙЛАМИ В ФОРМАТЕ JSON

JSON (JavaScript Object Notation) - это текстовый формат для хранения и обмена данными.

JSON по синтаксису очень похож на Python. И достаточно удобен для восприятия.

Как и в случае с CSV, в Python есть модуль, который позволяет легко записывать и читать данные в формате JSON.

ЧТЕНИЕ

Файл sw_templates.json:

```
{
  "access": [
    "switchport mode access",
    "switchport access vlan",
    "switchport nonegotiate",
    "spanning-tree portfast",
    "spanning-tree bpduguard enable"
  ],
  "trunk": [
    "switchport trunk encapsulation dot1q",
    "switchport mode trunk",
    "switchport trunk native vlan 999",
    "switchport trunk allowed vlan"
  ]
}
```

ЧТЕНИЕ. JSON.LOAD()

Чтение файла в формате JSON в объект Python (файл json_read_load.py):

```
import json

with open('sw_templates.json') as f:
    templates = json.load(f)

for section, commands in templates.items():
    print(section)
    print('\n'.join(commands))
```

ЧТЕНИЕ. JSON.LOAD()

Вывод будет таким:

```
$ python json_read_load.py
{'access': ['switchport mode access', 'switchport access vlan', 'switchport nonegotiate', 'spanning-tree por
access
switchport mode access
switchport access vlan
switchport nonegotiate
spanning-tree portfast
spanning-tree bpduguard enable
trunk
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk native vlan 999
switchport trunk allowed vlan
```

ЧТЕНИЕ. JSON.LOADS()

Считывание строки в формате JSON в объект Python (файл json_read_loads.py):

```
import json

with open('sw_templates.json') as f:
    file_content = f.read()
    templates = json.loads(file_content)

print(templates)

for section, commands in templates.items():
    print(section)
    print('\n'.join(commands))
```

ЗАПИСЬ. JSON.DUMPS()

Преобразование объекта в строку в формате JSON (json_write_dumps.py):

```
import json


trunk_template = ['switchport trunk encapsulation dot1q',
                  'switchport mode trunk',
                  'switchport trunk native vlan 999',
                  'switchport trunk allowed vlan']

access_template = ['switchport mode access',
                   'switchport access vlan',
                   'switchport nonegotiate',
                   'spanning-tree portfast',
                   'spanning-tree bpduguard enable']

to_json = {'trunk':trunk_template, 'access':access_template}

with open('sw_templates.json', 'w') as f:
    f.write(json.dumps(to_json))

with open('sw_templates.json') as f:
    print(f.read())
```

ЗАПИСЬ. JSON.DUMP()

Запись объекта Python в файл в формате JSON (файл json_write_dump.py):

```
import json


trunk_template = ['switchport trunk encapsulation dot1q',
                  'switchport mode trunk',
                  'switchport trunk native vlan 999',
                  'switchport trunk allowed vlan']

access_template = ['switchport mode access',
                   'switchport access vlan',
                   'switchport nonegotiate',
                   'spanning-tree portfast',
                   'spanning-tree bpduguard enable']

to_json = {'trunk':trunk_template, 'access':access_template}

with open('sw_templates.json', 'w') as f:
    json.dump(to_json, f)

with open('sw_templates.json') as f:
    print(f.read())
```

ЗАПИСЬ

Более удобный для чтения вывод (файл json_write_indent.py):

```
import json


trunk_template = ['switchport trunk encapsulation dot1q',
                  'switchport mode trunk',
                  'switchport trunk native vlan 999',
                  'switchport trunk allowed vlan']

access_template = ['switchport mode access',
                   'switchport access vlan',
                   'switchport nonegotiate',
                   'spanning-tree portfast',
                   'spanning-tree bpduguard enable']

to_json = {'trunk':trunk_template, 'access':access_template}

with open('sw_templates.json', 'w') as f:
    json.dump(to_json, f, sort_keys=True, indent=2)

with open('sw_templates.json') as f:
    print(f.read())
```

ЗАПИСЬ

Теперь содержимое файла sw_templates.json выглядит так:

```
{
  "access": [
    "switchport mode access",
    "switchport access vlan",
    "switchport nonegotiate",
    "spanning-tree portfast",
    "spanning-tree bpduguard enable"
  ],
  "trunk": [
    "switchport trunk encapsulation dot1q",
    "switchport mode trunk",
    "switchport trunk native vlan 999",
    "switchport trunk allowed vlan"
  ]
}
```

ИЗМЕНЕНИЕ ТИПА ДАННЫХ

При работе с форматом json, данные не всегда будут того же типа, что исходные данные в Python.

Например, кортежи, при записи в JSON, превращаются в списки:

```
In [1]: import json

In [2]: trunk_template = ('switchport trunk encapsulation dot1q',
   ...:                   'switchport mode trunk',
   ...:                   'switchport trunk native vlan 999',
   ...:                   'switchport trunk allowed vlan')

In [3]: print type(trunk_template)
<type 'tuple'>

In [4]: with open('trunk_template.json', 'w') as f:
   ...:     json.dump(trunk_template, f, sort_keys=True, indent=2)
   ...:
```

ИЗМЕНЕНИЕ ТИПА ДАННЫХ

```
In [5]: cat trunk_template.json
[
  "switchport trunk encapsulation dot1q",
  "switchport mode trunk",
  "switchport trunk native vlan 999",
  "switchport trunk allowed vlan"
]
In [6]: templates = json.load(open('trunk_template.json'))

In [7]: type(templates)
Out[7]: list

In [8]: print(templates)
['switchport trunk encapsulation dot1q', 'switchport mode trunk', 'switchport trunk native vlan 999', 'switc
```

КОНВЕРТАЦИЯ ДАННЫХ PYTHON В JSON

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

КОНВЕРТАЦИЯ JSON В ДАННЫЕ PYTHON

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

КЛЮЧИ СЛОВАРЕЙ

В формат JSON нельзя записать словарь у которого ключи - кортежи:

```
In [9]: to_json = {('trunk', 'cisco'):trunk_template, 'access':access_template}

In [10]: with open('sw_templates.json', 'w') as f:
    ...:     json.dump(to_json, f)
    ...:
...
TypeError: key ('trunk', 'cisco') is not a string
```

КЛЮЧИ СЛОВАРЕЙ

Специальный параметр позволяет игнорировать такие ключи:

```
In [11]: with open('sw_templates.json', 'w') as f:
    ...:     json.dump(to_json, f, skipkeys=True)
    ...:
    ...:

In [12]: cat sw_templates.json
{"access": ["switchport mode access", "switchport access vlan", "switchport nonegotiate",
"spanning-tree portfast", "spanning-tree bpduguard enable"]}
```

РАБОТА С ФАЙЛАМИ В ФОРМАТЕ YAML

РАБОТА С ФАЙЛАМИ В ФОРМАТЕ YAML

YAML (YAML Ain't Markup Language) - еще один текстовый формат для записи данных.

YAML более приятен для восприятия человеком, чем JSON, поэтому его часто используют для описания сценариев в ПО. Например, в Ansible.

СИНТАКСИС YAML

СИНТАКСИС YAML

Как и Python, YAML использует отступы для указания структуры документа. Но в YAML можно использовать только пробелы и нельзя использовать знаки табуляции.

Еще одна схожесть с Python: комментарии начинаются с символа # и продолжаются до конца строки.

СПИСОК

Список может быть записан в одну строку:

```
[switchport mode access, switchport access vlan, switchport nonegotiate, spanning-tree portfast, spanning-tr
```

Или каждый элемент списка в своей строке:

```
- switchport mode access
- switchport access vlan
- switchport nonegotiate
- spanning-tree portfast
- spanning-tree bpduguard enable
```

Когда список записан таким блоком, каждая строка должна начинаться с - (минуса и пробела). И все строки в списке должны быть на одном уровне отступа.

СЛОВАРЬ

Словарь также может быть записан в одну строку:

```
{ vlan: 100, name: IT }
```

Или блоком:

```
vlan: 100
name: IT
```

СТРОКИ

Строки в YAML не обязательно брать в кавычки. Это удобно, но иногда всё же следует использовать кавычки. Например, когда в строке используется какой-то специальный символ (специальный для YAML).

Такую строку, например, нужно взять в кавычки, чтобы она была корректно воспринята YAML:

```
command: "sh interface | include Queueing strategy:"
```

КОМБИНАЦИЯ ЭЛЕМЕНТОВ

Словарь, в котором есть два ключа: access и trunk. Значения, которые соответствуют этим ключам - списки команд:

```
access:
- switchport mode access
- switchport access vlan
- switchport nonegotiate
- spanning-tree portfast
- spanning-tree bpduguard enable

trunk:
- switchport trunk encapsulation dot1q
- switchport mode trunk
- switchport trunk native vlan 999
- switchport trunk allowed vlan
```

КОМБИНАЦИЯ ЭЛЕМЕНТОВ

Список словарей:

```
- BS: 1550
  IT: 791
  id: 11
  name: Liverpool
  to_id: 1
  to_name: LONDON
- BS: 1510
  IT: 793
  id: 12
  name: Bristol
  to_id: 1
  to_name: LONDON
- BS: 1650
  IT: 892
  id: 14
  name: Coventry
  to_id: 2
  to_name: Manchester
```

МОДУЛЬ PYYAML

МОДУЛЬ PYYAML

Для работы с YAML в Python используется модуль PyYAML. Он не входит в стандартную библиотеку модулей, поэтому его нужно установить:

```
pip install pyyaml
```

Работа с ним аналогична модулям csv и json.

ЧТЕНИЕ ИЗ YAML

Файл info.yaml:

```yaml
- BS: 1550
  IT: 791
  id: 11
  name: Liverpool
  to_id: 1
  to_name: LONDON
- BS: 1510
  IT: 793
  id: 12
  name: Bristol
  to_id: 1
  to_name: LONDON
- BS: 1650
  IT: 892
  id: 14
  name: Coventry
  to_id: 2
  to_name: Manchester
```

ЧТЕНИЕ ИЗ YAML

Чтение из YAML (файл yaml_read.py):

```
import yaml
import pprint

with open('info.yaml') as f:
    templates = yaml.load(f)

pprint.pprint(templates)
```

ЧТЕНИЕ ИЗ YAML

Результат:

```
$ python yaml_read.py
[{'BS': 1550,
  'IT': 791,
  'id': 11,
  'name': 'Liverpool',
  'to_id': 1,
  'to_name': 'LONDON'},
 {'BS': 1510,
  'IT': 793,
  'id': 12,
  'name': 'Bristol',
  'to_id': 1,
  'to_name': 'LONDON'},
 {'BS': 1650,
  'IT': 892,
  'id': 14,
  'name': 'Coventry',
  'to_id': 2,
  'to_name': 'Manchester'}]
```

ЗАПИСЬ В YAML

Запись объектов Python в YAML (файл yaml_write.py):

```
import yaml

trunk_template = ['switchport trunk encapsulation dot1q',
                  'switchport mode trunk',
                  'switchport trunk native vlan 999',
                  'switchport trunk allowed vlan']

access_template = ['switchport mode access',
                   'switchport access vlan',
                   'switchport nonegotiate',
                   'spanning-tree portfast',
                   'spanning-tree bpduguard enable']

to_yaml = {'trunk':trunk_template, 'access':access_template}

with open('sw_templates.yaml', 'w') as f:
    yaml.dump(to_yaml, f)

with open('sw_templates.yaml') as f:
    print(f.read())
```

ЗАПИСЬ В YAML

Файл sw_templates.yaml выглядит таким образом:

```
access: [switchport mode access, switchport access vlan, switchport nonegotiate, spanning-tree
    portfast, spanning-tree bpduguard enable]
trunk: [switchport trunk encapsulation dot1q, switchport mode trunk, switchport trunk
    native vlan 999, switchport trunk allowed vlan]
```

ЗАПИСЬ В YAML

Параметр default_flow_style=False (файл yaml_write_default_flow_style.py):

```
import yaml

trunk_template = ['switchport trunk encapsulation dot1q',
                  'switchport mode trunk',
                  'switchport trunk native vlan 999',
                  'switchport trunk allowed vlan']

access_template = ['switchport mode access',
                   'switchport access vlan',
                   'switchport nonegotiate',
                   'spanning-tree portfast',
                   'spanning-tree bpduguard enable']

to_yaml = {'trunk':trunk_template, 'access':access_template}

with open('sw_templates.yaml', 'w') as f:
    yaml.dump(to_yaml, f, default_flow_style=False)

with open('sw_templates.yaml') as f:
    print f.read()
```

ЗАПИСЬ В YAML

Теперь содержимое файла sw_templates.yaml выглядит таким образом:

```
access:
- switchport mode access
- switchport access vlan
- switchport nonegotiate
- spanning-tree portfast
- spanning-tree bpduguard enable
trunk:
- switchport trunk encapsulation dot1q
- switchport mode trunk
- switchport trunk native vlan 999
- switchport trunk allowed vlan
```