PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

РАСПАКОВКА ПЕРЕМЕННЫХ

РАСПАКОВКА ПЕРЕМЕННЫХ

Распаковка переменных - это специальный синтаксис, который позволяет присваивать переменным элементы итерируемого объекта.

> *Достаточно часто этот функционал встречается под именем tuple unpacking. Но распаковка работает на любом итерируемом объекте, не только с кортежами*

РАСПАКОВКА ПЕРЕМЕННЫХ

Пример распаковки переменных:

```
In [1]: interface = ['FastEthernet0/1', '10.1.1.1', 'up', 'up']

In [2]: intf, ip, status, protocol = interface

In [3]: intf
Out[3]: 'FastEthernet0/1'

In [4]: ip
Out[4]: '10.1.1.1'
```

Такой вариант намного удобней использовать, чем использование индексов:

```
In [5]: intf, ip, status, protocol = interface[0], interface[1], interface[2], interface[3]
```

РАСПАКОВКА ПЕРЕМЕННЫХ

При распаковке переменных каждый элемент списка попадает в соответствующую переменную. Но важно учитывать, что переменных слева должно быть ровно столько, сколько элементов в списке.

Если переменных больше или меньше, возникнет исключение:

```
In [6]: intf, ip, status = interface
---------------------------------------------------------
ValueError                Traceback (most recent call last)
<ipython-input-11-a304c4372b1a> in <module>()
----> 1 intf, ip, status = interface

ValueError: too many values to unpack (expected 3)

In [7]: intf, ip, status, protocol, other = interface
---------------------------------------------------------
ValueError                Traceback (most recent call last)
<ipython-input-12-ac93e78b978c> in <module>()
----> 1 intf, ip, status, protocol, other = interface

ValueError: not enough values to unpack (expected 5, got 4)
```

ЗАМЕНА НЕНУЖНЫХ ЭЛЕМЕНТОВ _

Достаточно часто из всех элементов итерируемого объекта нужны только некоторые. Но выше был пример того, что синтаксис распаковки требует указать ровно столько переменных, сколько элементов в итерируемом объекте.

Если, например, из строки line надо получить только VLAN, MAC и интерфейс, надо все равно указать переменную для типа записи:

```
In [8]: line = '100    01bb.c580.7000    DYNAMIC     Gi0/1'

In [9]: vlan, mac, item_type, intf = line.split()

In [10]: vlan
Out[10]: '100'

In [11]: intf
Out[11]: 'Gi0/1'
```

ЗАМЕНА НЕНУЖНЫХ ЭЛЕМЕНТОВ _

Но, если тип записи не нужен в дальнейшем, можно заменить переменную item_type нижним подчеркиванием:

```
In [12]: vlan, mac, _, intf = line.split()
```

Таким образом явно указывается то, что этот элемент не нужен.

ЗАМЕНА НЕНУЖНЫХ ЭЛЕМЕНТОВ _

Нижнее подчеркивание можно использовать и несколько раз:

```
In [13]: dhcp = '00:09:BB:3D:D6:58   10.1.10.2        86250       dhcp-snooping   10    FastEthernet0/1'

In [14]: mac, ip, _, _, vlan, intf = dhcp.split()

In [15]: mac
Out[15]: '00:09:BB:3D:D6:58'

In [16]: vlan
Out[16]: '10'
```

ИСПОЛЬЗОВАНИЕ *

Распаковка переменных поддерживает специальный синтаксис, который позволяет распаковывать несколько элементов в один. Если поставить * перед именем переменной, в нее запишутся все элементы, кроме тех, что присвоены явно.

ИСПОЛЬЗОВАНИЕ *

Например, так можно получить первый элемент в переменную first, а остальные в rest:

```
In [18]: vlans = [10, 11, 13, 30]

In [19]: first, *rest = vlans

In [20]: first
Out[20]: 10

In [21]: rest
Out[21]: [11, 13, 30]
```

ИСПОЛЬЗОВАНИЕ *

При этом переменная со звездочкой всегда будет содержать список:

```
In [22]: vlans = (10, 11, 13, 30)

In [22]: first, *rest = vlans

In [23]: first
Out[23]: 10

In [24]: rest
Out[24]: [11, 13, 30]
```

ИСПОЛЬЗОВАНИЕ *

Если элемент всего один, распаковка все равно отработает:

```
In [25]: first, *rest = vlans

In [26]: first
Out[26]: 55

In [27]: rest
Out[27]: []
```

ИСПОЛЬЗОВАНИЕ *

Такая переменная со звездочкой в выражении распаковки может быть только одна.

```
In [28]: vlans = (10, 11, 13, 30)

In [29]: first, *rest, *others = vlans
  File "<ipython-input-37-dedf7a08933a>", line 1
    first, *rest, *others = vlans
                            ^
SyntaxError: two starred expressions in assignment
```

ИСПОЛЬЗОВАНИЕ *

Такая переменная может находиться не только в конце выражения:

```
In [30]: vlans = (10, 11, 13, 30)

In [31]: *rest, last = vlans

In [32]: rest
Out[32]: [10, 11, 13]

In [33]: last
Out[33]: 30
```

ИСПОЛЬЗОВАНИЕ *

Таким образом можно указать, что нужен первый, второй и последний элемент:

```
In [34]: cdp = 'SW1              Eth 0/0          140          S I      WS-C3750-  Eth 0/1'

In [35]: name, l_intf, *other, r_intf = cdp.split()

In [36]: name
Out[36]: 'SW1'

In [37]: l_intf
Out[37]: 'Eth'

In [38]: r_intf
Out[38]: '0/1'
```

ПРИМЕРЫ РАСПАКОВКИ

РАСПАКОВКА ИТЕРИРУЕМЫХ ОБЪЕКТОВ

Эти примеры показывают, что распаковывать можно не только списки, кортежи и строки, но и любой другой итерируемый объект.

Распаковка range:

```
In [39]: first, *rest = range(1,6)

In [40]: first
Out[40]: 1

In [41]: rest
Out[41]: [2, 3, 4, 5]
```

РАСПАКОВКА ИТЕРИРУЕМЫХ ОБЪЕКТОВ

Распаковка zip:

```
In [42]: a = [1,2,3,4,5]

In [43]: b = [100,200,300,400,500]

In [44]: zip(a, b)
Out[44]: <zip at 0xb4df4fac>

In [45]: list(zip(a, b))
Out[45]: [(1, 100), (2, 200), (3, 300), (4, 400), (5, 500)]

In [46]: first, *rest, last = zip(a, b)

In [47]: first
Out[47]: (1, 100)

In [48]: rest
Out[48]: [(2, 200), (3, 300), (4, 400)]

In [49]: last
Out[49]: (5, 500)
```

ПРИМЕР РАСПАКОВКИ В ЦИКЛЕ FOR

Пример цикла, который проходится по ключам:

```
In [50]: access_template = ['switchport mode access',
    ...:                    'switchport access vlan',
    ...:                    'spanning-tree portfast',
    ...:                    'spanning-tree bpduguard enable']
    ...:

In [51]: access = {'0/12':10,
    ...:           '0/14':11,
    ...:           '0/16':17}
    ...:

In [52]: for intf in access:
    ...:     print('interface FastEthernet' + intf)
    ...:     for command in access_template:
    ...:         if command.endswith('access vlan'):
    ...:             print(' {} {}'.format(command, access[intf]))
    ...:         else:
    ...:             print(' {}'.format(command))
    ...:
interface FastEthernet0/12
 switchport mode access
 switchport access vlan 10
 spanning-tree portfast
 spanning-tree bpduguard enable
interface FastEthernet0/14
 switchport mode access
 switchport access vlan 11
 spanning-tree portfast
 spanning-tree bpduguard enable
```

РАСПАКОВКА ИТЕРИРУЕМЫХ ОБЪЕКТОВ

Вместо этого можно проходиться по парам ключ-значение и сразу же распаковывать их в разные переменные:

```
In [53]: for intf, vlan in access.items():
    ...:     print('interface FastEthernet' + intf)
    ...:     for command in access_template:
    ...:         if command.endswith('access vlan'):
    ...:             print(' {} {}'.format(command, vlan))
    ...:         else:
    ...:             print(' {}'.format(command))
    ...:
```

РАСПАКОВКА ИТЕРИРУЕМЫХ ОБЪЕКТОВ

Пример распаковки элементов списка в цикле:

```
In [54]: table
Out[54]:
[['100', 'a1b2.ac10.7000', 'DYNAMIC', 'Gi0/1'],
 ['200', 'a0d4.cb20.7000', 'DYNAMIC', 'Gi0/2'],
 ['300', 'acb4.cd30.7000', 'DYNAMIC', 'Gi0/3'],
 ['100', 'a2bb.ec40.7000', 'DYNAMIC', 'Gi0/4'],
 ['500', 'aa4b.c550.7000', 'DYNAMIC', 'Gi0/5'],
 ['200', 'a1bb.1c60.7000', 'DYNAMIC', 'Gi0/6'],
 ['300', 'aa0b.cc70.7000', 'DYNAMIC', 'Gi0/7']]


In [55]: for line in table:
    ...:     vlan, mac, _, intf = line
    ...:     print(vlan, mac, intf)
    ...:
100 a1b2.ac10.7000 Gi0/1
200 a0d4.cb20.7000 Gi0/2
300 acb4.cd30.7000 Gi0/3
100 a2bb.ec40.7000 Gi0/4
500 aa4b.c550.7000 Gi0/5
200 a1bb.1c60.7000 Gi0/6
300 aa0b.cc70.7000 Gi0/7
```

РАСПАКОВКА ИТЕРИРУЕМЫХ ОБЪЕКТОВ

Но еще лучше сделать так:

```
In [56]: for vlan, mac, _, intf in table:
    ...:     print(vlan, mac, intf)
    ...:
100 a1b2.ac10.7000 Gi0/1
200 a0d4.cb20.7000 Gi0/2
300 acb4.cd30.7000 Gi0/3
100 a2bb.ec40.7000 Gi0/4
500 aa4b.c550.7000 Gi0/5
200 a1bb.1c60.7000 Gi0/6
300 aa0b.cc70.7000 Gi0/7
```

LIST, DICT, SET COMPREHENSIONS

LIST, DICT, SET COMPREHENSIONS

Python поддерживает специальные выражения, которые позволяют компактно создавать списки, словари и множества.

На английском эти выражения называются, соответственно:

- List comprehensions
- Dict comprehensions
- Set comprehensions

Эти выражения не только позволяют более компактно создавать соответствующие объекты, но и создают их быстрее. И хотя поначалу они требуют определенной привычки использования и понимания, они очень часто используются.

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Генератор списка - это выражение вида:

```
In [1]: vlans = ['vlan {}'.format(num) for num in range(10,16)]

In [2]: print(vlans)
['vlan 10', 'vlan 11', 'vlan 12', 'vlan 13', 'vlan 14', 'vlan 15']
```

В общем случае, это выражение, которое преобразует итерируемый объект в список. То есть, последовательность элементов преобразуется и добавляется в новый список.

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Выражению выше аналогичен такой цикл:

```
In [3]: vlans = []

In [4]: for num in range(10,16):
   ...:     vlans.append('vlan {}'.format(num))
   ...:

In [5]: print(vlans)
['vlan 10', 'vlan 11', 'vlan 12', 'vlan 13', 'vlan 14', 'vlan 15']
```

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

В list comprehensions можно использовать выражение if. Таким образом можно добавлять в список только некоторые объекты.

Например, такой цикл отбирает те элементы, которые являются числами, конвертирует их и добавляет в итоговый список only_digits:

```
In [6]: items = ['10', '20', 'a', '30', 'b', '40']

In [7]: only_digits = []

In [8]: for item in items:
   ...:     if item.isdigit():
   ...:         only_digits.append(int(item))
   ...:

In [9]: print(only_digits)
[10, 20, 30, 40]
```

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Аналогичный вариант в виде list comprehensions:

```
In [10]: items = ['10', '20', 'a', '30', 'b', '40']

In [11]: only_digits = [int(item) for item in items if item.isdigit()]

In [12]: print(only_digits)
[10, 20, 30, 40]
```

Конечно, далеко не все циклы можно переписать как генератор списка, но когда это можно сделать, и при этом выражение не усложняется, лучше использовать генераторы списка.

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

С помощью генератора списка также удобно получать элементы из вложенных словарей:

```
In [13]: london_co = {
    ...:     'r1' : {
    ...:     'hostname': 'london_r1',
    ...:     'location': '21 New Globe Walk',
    ...:     'vendor': 'Cisco',
    ...:     'model': '4451',
    ...:     'IOS': '15.4',
    ...:     'IP': '10.255.0.1'
    ...:     },
    ...:     'r2' : {
    ...:     'hostname': 'london_r2',
    ...:     'location': '21 New Globe Walk',
    ...:     'vendor': 'Cisco',
    ...:     'model': '4451',
    ...:     'IOS': '15.4',
    ...:     'IP': '10.255.0.2'
    ...:     },
    ...:     'sw1' : {
    ...:     'hostname': 'london_sw1',
    ...:     'location': '21 New Globe Walk',
    ...:     'vendor': 'Cisco',
    ...:     'model': '3850',
    ...:     'IOS': '3.6.XE',
    ...:     'IP': '10.255.0.101'
```

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Полный синтаксис генератора списка выглядит так:

```
[expression for item1 in iterable1 if condition1
            for item2 in iterable2 if condition2
            ...
            for itemN in iterableN if conditionN ]
```

Это значит, можно использовать несколько for в выражении.

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Например, в списке vlans находятся несколько вложенных списков с VLAN'ами:

```
In [16]: vlans = [[10,21,35], [101, 115, 150], [111, 40, 50]]
```

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Из этого списка надо сформировать один плоский список с номерами VLAN. Первый вариант, с помощью циклов for:

```
In [17]: result = []

In [18]: for vlan_list in vlans:
    ...:     for vlan in vlan_list:
    ...:         result.append(vlan)
    ...:

In [19]: print(result)
[10, 21, 35, 101, 115, 150, 111, 40, 50]
```

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Аналогичный вариант с генератором списков:

```
In [20]: vlans = [[10,21,35], [101, 115, 150], [111, 40, 50]]

In [21]: result = [vlan for vlan_list in vlans for vlan in vlan_list]

In [22]: print(result)
[10, 21, 35, 101, 115, 150, 111, 40, 50]
```

LIST COMPREHENSIONS (ГЕНЕРАТОРЫ СПИСКОВ)

Можно одновременно проходиться по двум последовательностям, используя zip:

```
In [23]: vlans = [100, 110, 150, 200]

In [24]: names = ['mngmt', 'voice', 'video', 'dmz']

In [25]: result = ['vlan {}\n name {}'.format(vlan, name) for vlan, name in zip(vlans, names)]

In [26]: print('\n'.join(result))
vlan 100
 name mngmt
vlan 110
 name voice
vlan 150
 name video
vlan 200
 name dmz
```

DICT COMPREHENSIONS (ГЕНЕРАТОРЫ СЛОВАРЕЙ)

DICT COMPREHENSIONS (ГЕНЕРАТОРЫ СЛОВАРЕЙ)

Генераторы словарей аналогичны генераторам списков, но они используются для создания словарей.

Например, такое выражение:

```
In [27]: d = {}

In [28]: for num in range(1,11):
    ...:     d[num] = num**2
    ...:

In [29]: print(d)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

DICT COMPREHENSIONS (ГЕНЕРАТОРЫ СЛОВАРЕЙ)

Можно заменить генератором словаря:

```
In [30]: d = {num: num**2 for num in range(1,11)}

In [31]: print(d)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

DICT COMPREHENSIONS (ГЕНЕРАТОРЫ СЛОВАРЕЙ)

Еще один пример, в котором надо преобразовать существующий словарь и перевести все ключи в нижний регистр. Для начала, вариант решения без генератора словаря:

```
In [32]: r1 = {'IOS': '15.4',
    ...:   'IP': '10.255.0.1',
    ...:   'hostname': 'london_r1',
    ...:   'location': '21 New Globe Walk',
    ...:   'model': '4451',
    ...:   'vendor': 'Cisco'}
    ...:

In [33]: lower_r1 = {}

In [34]: for key, value in r1.items():
    ...:     lower_r1[str.lower(key)] = value
    ...:

In [35]: lower_r1
Out[35]:
{'hostname': 'london_r1',
 'ios': '15.4',
 'ip': '10.255.0.1',
 'location': '21 New Globe Walk',
 'model': '4451',
 'vendor': 'Cisco'}
```

DICT COMPREHENSIONS (ГЕНЕРАТОРЫ СЛОВАРЕЙ)

Аналогичный вариант с помощью генератора словаря:

```
In [36]: r1 = {'IOS': '15.4',
    ...:   'IP': '10.255.0.1',
    ...:   'hostname': 'london_r1',
    ...:   'location': '21 New Globe Walk',
    ...:   'model': '4451',
    ...:   'vendor': 'Cisco'}
    ...:

In [37]: lower_r1 = {str.lower(key): value for key, value in r1.items()}

In [38]: lower_r1
Out[38]:
{'hostname': 'london_r1',
 'ios': '15.4',
 'ip': '10.255.0.1',
 'location': '21 New Globe Walk',
 'model': '4451',
 'vendor': 'Cisco'}
```

DICT COMPREHENSIONS (ГЕНЕРАТОРЫ СЛОВАРЕЙ)

Как и list comprehensions, dict comprehensions можно делать вложенными. Попробуем аналогичным образом преобразовать ключи во вложенных словарях:

```
In [39]: london_co = {
    ...:     'r1' : {
    ...:     'hostname': 'london_r1',
    ...:     'location': '21 New Globe Walk',
    ...:     'vendor': 'Cisco',
    ...:     'model': '4451',
    ...:     'IOS': '15.4',
    ...:     'IP': '10.255.0.1'
    ...:     },
    ...:     'r2' : {
    ...:     'hostname': 'london_r2',
    ...:     'location': '21 New Globe Walk',
    ...:     'vendor': 'Cisco',
    ...:     'model': '4451',
    ...:     'IOS': '15.4',
    ...:     'IP': '10.255.0.2'
    ...:     },
    ...:     'sw1' : {
    ...:     'hostname': 'london_sw1',
    ...:     'location': '21 New Globe Walk',
    ...:     'vendor': 'Cisco',
    ...:     'model': '3850',
```

DICT COMPREHENSIONS (ГЕНЕРАТОРЫ СЛОВАРЕЙ)

Аналогичное преобразование с dict comprehensions:

```
In [43]: result = {device: {str.lower(key):value for key, value in params.items()} for device, params in lor

In [44]: result
Out[44]:
{'r1': {'hostname': 'london_r1',
  'ios': '15.4',
  'ip': '10.255.0.1',
  'location': '21 New Globe Walk',
  'model': '4451',
  'vendor': 'Cisco'},
 'r2': {'hostname': 'london_r2',
  'ios': '15.4',
  'ip': '10.255.0.2',
  'location': '21 New Globe Walk',
  'model': '4451',
  'vendor': 'Cisco'},
 'sw1': {'hostname': 'london_sw1',
  'ios': '3.6.XE',
  'ip': '10.255.0.101',
  'location': '21 New Globe Walk',
  'model': '3850',
  'vendor': 'Cisco'}}
```

SET COMPREHENSIONS (ГЕНЕРАТОРЫ МНОЖЕСТВ)

SET COMPREHENSIONS (ГЕНЕРАТОРЫ МНОЖЕСТВ)

Генераторы множеств в целом аналогичны генераторам списков.

Например, надо получить множество с уникальными номерами VLAN'ов:

```
In [45]: vlans = [10, '30', 30, 10, '56']

In [46]: unique_vlans = {int(vlan) for vlan in vlans}

In [47]: unique_vlans
Out[47]: {10, 30, 56}
```

SET COMPREHENSIONS (ГЕНЕРАТОРЫ МНОЖЕСТВ)

Аналогичное решение, без использования set comprehensions:

```
In [48]: vlans = [10, '30', 30, 10, '56']

In [49]: unique_vlans = set()

In [50]: for vlan in vlans:
    ...:     unique_vlans.add(int(vlan))
    ...:

In [51]: unique_vlans
Out[51]: {10, 30, 56}
```

РАБОТА СО СЛОВАРЯМИ

РАБОТА СО СЛОВАРЯМИ

При обработке вывода команд или конфигурации часто надо будет записать итоговые данные в словарь.

Не всегда очевидно как обрабатывать вывод команд и каким образом в целом подходить к разбору вывода на части. В этом подразделе рассматриваются несколько примеров, с возрастающим уровнем сложности.

РАЗБОР ВЫВОДА СТОЛБЦАМИ

В этом примере будет разбираться вывод команды sh ip int br. Из вывода команды нам надо получить соответствия имя интерфейса - IP-адрес. То есть имя интерфейса - это ключ словаря, а IP-адрес - значение. При этом, соответствие надо делать только для тех интерфейсов, у которых назначен IP-адрес.

РАЗБОР ВЫВОДА СТОЛБЦАМИ

Пример вывода команды sh ip int br (файл sh_ip_int_br.txt):

```
R1#show ip interface brief
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            15.0.15.1       YES manual up                    up
FastEthernet0/1            10.0.12.1       YES manual up                    up
FastEthernet0/2            10.0.13.1       YES manual up                    up
FastEthernet0/3            unassigned      YES unset  up                    down
Loopback0                  10.1.1.1        YES manual up                    up
Loopback100                100.0.0.1       YES manual up                    up
```

РАЗБОР ВЫВОДА СТОЛБЦАМИ

Файл working_with_dict_example_1.py:

```
result = {}

with open('sh_ip_int_br.txt') as f:
    for line in f:
        line = line.split()
        if line and line[1][0].isdigit():
            interface, address, *other = line
            result[interface] = address

print(result)
```

РАЗБОР ВЫВОДА СТОЛБЦАМИ

Результатом выполнения скрипта будет такой словарь (тут он разбит на пары ключ-значение для удобства, в реальном выводе скрипта словарь будет отображаться в одну строку):

```
{'FastEthernet0/0': '15.0.15.1',
 'FastEthernet0/1': '10.0.12.1',
 'FastEthernet0/2': '10.0.13.1',
 'Loopback0': '10.1.1.1',
 'Loopback100': '100.0.0.1'}
```

ПОЛУЧЕНИЕ КЛЮЧА И ЗНАЧЕНИЯ ИЗ РАЗНЫХ СТРОК ВЫВОДА

ПОЛУЧЕНИЕ КЛЮЧА И ЗНАЧЕНИЯ ИЗ РАЗНЫХ СТРОК ВЫВОДА

Очень часто вывод команд выглядит таким образом, что ключ и значение находятся в разных строках. И надо придумать каким образом обрабатывать вывод, чтобы получить нужное соответствие.

Например, из вывода команды sh ip interface надо получить соответствие имя интерфейса - MTU (файл sh_ip_interface.txt):

```
Ethernet0/0 is up, line protocol is up
  Internet address is 192.168.100.1/24
  Broadcast address is 255.255.255.255
  Address determined by non-volatile memory
  MTU is 1500 bytes
  Helper address is not set
  ...
Ethernet0/1 is up, line protocol is up
  Internet address is 192.168.200.1/24
  Broadcast address is 255.255.255.255
  Address determined by non-volatile memory
  MTU is 1500 bytes
  Helper address is not set
  ...
```

ПОЛУЧЕНИЕ КЛЮЧА И ЗНАЧЕНИЯ ИЗ РАЗНЫХ СТРОК ВЫВОДА

Имя интерфейса находится в строке вида `Ethernet0/0 is up, line protocol is up`, а `MTU` в строке вида `MTU is 1500 bytes`.

Например, попробуем запоминать каждый раз интерфейс и выводить его значение, когда встречается MTU, вместе со значением MTU:

```
In [2]: with open('sh_ip_interface.txt') as f:
   ...:     for line in f:
   ...:         if 'line protocol' in line:
   ...:             interface = line.split()[0]
   ...:         elif 'MTU is' in line:
   ...:             mtu = line.split()[-2]
   ...:             print('{:15}{}'.format(interface, mtu))
   ...:
Ethernet0/0    1500
Ethernet0/1    1500
Ethernet0/2    1500
Ethernet0/3    1500
Loopback0      1514
```

ПОЛУЧЕНИЕ КЛЮЧА И ЗНАЧЕНИЯ ИЗ РАЗНЫХ СТРОК ВЫВОДА

Вывод организован таким образом, что всегда сначала идет строка с интерфейсом, а затем через несоколько строк - строка с MTU. Если запоминать имя интерфейса каждый раз, когда оно встречается, то на момент когда встретится строка с MTU, последний запомненный интерфейс - это тот к которому относится MTU.

Теперь, если необходимо создать словарь с соответствием интерфейс - MTU, достаточно записать значения на момент, когда был найден MTU.

ПОЛУЧЕНИЕ КЛЮЧА И ЗНАЧЕНИЯ ИЗ РАЗНЫХ СТРОК ВЫВОДА

Файл working_with_dict_example_2.py:

```
result = {}

with open('sh_ip_interface.txt') as f:
    for line in f:
        if 'line protocol' in line:
            interface = line.split()[0]
        elif 'MTU is' in line:
            mtu = line.split()[-2]
            result[interface] = mtu

print(result)
```

ПОЛУЧЕНИЕ КЛЮЧА И ЗНАЧЕНИЯ ИЗ РАЗНЫХ СТРОК ВЫВОДА

Результатом выполнения скрипта будет такой словарь (тут он разбит на пары ключ-значение для удобства, в реальном выводе скрипта словарь будет отображаться в одну строку):

```
{'Ethernet0/0': '1500',
 'Ethernet0/1': '1500',
 'Ethernet0/2': '1500',
 'Ethernet0/3': '1500',
 'Loopback0': '1514'}
```

Этот прием будет достаточно часто полезен, так как вывод команд, в целом, организован очень похожим образом.

ВЛОЖЕННЫЙ СЛОВАРЬ

ВЛОЖЕННЫЙ СЛОВАРЬ

Если из вывода команды надо получить несколько параметров, очень удобно использовать словарь с вложенным словарем.

Например, из вывода sh ip interface надо получить два параметра: IP-адрес и MTU. Для начала, вывод информации:

```
In [2]: with open('sh_ip_interface.txt') as f:
   ...:     for line in f:
   ...:         if 'line protocol' in line:
   ...:             interface = line.split()[0]
   ...:         elif 'Internet address' in line:
   ...:             ip_address = line.split()[-1]
   ...:         elif 'MTU' in line:
   ...:             mtu = line.split()[-2]
   ...:             print('{:15}{:17}{}'.format(interface, ip_address, mtu))
   ...:
Ethernet0/0    192.168.100.1/24 1500
Ethernet0/1    192.168.200.1/24 1500
Ethernet0/2    19.1.1.1/24      1500
Ethernet0/3    192.168.230.1/24 1500
Loopback0      4.4.4.4/32       1514
```

ВЛОЖЕННЫЙ СЛОВАРЬ

Тут используется такой же прием, как в предыдущем примере, но добавляется еще одна вложенность словаря:

```
result = {}

with open('sh_ip_interface.txt') as f:
    for line in f:
        if 'line protocol' in line:
            interface = line.split()[0]
            result[interface] = {}
        elif 'Internet address' in line:
            ip_address = line.split()[-1]
            result[interface]['ip'] = ip_address
        elif 'MTU' in line:
            mtu = line.split()[-2]
            result[interface]['mtu'] = mtu

print(result)
```

ВЛОЖЕННЫЙ СЛОВАРЬ

Каждый раз, когда встречается интерфейс, в словаре result создается ключ с именем интерфейса, которому соответствует пустой словарь. Эта заготовка нужна для того, чтобы на момент когда встретится IP-адрес или MTU можно было записать параметр во вложенный словарь соответствующего интерфейса.

ВЛОЖЕННЫЙ СЛОВАРЬ

Результатом выполнения скрипта будет такой словарь (тут он разбит на пары ключ-значение для удобства, в реальном выводе скрипта словарь будет отображаться в одну строку):

```
{'Ethernet0/0': {'ip': '192.168.100.1/24', 'mtu': '1500'},
 'Ethernet0/1': {'ip': '192.168.200.1/24', 'mtu': '1500'},
 'Ethernet0/2': {'ip': '19.1.1.1/24', 'mtu': '1500'},
 'Ethernet0/3': {'ip': '192.168.230.1/24', 'mtu': '1500'},
 'Loopback0': {'ip': '4.4.4.4/32', 'mtu': '1514'}}
```

ВЫВОД С ПУСТЫМИ ЗНАЧЕНИЯМИ

ВЫВОД С ПУСТЫМИ ЗНАЧЕНИЯМИ

Иногда, в выводе будут попадаться секции с пустыми значениями. Например, в случае с выводом sh ip interface, могут попадаться интерфейс, которые выглядят так:

```
Ethernet0/1 is up, line protocol is up
  Internet protocol processing disabled
Ethernet0/2 is administratively down, line protocol is down
  Internet protocol processing disabled
Ethernet0/3 is administratively down, line protocol is down
  Internet protocol processing disabled
```

Соответственно тут нет MTU или IP-адреса.

ВЫВОД С ПУСТЫМИ ЗНАЧЕНИЯМИ

И, если выполнить предыдущий скрипт для файла с такими интерфейсами, результат будет таким (вывод для файла sh_ip_interface2.txt):

```
{'Ethernet0/0': {'ip': '192.168.100.2/24', 'mtu': '1500'},
 'Ethernet0/1': {},
 'Ethernet0/2': {},
 'Ethernet0/3': {},
 'Loopback0': {'ip': '2.2.2.2/32', 'mtu': '1514'}}
```

ВЫВОД С ПУСТЫМИ ЗНАЧЕНИЯМИ

Если необходимо добавлять интерфейсы в словарь только, когда на интерфейсе назначен IP-адрес, надо перенести создание ключа с именем интерфейса на момент, когда встречается строка с IP-адресом (файл working_with_dict_example_4.py):

```
result = {}

with open('sh_ip_interface2.txt') as f:
    for line in f:
        if 'line protocol' in line:
            interface = line.split()[0]
        elif 'Internet address' in line:
            ip_address = line.split()[-1]
            result[interface] = {}
            result[interface]['ip'] = ip_address
        elif 'MTU' in line:
            mtu = line.split()[-2]
            result[interface]['mtu'] = mtu

print(result)
```

ВЫВОД С ПУСТЫМИ ЗНАЧЕНИЯМИ

В этом случае, результатом будет такой словарь:

```
{'Ethernet0/0': {'ip': '192.168.100.2/24', 'mtu': '1500'},
 'Loopback0': {'ip': '2.2.2.2/32', 'mtu': '1514'}}
```