PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

ПОЛЕЗНЫЕ МОДУЛИ

МОДУЛЬ SUBPROCESS

МОДУЛЬ SUBPROCESS

Модуль subprocess позволяет создавать новые процессы.
При этом он может подключаться к стандартным потокам ввода/вывода/ошибок и получать код возврата.

С помощью subprocess можно, например, выполнять любые команды Linux из скрипта.
И, в зависимости от ситуации, получать вывод или только проверять, что команда выполнилась без ошибок.

ФУНКЦИЯ `subprocess.run()`

Функция `subprocess.run()` - основной способ работы с модулем subprocess.

```
In [1]: import subprocess

In [2]: result = subprocess.run('ls')
ipython_as_mngmt_console.md  README.md         version_control.md
module_search.md             useful_functions
naming_conventions           useful_modules
```

ФУНКЦИЯ `subprocess.run()`

В переменной result теперь содержится специальный объект CompletedProcess. Из этого объекта можно получить информацию о выполнении процесса, например, о коде возврата:

```
In [3]: result
Out[3]: CompletedProcess(args='ls', returncode=0)

In [4]: result.returncode
Out[4]: 0
```

ФУНКЦИЯ `subprocess.run()`

Если необходимо вызвать команду с аргументами, её нужно передавать таким образом (как список):

```
In [5]: result = subprocess.run(['ls', '-ls'])
total 28
4 -rw-r--r-- 1 vagrant vagrant   56 Jun  7 19:35 ipython_as_mngmt_console.md
4 -rw-r--r-- 1 vagrant vagrant 1638 Jun  7 19:35 module_search.md
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun  7 19:35 naming_conventions
4 -rw-r--r-- 1 vagrant vagrant  277 Jun  7 19:35 README.md
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun 16 05:11 useful_functions
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun 17 16:28 useful_modules
4 -rw-r--r-- 1 vagrant vagrant   49 Jun  7 19:35 version_control.md
```

ФУНКЦИЯ `subprocess.run()`

При попытке выполнить команду с использованием wildcard выражений, например, использовать *, возникнет ошибка:

```
In [6]: result = subprocess.run(['ls', '-ls', '*md'])
ls: cannot access *md: No such file or directory
```

ФУНКЦИЯ `subprocess.run()`

Чтобы вызывать команды, в которых используются wildcard выражения, нужно добавлять аргумент shell и вызывать команду таким образом:

```
In [7]: result = subprocess.run('ls -ls *md', shell=True)
4 -rw-r--r-- 1 vagrant vagrant   56 Jun  7 19:35 ipython_as_mngmt_console.md
4 -rw-r--r-- 1 vagrant vagrant 1638 Jun  7 19:35 module_search.md
4 -rw-r--r-- 1 vagrant vagrant  277 Jun  7 19:35 README.md
4 -rw-r--r-- 1 vagrant vagrant   49 Jun  7 19:35 version_control.md
```

ФУНКЦИЯ `subprocess.run()`

Ещё одна особенность функции run() - она ожидает завершения выполнения команды. Если попробовать, например, запустить команду ping, то этот аспект будет заметен:

```
In [8]: result = subprocess.run(['ping', '-c', '3', '-n', '8.8.8.8'])
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=43 time=55.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=43 time=54.7 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=43 time=54.4 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 54.498/54.798/55.116/0.252 ms
```

ПОЛУЧЕНИЕ РЕЗУЛЬТАТА ВЫПОЛНЕНИЯ КОМАНДЫ

По умолчанию функция run возвращает результат выполнения команды на стандартный поток вывода.

Если нужно получить результат выполнения команды, надо добавить аргумент stdout и указать ему значение subprocess.PIPE:

```
In [9]: result = subprocess.run(['ls', '-ls'], stdout=subprocess.PIPE)
```

ПОЛУЧЕНИЕ РЕЗУЛЬТАТА ВЫПОЛНЕНИЯ КОМАНДЫ

Теперь можно получить результат выполнения команды таким образом:

```
In [10]: print(result.stdout)
b'total 28\n4 -rw-r--r-- 1 vagrant vagrant   56 Jun  7 19:35 ipython_as_mngmt_console.md\n4 -rw-r--r-- 1 vag
```

ПОЛУЧЕНИЕ РЕЗУЛЬТАТА ВЫПОЛНЕНИЯ КОМАНДЫ

Модуль вернул вывод в виде байтовой строки.

Для перевода её в unicode есть два варианта:

- выполнить decode полученной строки
- указать аргумент encoding

ПОЛУЧЕНИЕ РЕЗУЛЬТАТА ВЫПОЛНЕНИЯ КОМАНДЫ

Вариант с decode:

```
In [11]: print(result.stdout.decode('utf-8'))
total 28
4 -rw-r--r-- 1 vagrant vagrant   56 Jun  7 19:35 ipython_as_mngmt_console.md
4 -rw-r--r-- 1 vagrant vagrant 1638 Jun  7 19:35 module_search.md
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun  7 19:35 naming_conventions
4 -rw-r--r-- 1 vagrant vagrant  277 Jun  7 19:35 README.md
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun 16 05:11 useful_functions
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun 17 16:30 useful_modules
4 -rw-r--r-- 1 vagrant vagrant   49 Jun  7 19:35 version_control.md
```

ПОЛУЧЕНИЕ РЕЗУЛЬТАТА ВЫПОЛНЕНИЯ КОМАНДЫ

Вариант с encoding:

```
In [12]: result = subprocess.run(['ls', '-ls'], stdout=subprocess.PIPE, encoding='utf-8')

In [13]: print(result.stdout)
total 28
4 -rw-r--r-- 1 vagrant vagrant   56 Jun  7 19:35 ipython_as_mngmt_console.md
4 -rw-r--r-- 1 vagrant vagrant 1638 Jun  7 19:35 module_search.md
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun  7 19:35 naming_conventions
4 -rw-r--r-- 1 vagrant vagrant  277 Jun  7 19:35 README.md
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun 16 05:11 useful_functions
4 drwxr-xr-x 2 vagrant vagrant 4096 Jun 17 16:31 useful_modules
4 -rw-r--r-- 1 vagrant vagrant   49 Jun  7 19:35 version_control.md
```

ОТКЛЮЧЕНИЕ ВЫВОДА

Иногда достаточно получения кода возврата и нужно отключить вывод результата выполнения на стандартный поток вывода, и при этом сам результат не нужен.

Это можно сделать, передав функции run аргумент stdout со значением subprocess.DEVNULL:

```
In [14]: result = subprocess.run(['ls', '-ls'], stdout=subprocess.DEVNULL)

In [15]: print(result.stdout)
None

In [16]: print(result.returncode)
0
```

РАБОТА СО СТАНДАРТНЫМ ПОТОКОМ ОШИБОК

Если команда была выполнена с ошибкой или не отработала корректно, вывод команды попадет на стандартный поток ошибок.

Получить этот вывод можно так же, как и стандартный поток вывода:

```
In [17]: result = subprocess.run(['ping', '-c', '3', '-n', 'a'], stderr=subprocess.PIPE, encoding='utf-8')
```

РАБОТА СО СТАНДАРТНЫМ ПОТОКОМ ОШИБОК

Теперь в result.stdout пустая строка, а в result.stderr находится стандартный поток вывода:

```
In [18]: print(result.stdout)
None

In [19]: print(result.stderr)
ping: unknown host a


In [20]: print(result.returncode)
2
```

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ МОДУЛЯ

Пример использования модуля subprocess (файл subprocess_run_basic.py):

```
import subprocess

reply = subprocess.run(['ping', '-c', '3', '-n', '8.8.8.8'])

if reply.returncode == 0:
    print('Alive')
else:
    print('Unreachable')
```

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ МОДУЛЯ

Результат выполнения будет таким:

```
$ python subprocess_run_basic.py
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=43 time=54.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=43 time=54.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=43 time=53.9 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 53.962/54.145/54.461/0.293 ms
Alive
```

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ МОДУЛЯ

Функция ping_ip проверяет доступность IP-адреса и возвращает True и stdout, если адрес доступен, или False и stderr, если адрес недоступен (файл subprocess_ping_function.py):

```
import subprocess


def ping_ip(ip_address):
    """
    Ping IP address and return tuple:
    On success:
        * True
        * command output (stdout)
    On failure:
        * False
        * error output (stderr)
    """
    reply = subprocess.run(['ping', '-c', '3', '-n', ip_address],
                           stdout=subprocess.PIPE,
                           stderr=subprocess.PIPE,
                           encoding='utf-8')
    if reply.returncode == 0:
        return True, reply.stdout
    else:
        return False, reply.stderr

print(ping_ip('8.8.8.8'))
print(ping_ip('a'))
```

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ МОДУЛЯ

Результат выполнения будет таким:

```
$ python subprocess_ping_function.py
(True, 'PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.\n64 bytes from 8.8.8.8: icmp_seq=1 ttl=43 time=63.8 ms\
(False, 'ping: unknown host a\n')
```

МОДУЛЬ OS

МОДУЛЬ OS

Модуль os позволяет работать с файловой системой, с окружением, управлять процессами.

```
In [1]: import os

In [2]: os.mkdir('test')

In [3]: ls -ls
total 0
0 drwxr-xr-x  2 nata  nata  68 Jan 23 18:58 test/
```

МОДУЛЬ OS

Кроме того, в модуле есть соответствующие проверки на существование. Например, если попробовать повторно создать каталог, возникнет ошибка:

```
In [4]: os.mkdir('test')
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-4-cbf3b897c095> in <module>()
----> 1 os.mkdir('test')

FileExistsError: [Errno 17] File exists: 'test'
```

МОДУЛЬ OS

В таком случае пригодится проверка `os.path.exists`:

```
In [5]: os.path.exists('test')
Out[5]: True

In [6]: if not os.path.exists('test'):
   ...:     os.mkdir('test')
   ...:
```

МОДУЛЬ OS

Метод listdir позволяет посмотреть содержимое каталога:

```
In [7]: os.listdir('.')
Out[7]: ['cover3.png', 'dir2', 'dir3', 'README.txt', 'test']
```

С помощью проверок `os.path.isdir` и `os.path.isfile` можно получить отдельно список файлов и список каталогов:

```
In [8]: dirs = [ d for d in os.listdir('.') if os.path.isdir(d)]

In [9]: dirs
Out[9]: ['dir2', 'dir3', 'test']

In [10]: files = [ f for f in os.listdir('.') if os.path.isfile(f)]

In [11]: files
Out[11]: ['cover3.png', 'README.txt']
```

МОДУЛЬ OS

Также в модуле есть отдельные методы для работы с путями:

```
In [12]: os.path.basename(file)
Out[12]: 'README.md'

In [13]: os.path.dirname(file)
Out[13]: 'Programming/PyNEng/book/25_additional_info'

In [14]: os.path.split(file)
Out[14]: ('Programming/PyNEng/book/25_additional_info', 'README.md')
```

МОДУЛЬ IPADDRESS

`ipaddress.ip_address()`

Функция `ipaddress.ip_address()` позволяет создавать объект IPv4Address или IPv6Address соответственно.

IPv4 адрес:

```
In [1]: import ipaddress

In [2]: ipv4 = ipaddress.ip_address('10.0.1.1')

In [3]: ipv4
Out[3]: IPv4Address('10.0.1.1')

In [4]: print(ipv4)
10.0.1.1
```

`ipaddress.ip_address()`

У объекта есть несколько методов и атрибутов:

```
In [5]: ipv4.
ipv4.compressed       ipv4.is_loopback      ipv4.is_unspecified   ipv4.version
ipv4.exploded         ipv4.is_multicast     ipv4.max_prefixlen
ipv4.is_global        ipv4.is_private       ipv4.packed
ipv4.is_link_local    ipv4.is_reserved      ipv4.reverse_pointer
```

`ipaddress.ip_address()`

С помощью атрибутов is_ можно проверить, к какому диапазону принадлежит адрес:

```
In [6]: ipv4.is_loopback
Out[6]: False

In [7]: ipv4.is_multicast
Out[7]: False

In [8]: ipv4.is_reserved
Out[8]: False

In [9]: ipv4.is_private
Out[9]: True
```

ipaddress.ip_address()

С полученными объектами можно выполнять различные операции:

```
In [10]: ip1 = ipaddress.ip_address('10.0.1.1')

In [11]: ip2 = ipaddress.ip_address('10.0.2.1')

In [12]: ip1 > ip2
Out[12]: False

In [13]: ip2 > ip1
Out[13]: True

In [14]: ip1 == ip2
Out[14]: False

In [15]: ip1 != ip2
Out[15]: True

In [16]: str(ip1)
Out[16]: '10.0.1.1'

In [17]: int(ip1)
Out[17]: 167772417

In [18]: ip1 + 5
Out[18]: IPv4Address('10.0.1.6')

In [19]: ip1 - 5
Out[19]: IPv4Address('10.0.0.252')
```

`ipaddress.ip_network()`

Функция `ipaddress.ip_network()` позволяет создать объект, который описывает сеть (IPv4 или IPv6).

Сеть IPv4:

```
In [20]: subnet1 = ipaddress.ip_network('80.0.1.0/28')
```

`ipaddress.ip_network()`

Как и у адреса, у сети есть различные атрибуты и методы:

```
In [21]: subnet1.broadcast_address
Out[21]: IPv4Address('80.0.1.15')

In [22]: subnet1.with_netmask
Out[22]: '80.0.1.0/255.255.255.240'

In [23]: subnet1.with_hostmask
Out[23]: '80.0.1.0/0.0.0.15'

In [24]: subnet1.prefixlen
Out[24]: 28

In [25]: subnet1.num_addresses
Out[25]: 16
```

ipaddress.ip_network()

Метод hosts() возвращает генератор, поэтому, чтобы посмотреть все хосты, надо применить функцию list:

```
In [26]: list(subnet1.hosts())
Out[26]:
[IPv4Address('80.0.1.1'),
 IPv4Address('80.0.1.2'),
 IPv4Address('80.0.1.3'),
 IPv4Address('80.0.1.4'),
 IPv4Address('80.0.1.5'),
 IPv4Address('80.0.1.6'),
 IPv4Address('80.0.1.7'),
 IPv4Address('80.0.1.8'),
 IPv4Address('80.0.1.9'),
 IPv4Address('80.0.1.10'),
 IPv4Address('80.0.1.11'),
 IPv4Address('80.0.1.12'),
 IPv4Address('80.0.1.13'),
 IPv4Address('80.0.1.14')]
```

ipaddress.ip_network()

Метод subnets позволяет разбивать на подсети. По умолчанию он разбивает сеть на две подсети:

```
In [27]: list(subnet1.subnets())
Out[27]: [IPv4Network('80.0.1.0/29'), IPv4Network(u'80.0.1.8/29')]
```

Но можно передать параметр prefixlen_diff, чтобы указать количество бит для подсетей:

```
In [28]: list(subnet1.subnets(prefixlen_diff=2))
Out[28]:
[IPv4Network('80.0.1.0/30'),
 IPv4Network('80.0.1.4/30'),
 IPv4Network('80.0.1.8/30'),
 IPv4Network('80.0.1.12/30')]
```

ipaddress.ip_network()

Или с помощью параметра new_prefix просто указать, какая маска должна быть у подсетей:

```
In [29]: list(subnet1.subnets(new_prefix=30))
Out[29]:
[IPv4Network('80.0.1.0/30'),
 IPv4Network('80.0.1.4/30'),
 IPv4Network('80.0.1.8/30'),
 IPv4Network('80.0.1.12/30')]

In [30]: list(subnet1.subnets(new_prefix=29))
Out[30]: [IPv4Network('80.0.1.0/29'), IPv4Network(u'80.0.1.8/29')]
```

ipaddress.ip_network()

По IP-адресам в сети можно проходиться в цикле:

```
In [31]: for ip in subnet1:
   ....:     print(ip)
   ....:
80.0.1.0
80.0.1.1
80.0.1.2
80.0.1.3
80.0.1.4
80.0.1.5
80.0.1.6
80.0.1.7
80.0.1.8
80.0.1.9
80.0.1.10
80.0.1.11
80.0.1.12
80.0.1.13
80.0.1.14
80.0.1.15
```

ipaddress.ip_network()

Или обращаться к конкретному адресу:

```
In [32]: subnet1[0]
Out[32]: IPv4Address('80.0.1.0')

In [33]: subnet1[5]
Out[33]: IPv4Address('80.0.1.5')
```

Таким образом можно проверять, находится ли IP-адрес в сети:

```
In [34]: ip1 = ipaddress.ip_address('80.0.1.3')

In [35]: ip1 in subnet1
Out[35]: True
```

`ipaddress.ip_interface()`

Функция `ipaddress.ip_interface()` позволяет создавать объект IPv4Interface или IPv6Interface соответственно.

Попробуем создать интерфейс:

```
In [36]: int1 = ipaddress.ip_interface('10.0.1.1/24')
```

ipaddress.ip_interface()

Используя методы объекта IPv4Interface, можно получать адрес, маску или сеть интерфейса:

```
In [37]: int1.ip
Out[37]: IPv4Address('10.0.1.1')

In [38]: int1.network
Out[38]: IPv4Network('10.0.1.0/24')

In [39]: int1.netmask
Out[39]: IPv4Address('255.255.255.0')
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ МОДУЛЯ

Так как в модуль встроены проверки корректности адресов, можно ими пользоваться, например, чтобы проверить, является ли адрес адресом сети или хоста:

```
In [40]: IP1 = '10.0.1.1/24'

In [41]: IP2 = '10.0.1.0/24'

In [42]: def check_if_ip_is_network(ip_address):
   ....:     try:
   ....:         ipaddress.ip_network(ip_address)
   ....:         return True
   ....:     except ValueError:
   ....:         return False
   ....:

In [43]: check_if_ip_is_network(IP1)
Out[43]: False

In [44]: check_if_ip_is_network(IP2)
Out[44]: True
```

МОДУЛЬ ARGPARSE

МОДУЛЬ ARGPARSE

argparse - это модуль для обработки аргументов командной строки.

Примеры того, что позволяет делать модуль:

- создавать аргументы и опции, с которыми может вызываться скрипт
- указывать типы аргументов, значения по умолчанию
- указывать, какие действия соответствуют аргументам
- выполнять вызов функции при указании аргумента
- отображать сообщения с подсказками по использованию скрипта

МОДУЛЬ ARGPARSE

argparse не единственный модуль для обработки аргументов командной строки.
И даже не единственный такой модуль в стандартной библиотеке.

Мы будем рассматривать только argparse. Но, если вы столкнетесь с необходимостью использовать подобные модули, обязательно посмотрите и на те модули, которые не входят в стандартную библиотеку Python.
Например, на click.

МОДУЛЬ ARGPARSE

```
import subprocess
import argparse


def ping_ip(ip_address, count):
    '''
    Ping IP address and return tuple:
    On success: (return code = 0, command output)
    On failure: (return code, error output (stderr))
    '''
    reply = subprocess.run('ping -c {count} -n {ip}'
                           .format(count=count, ip=ip_address),
                           shell=True,
                           stdout=subprocess.PIPE,
                           stderr=subprocess.PIPE,
                           encoding='utf-8')
    if reply.returncode == 0:
        return True, reply.stdout
    else:
        return False, reply.stdout+reply.stderr



parser = argparse.ArgumentParser(description='Ping script')

parser.add_argument('-a', action="store", dest="ip")
parser.add_argument('-c', action="store", dest="count", default=2, type=int)

args = parser.parse_args()
print(args)
```

МОДУЛЬ ARGPARSE

```
$ python ping_function.py -a 8.8.8.8 -c 5
Namespace(count=5, ip='8.8.8.8')
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=48 time=48.673 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=48 time=49.902 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=48 time=48.696 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=48 time=50.040 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=48 time=48.831 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 48.673/49.228/50.040/0.610 ms
```

МОДУЛЬ ARGPARSE

Передаем только IP-адрес:

```
$ python ping_function.py -a 8.8.8.8
Namespace(count=2, ip='8.8.8.8')
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=48 time=48.563 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=48 time=49.616 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 48.563/49.090/49.616/0.526 ms
```

МОДУЛЬ ARGPARSE

Также, благодаря argparse, доступен help:

```
$ python ping_function.py -h
usage: ping_function.py [-h] -a IP [-c COUNT]
Ping script
optional arguments:
  -h, --help  show this help message and exit
  -a IP
  -c COUNT
```

Обратите внимание, что в сообщении все опции находятся в секции `optional arguments`.

argparse сам определяет, что указаны опции, так как они начинаются с `-` и в имени только одна буква.

МОДУЛЬ ARGPARSE

Файл ping_function_ver2.py:

```
import subprocess
from tempfile import TemporaryFile

import argparse


def ping_ip(ip_address, count):
    '''
    Ping IP address and return tuple:
    On success: (return code = 0, command output)
    On failure: (return code, error output (stderr))
    '''
    reply = subprocess.run('ping -c {count} -n {ip}' .format(count=count, ip=ip_address),
                           shell=True,
                           stdout=subprocess.PIPE,
                           stderr=subprocess.PIPE,
                           encoding='utf-8')
    if reply.returncode == 0:
        return True, reply.stdout
    else:
        return False, reply.stdout+reply.stderr



parser = argparse.ArgumentParser(description='Ping script')

parser.add_argument('host', action="store", help="IP or name to ping")
parser.add_argument('-c', action="store", dest="count", default=2, type=int,
                    help="Number of packets")
```

МОДУЛЬ ARGPARSE

Теперь, вместо указания опции `-a`, можно просто передать IP-адрес.
Он будет автоматически сохранен в переменной `host`.
И автоматически считается обязательным.

То есть, теперь не нужно указывать `required=True` и `dest="ip"`.

Кроме того, в скрипте указаны сообщения, которые будут выводиться при вызове help.

МОДУЛЬ ARGPARSE

Теперь вызов скрипта выглядит так:

```
$ python ping_function_ver2.py 8.8.8.8 -c 2
Namespace(host='8.8.8.8', count=2)
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=48 time=49.203 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=48 time=51.764 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 49.203/50.484/51.764/1.280 ms
```

МОДУЛЬ ARGPARSE

Сообщение help:

```
$ python ping_function_ver2.py -h
usage: ping_function_ver2.py [-h] [-c COUNT] host
Ping script
positional arguments:
  host        IP or name to ping
optional arguments:
  -h, --help  show this help message and exit
  -c COUNT    Number of packets
```

МОДУЛЬ TABULATE

МОДУЛЬ TABULATE

tabulate - это библиотека, которая позволяет красиво отображать табличные данные.

tabulate не входит в стандартную библиотеку Python, поэтому его нужно установить:

```
pip install tabulate
```

Модуль поддерживает такие типы табличных данных:

- список списков (в общем случае iterable of iterables)
- список словарей (или любой другой итерируемый объект со словарями). Ключи используются как имена столбцов
- словарь с итерируемыми объектами. Ключи используются как имена столбцов

МОДУЛЬ TABULATE

Для генерации таблицы используется функция tabulate:

```
In [1]: from tabulate import tabulate

In [2]: sh_ip_int_br = [('FastEthernet0/0', '15.0.15.1', 'up', 'up'),
   ...:  ('FastEthernet0/1', '10.0.12.1', 'up', 'up'),
   ...:  ('FastEthernet0/2', '10.0.13.1', 'up', 'up'),
   ...:  ('Loopback0', '10.1.1.1', 'up', 'up'),
   ...:  ('Loopback100', '100.0.0.1', 'up', 'up')]
   ...:

In [4]: print(tabulate(sh_ip_int_br))
--------------  ---------  --  --
FastEthernet0/0  15.0.15.1  up  up
FastEthernet0/1  10.0.12.1  up  up
FastEthernet0/2  10.0.13.1  up  up
Loopback0        10.1.1.1   up  up
Loopback100      100.0.0.1  up  up
--------------  ---------  --  --
```

HEADERS

Параметр headers позволяет передавать дополнительный аргумент, в котором указаны имена столбцов:

```
In [8]: columns=['Interface', 'IP', 'Status', 'Protocol']

In [9]: print(tabulate(sh_ip_int_br, headers=columns))
Interface        IP         Status    Protocol
---------------  ---------  --------  ----------
FastEthernet0/0  15.0.15.1  up        up
FastEthernet0/1  10.0.12.1  up        up
FastEthernet0/2  10.0.13.1  up        up
Loopback0        10.1.1.1   up        up
Loopback100      100.0.0.1  up        up
```

HEADERS

Достаточно часто первый набор данных - это заголовки. Тогда достаточно указать headers равным "firstrow":

```
In [18]: data
Out[18]:
[('Interface', 'IP', 'Status', 'Protocol'),
 ('FastEthernet0/0', '15.0.15.1', 'up', 'up'),
 ('FastEthernet0/1', '10.0.12.1', 'up', 'up'),
 ('FastEthernet0/2', '10.0.13.1', 'up', 'up'),
 ('Loopback0', '10.1.1.1', 'up', 'up'),
 ('Loopback100', '100.0.0.1', 'up', 'up')]

In [20]: print(tabulate(data, headers='firstrow'))
Interface        IP         Status    Protocol
---------------  ---------  --------  ----------
FastEthernet0/0  15.0.15.1  up        up
FastEthernet0/1  10.0.12.1  up        up
FastEthernet0/2  10.0.13.1  up        up
Loopback0        10.1.1.1   up        up
Loopback100      100.0.0.1  up        up
```

HEADERS

Если данные в виде списка словарей, надо указать headers равным "keys":

```
In [22]: list_of_dict
Out[22]:
[{'IP': '15.0.15.1',
  'Interface': 'FastEthernet0/0',
  'Protocol': 'up',
  'Status': 'up'},
 {'IP': '10.0.12.1',
  'Interface': 'FastEthernet0/1',
  'Protocol': 'up',
  'Status': 'up'},
 {'IP': '10.0.13.1',
  'Interface': 'FastEthernet0/2',
  'Protocol': 'up',
  'Status': 'up'},
 {'IP': '10.1.1.1',
  'Interface': 'Loopback0',
  'Protocol': 'up',
  'Status': 'up'},
 {'IP': '100.0.0.1',
  'Interface': 'Loopback100',
  'Protocol': 'up',
  'Status': 'up'}]

In [23]: print(tabulate(list_of_dict, headers='keys'))
Interface        IP         Status    Protocol
---------------  ---------  --------  ----------
FastEthernet0/0  15.0.15.1  up        up
```

СТИЛЬ ТАБЛИЦЫ

tabulate поддерживает разные стили отображения таблицы.

Формат grid:

```
In [24]: print(tabulate(list_of_dict, headers='keys', tablefmt="grid"))
+-----------------+-----------+-----------+------------+
| Interface       | IP        | Status    | Protocol   |
+=================+===========+===========+============+
| FastEthernet0/0 | 15.0.15.1 | up        | up         |
+-----------------+-----------+-----------+------------+
| FastEthernet0/1 | 10.0.12.1 | up        | up         |
+-----------------+-----------+-----------+------------+
| FastEthernet0/2 | 10.0.13.1 | up        | up         |
+-----------------+-----------+-----------+------------+
| Loopback0       | 10.1.1.1  | up        | up         |
+-----------------+-----------+-----------+------------+
| Loopback100     | 100.0.0.1 | up        | up         |
+-----------------+-----------+-----------+------------+
```

СТИЛЬ ТАБЛИЦЫ

Таблица в формате Markdown:

```
In [25]: print(tabulate(list_of_dict, headers='keys', tablefmt='pipe'))
| Interface       | IP         | Status    | Protocol   |
|:----------------|:-----------|:----------|:-----------|
| FastEthernet0/0 | 15.0.15.1  | up        | up         |
| FastEthernet0/1 | 10.0.12.1  | up        | up         |
| FastEthernet0/2 | 10.0.13.1  | up        | up         |
| Loopback0       | 10.1.1.1   | up        | up         |
| Loopback100     | 100.0.0.1  | up        | up         |
```

СТИЛЬ ТАБЛИЦЫ

Таблица в формате HTML:

```
In [26]: print(tabulate(list_of_dict, headers='keys', tablefmt='html'))
<table>
<thead>
<tr><th>Interface      </th><th>IP       </th><th>Status  </th><th>Protocol  </th></tr>
</thead>
<tbody>
<tr><td>FastEthernet0/0</td><td>15.0.15.1</td><td>up      </td><td>up        </td></tr>
<tr><td>FastEthernet0/1</td><td>10.0.12.1</td><td>up      </td><td>up        </td></tr>
<tr><td>FastEthernet0/2</td><td>10.0.13.1</td><td>up      </td><td>up        </td></tr>
<tr><td>Loopback0      </td><td>10.1.1.1 </td><td>up      </td><td>up        </td></tr>
<tr><td>Loopback100    </td><td>100.0.0.1</td><td>up      </td><td>up        </td></tr>
</tbody>
</table>
```

ВЫРАВНИВАНИЕ СТОЛБЦОВ

Можно указывать выравнивание для столбцов:

```
In [27]: print(tabulate(list_of_dict, headers='keys', tablefmt='pipe', stralign='center'))
|    Interface    |    IP     |  Status  |  Protocol  |
|:---------------:|:---------:|:--------:|:----------:|
| FastEthernet0/0 | 15.0.15.1 |    up    |     up     |
| FastEthernet0/1 | 10.0.12.1 |    up    |     up     |
| FastEthernet0/2 | 10.0.13.1 |    up    |     up     |
|    Loopback0    | 10.1.1.1  |    up    |     up     |
|   Loopback100   | 100.0.0.1 |    up    |     up     |
```

МОДУЛЬ PPRINT

МОДУЛЬ PPRINT

Модуль pprint позволяет красиво отображать объекты Python. При этом сохраняется структура объекта и отображение, которое выводит pprint, можно использовать для создания объекта.

МОДУЛЬ PPRINT

```
In [6]: london_co = {'r1': {'hostname': 'london_r1', 'location': '21 New Globe Wal
   ...: k', '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'}}
   ...:

In [7]: from pprint import pprint

In [8]: pprint(london_co)
{'r1': {'IOS': '15.4',
        'IP': '10.255.0.1',
        'hostname': 'london_r1',
        'location': '21 New Globe Walk',
        'model': '4451',
        'vendor': 'Cisco'},
 'r2': {'IOS': '15.4',
        'IP': '10.255.0.2',
        'hostname': 'london_r2',
        'location': '21 New Globe Walk',
        'model': '4451',
        'vendor': 'Cisco'},
 'sw1': {'IOS': '3.6.XE',
         'IP': '10.255.0.101',
         'hostname': 'london_sw1',
         'location': '21 New Globe Walk',
         'model': '3850',
         'vendor': 'Cisco'}}
```

МОДУЛЬ PPRINT

Список списков:

```
In [13]: interfaces = [['FastEthernet0/0', '15.0.15.1', 'YES', 'manual', 'up', 'up
    ...: '], ['FastEthernet0/1', '10.0.1.1', 'YES', 'manual', 'up', 'up'], ['FastE
    ...: thernet0/2', '10.0.2.1', 'YES', 'manual', 'up', 'down']]
    ...:

In [14]: pprint(interfaces)
[['FastEthernet0/0', '15.0.15.1', 'YES', 'manual', 'up', 'up'],
 ['FastEthernet0/1', '10.0.1.1', 'YES', 'manual', 'up', 'up'],
 ['FastEthernet0/2', '10.0.2.1', 'YES', 'manual', 'up', 'down']]
```

МОДУЛЬ PPRINT

Строка:

```
In [18]: tunnel
Out[18]: '\ninterface Tunnel0\n ip address 10.10.10.1 255.255.255.0\n ip mtu 1416\n ip ospf hello-interval 5

In [19]: pprint(tunnel)
('\n'
 'interface Tunnel0\n'
 ' ip address 10.10.10.1 255.255.255.0\n'
 ' ip mtu 1416\n'
 ' ip ospf hello-interval 5\n'
 ' tunnel source FastEthernet1/0\n'
 ' tunnel protection ipsec profile DMVPN\n')
```

ОГРАНИЧЕНИЕ ВЛОЖЕННОСТИ

У функции pprint есть дополнительный параметр depth, который позволяет ограничивать глубину отображения структуры данных.

```
In [3]: result = {
   ...:  'interface Tunnel0': [' ip unnumbered Loopback0',
   ...:   ' tunnel mode mpls traffic-eng',
   ...:   ' tunnel destination 10.2.2.2',
   ...:   ' tunnel mpls traffic-eng priority 7 7',
   ...:   ' tunnel mpls traffic-eng bandwidth 5000',
   ...:   ' tunnel mpls traffic-eng path-option 10 dynamic',
   ...:   ' no routing dynamic'],
   ...:  'ip access-list standard LDP': [' deny   10.0.0.0 0.0.255.255',
   ...:   ' permit 10.0.0.0 0.255.255.255'],
   ...:  'router bgp 100': {' address-family vpnv4': ['  neighbor 10.2.2.2 activat
   ...: e',
   ...:    '  neighbor 10.2.2.2 send-community both',
   ...:    '  exit-address-family'],
   ...:   ' bgp bestpath igp-metric ignore': [],
   ...:   ' bgp log-neighbor-changes': [],
   ...:   ' neighbor 10.2.2.2 next-hop-self': [],
   ...:   ' neighbor 10.2.2.2 remote-as 100': [],
   ...:   ' neighbor 10.2.2.2 update-source Loopback0': [],
   ...:   ' neighbor 10.4.4.4 remote-as 40': []},
   ...:  'router ospf 1': [' mpls ldp autoconfig area 0',
   ...:   ' mpls traffic-eng router-id Loopback0',
   ...:   ' mpls traffic-eng area 0',
   ...:   ' network 10.0.0.0 0.255.255.255 area 0']}
   ...:
```

ОГРАНИЧЕНИЕ ВЛОЖЕННОСТИ

Можно отобразить только ключи, указав глубину равной 1:

```
In [5]: pprint(result, depth=1)
{'interface Tunnel0': [...],
 'ip access-list standard LDP': [...],
 'router bgp 100': {...},
 'router ospf 1': [...]}
```

Скрытые уровни сложенности заменяются . . .

ОГРАНИЧЕНИЕ ВЛОЖЕННОСТИ

Если указать глубину равной 2, отобразится следующий уровень:

```
In [6]: pprint(result, depth=2)
{'interface Tunnel0': [' ip unnumbered Loopback0',
                       ' tunnel mode mpls traffic-eng',
                       ' tunnel destination 10.2.2.2',
                       ' tunnel mpls traffic-eng priority 7 7',
                       ' tunnel mpls traffic-eng bandwidth 5000',
                       ' tunnel mpls traffic-eng path-option 10 dynamic',
                       ' no routing dynamic'],
 'ip access-list standard LDP': [' deny   10.0.0.0 0.0.255.255',
                                 ' permit 10.0.0.0 0.255.255.255'],
 'router bgp 100': {' address-family vpnv4': [...],
                    ' bgp bestpath igp-metric ignore': [],
                    ' bgp log-neighbor-changes': [],
                    ' neighbor 10.2.2.2 next-hop-self': [],
                    ' neighbor 10.2.2.2 remote-as 100': [],
                    ' neighbor 10.2.2.2 update-source Loopback0': [],
                    ' neighbor 10.4.4.4 remote-as 40': []},
 'router ospf 1': [' mpls ldp autoconfig area 0',
                   ' mpls traffic-eng router-id Loopback0',
                   ' mpls traffic-eng area 0',
                   ' network 10.0.0.0 0.255.255.255 area 0']}
```

PFORMAT

pformat - это функция, которая отображает результат в виде строки. Ее удобно использовать, если необходимо записать структуру данных в какой-то файл, например, для логирования.

```
In [15]: from pprint import pformat

In [16]: formatted_result = pformat(result)

In [17]: print(formatted_result)
{'interface Tunnel0': [' ip unnumbered Loopback0',
                       ' tunnel mode mpls traffic-eng',
                       ' tunnel destination 10.2.2.2',
                       ' tunnel mpls traffic-eng priority 7 7',
                       ' tunnel mpls traffic-eng bandwidth 5000',
                       ' tunnel mpls traffic-eng path-option 10 dynamic',
                       ' no routing dynamic'],
 'ip access-list standard LDP': [' deny   10.0.0.0 0.0.255.255',
                                 ' permit 10.0.0.0 0.255.255.255'],
 'router bgp 100': {' address-family vpnv4': ['  neighbor 10.2.2.2 activate',
                                              '  neighbor 10.2.2.2 '
                                              'send-community both',
                                              '  exit-address-family'],
                    ' bgp bestpath igp-metric ignore': [],
                    ' bgp log-neighbor-changes': [],
                    ' neighbor 10.2.2.2 next-hop-self': [],
                    ' neighbor 10.2.2.2 remote-as 100': [],
                    ' neighbor 10.2.2.2 update-source Loopback0': [],
                    ' neighbor 10.4.4.4 remote-as 40': []},
 'router ospf 1': [' mpls ldp autoconfig area 0',
```