Модуль socket

Методы:

gethostname() – возвращает имя локального хоста

getfqdn() – возвращает полное доменное имя локального хоста

gethostbyname('имя хоста') – возвращает IP адрес по имени хоста

gethostbyname_ex('имя хоста') - возвращает кортеж:

- Имя хоста
- Список алиасов
- Список IP-адресов (IPV4)

gethostbyaddr('IP-адрес') - возвращает кортеж:

- Имя хоста
- Список алиасов
- Список IP-адресов (IPV4)

socket() – создает объект сокет

connect((ip,port)) – устанавливает подключение. В качестве аргумента получает кортеж, состоящий из IP-адреса (строка) и номера порта (число)

```
def scan_port(ip,port):
  sockt = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  sockt.settimeout(0.5)
  try:
     connect = sockt.connect((ip,port))
     print('Port :',port,' its open.')
     connect.close()
  except:
     print('Port :',port,' its close.')
```

Модуль netifaces

Методы:

interfaces() – возвращает список интерфейсов
(объект interface), доступных на текущей машине.

ifaddresses(interface) - возвращает информацию
о сетевом интерфейсе в виде словаря:

- Ключ - номер интерфейса (netifaces.**AF_INET – 2**, netifaces.**AF_INET6 – 23,** netifaces.**AF_LINK** - **-1000** - относится к MAC-адресу)
- Значение **– список словарей**.
  Каждый словарь содержит ключи - **'addr'**, **'netmask'**, **'broadcast'**

Пример:

```
Interf0 = netifaces.interfaces()[0]

addresses = netifaces.ifaddresses(interf0)

ipv4_addrs = addresses.get(netifaces.AF_INET, [])

mac_address = netifaces.ifaddresses(interf0)[netifaces.AF_LINK][0]['addr']
```

Модуль os

Включает множество функций для работы с операционной системой. В большинстве своем, поведение функций не зависит от ОС.

Метод **os.system(command)** - исполняет системную команду и возвращает код её завершения (в случае успеха 0).

Модуль platform

https://docs.python.org/3/library/platform.html#module-platform

Модуль pythonping

https://www.ictshore.com/python/python-ping-tutorial/

Пример 1:

```
>>> response_list = pythonping.ping('128.8.8.8', size=40, count=10)
>>> if response_list._responses[0] == 'Request timed out':
        print(0)
else:
        print(1)


1
```

```
import os
import platform


def isUp(hostname,giveFeedback=False):
```

```
if platform.system() == "Windows":
    response = os.system("ping "+hostname+" -n 1")
else:
    response = os.system("ping -c 1 " + hostname)

isUpBool = False
if response == 0:
    if giveFeedback:
        print (hostname, 'is up!')
    isUpBool = True
else:
    if giveFeedback:
        print (hostname, 'is down!')

return isUpBool
```

Модуль subprocess

(https://pythonworld.ru/moduli/modul-subprocess.html)

отвечает за выполнение следующих действий: порождение новых процессов, соединение с потоками стандартного ввода, стандартного вывода, стандартного вывода сообщений об ошибках и получение кодов возврата от этих процессов.

Переменные и Методы:

PIPE - значение, которое может использоваться в качестве аргумента stdin, stdout или stderr

popen() - Создание новых процессов и управление ими

communicate() - взаимодействует с процессом: посылает данные, содержащиеся в input в stdin процесса, ожидает завершения работы процесса, возвращает кортеж данных потока вывода и ошибок.

Пример:

```
from subprocess import PIPE, Popen
res = Popen("ping -n 1 192.168.3.3", shell=True, stdout=PIPE)
out = str(res.communicate()[0].decode("CP866"))

if out.find("100% потерь") == -1:
    print("Связь есть!")
else:
    print("Хост недоступен!")
ipcheck()
```

Модуль ipaddress

входит в стандартную библиотеку Python и упрощает работу с IP-адресами.

Объект ipaddress

Используется для манипулирования IP-адресами IPV4 и IPV6.

Функция **ipaddress.ip_address()** создает объект IPv4Address или IPv6Address

```
ip6 = ipaddress.ip_address('FFFF:9999:2:FDE:257:0:2FAE:112D')
ip4 = ipaddress.ip_address('192.168.0.255')


netaddr, broadcast = map(ip_address, ["2.0.0.0",  "2.15.255.255"])


ip_address('2.1.0.0') in ip_network('2.0.0.0/12')
ip_address('3.1.0.0') in ip_network('2.0.0.0/12')
```

Функции классов **IPv4Address** и **IPv6Address** позволяют создать IP-адрес из константы:

```
ip4 = ipaddress.IPv4Address(3232235521)
ip6 = ipaddress.IPv6Address(3232235521)
```

Чтобы узнать тип IP-адреса используйте функцию type:

```
print(type(ipaddress.ip_address('192.168.0.255')))
```

Методы объекта IP-адрес:

version	Соответствующий номер версии: 4 для IPv4, 6 для IPv6.
exploded	Строковое представление. Ведущие нули никогда не включаются в представление. Поскольку IPv4 не определяет сокращенную запись для адресов с октетами, установленными в ноль, эти два атрибута всегда совпадают с str(addr) для адресов IPv4. Предоставление этих атрибутов облегчает написание кода дисплея, который может обрабатывать адреса как IPv4, так и IPv6.
max_prefixlen	Общее количество бит в адресном представлении для этой версии: 32 для IPv4, 128 для IPv6. Префикс определяет количество старших бит в адресе, которые сравниваются, чтобы определить, является ли адрес частью сети.
packed	Двоичное представление адреса - байтовый объект соответствующей длины (сначала старший значащий октет). Это 4 байта для IPv4 и 16 байтов для IPv6.
reverse_pointer	Имя обратной записи DNS PTR для IP-адреса. Это имя, которое может использоваться для выполнения поиска PTR, а не само разрешенное имя хоста.

is_unspecified	True, если адрес не указан
is_multicast	True, если адрес зарезервирован для многоадресного использования
is_private	True, если адрес выделен для частных сетей
is_global	True, если адрес выделен для публичных сетей.
is_loopback	True, если это адрес обратной связи
is_link_local	True, если адрес зарезервирован для локального использования ссылки
is_reserved	True, если адрес в противном случае зарезервирован IETF.

С IP-объектами можно выполнять различные операции:

```
ip1 = ipaddress.ip_address('10.0.1.1')
ip2 = ipaddress.ip_address('10.0.2.1')

ip1 > ip2
ip2 > ip1
ip1 == ip2
ip1 != ip2

str(ip1)
int(ip1)

ip1 + 5
ip1 - 5
```

Объект сеть – ip_network

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

```
subnet1 = ipaddress.ip_network('80.0.1.0/28')
```

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

broadcast_address()	Широковещательный адрес для сети. Пакеты, отправленные на широковещательный адрес, должны приниматься каждым хостом в сети. `subnet1.broadcast_address`
hosts()	Возвращает итератор для используемых хостов в сети. Используемые хосты - это все IP-адреса, которые принадлежат сети, кроме самого сетевого адреса и сетевого

	широковещательного адреса. Для сетей с длиной маски 31 сетевой адрес и сетевой широковещательный адрес также включаются в результат.
subnets(prefixlen_diff=1, new_prefix=None)	Используется для разбиения сети на подсети. По умолчанию он разбивает сеть на две подсети. Возвращает итератор сетевых объектов. • **prefixlen_diff** - это величина, на которую длина префикса должна быть увеличена. • **new_prefix** - желаемый новый префикс подсетей; он должен быть больше текущего префикса. Только один параметр должен быть установлен.

По IP-адресам в сети можно проходиться в цикле:

```
for ip in subnet1:
        print(ip)
```

Или обращаться к конкретному адресу:

```
subnet1[0]
```

Можно проверять, находится ли IP-адрес в подсети:

```
ip1 in subnet1
```

Объект Интерфейс

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

```
int1 = ipaddress.ip_interface('10.0.1.1/24')
```

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

```
int1.ip
int1.network
int1.netmask
```

Модуль tabulate

Метод **tabulate**

tabulate(data, headers, tablefmt, stralign) – функция, используемая для генерации таблицы.

- **data** – итерируемый объект (список списков, список кортежей, список словарей или словарь с итерируемыми объектами), данные которого оформляются в виде таблицы
- **headers** – определяет строку заголовков для таблицы:
  - 'firstrow' - если первый набор данных - это заголовки
  - 'keys' - если данные в виде списка словарей
  - Имя переменной типа список – содержащий имена столбцов:

```
>>> data = [('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')]
>>>
>>> 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
```

- **tablefmt** - стиль отображения таблицы:
  - "grid" – формат таблицы
  - 'pipe' - формат Markdown
  - 'html' – формат html
- **stralign** – выравнивание столбцов
  - 'center'
  - 'left'`
  - 'right'

Задание к зачету

I. Создайте модуль **zachet.py**, содержащий следующие функции:

1) Функцию **fn_ipaddresses**, которая возвращает информацию о параметрах сетевых интерфейсов локального хоста*
   a. Функция не имеет никаких аргументов
   b. Функция должна возвращать словарь следующего вида:
   {'ipv4': [], 'ipv6': []}.
   Каждый список включает набор кортежей вида
   (IP-address, Net-prefix) для всех интерфейсов узла, на котором запускается код.

Пример:

{'ipv4': [('192.176.1.1', 24), ('192.168.15.2', 32)],
'ipv6': [('fe80::612e:dedc:4939:e55e%16', 64), ('fe80::f89b:3294:3f82:d507%20', 64)]}

2) Функцию **fn_portscan**, которая проверяет доступность портов.
   a. Функция ожидает в качестве аргумента словарь IP-адресов, сформированный **fn_ipaddresses**
   b. Функция должна формировать 2 файла: файл с открытыми портами, файл с закрытыми портами. Каждый файл содержит набор строк вида:
   IP-адрес: **'192.176.1.1', порты: 20, 1433, 1306**
   IP-адрес: **'192.176.15.17', порты: 1433, 1306, 8080**

3) Функцию **fn_ipaccess**, которая проверяет доступность IP-адресов
   a. Функция ожидает в качестве аргумента список IP-адресов
   b. Функция должна возвращать кортеж с двумя списками:

- список доступных IP-адресов
- список недоступных IP-адресов

Для проверки доступности IP-адреса, используйте **ping**

II. Добавьте в модуль функцию **main.**

1. Код функции должен выполняться только при условии, что модуль запускается непосредственно. В случае импорта данного модуля в другие модули код данной функции выполняться не должен.
2. Функция **main** должна вызывать вышеуказанные функции и выводить в консоль следующий результат:
   - Результат функции **fn_ipaddresses** должен быть представлен в виде таблицы следующего вида:

ipv4	ipv6
('192.176.1.1', 24)	('fe80::612e:dedc:4939:e55e%16', 64)
('192.168.15.2', 32)	('fe80::f89b:3294:3f82:d507%20', 64)

- Результат функции **fn_portscan –** имена файлов и количество строк в каждом из них
- Результат функции **fn_ipaccess** – таблица вида:

Доступные	Не доступные
192.168.17.1	192.168.17.11
192.168.17.2	192.168.17.21
192.168.17.3	192.168.17.31
	192.168.17.12
	192.168.17.13

III. Проверьте работоспособность вашего кода:

a. Запустите на выполнение модуль **zachet.py** и убедитесь в получении результатов выполнения функции **main**

b. Создайте модуль **test.py**. Выполните импорт модуля **zachet.py**. Запустите функцию **fn_ipaccess** и выведите в консоль результат ее выполнения. Убедитесь, что код функции main из модуля **zachet.py** не выполняется.