PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

МОДУЛИ

МОДУЛИ

Модуль в Python - это обычный текстовый файл с кодом Python и расширением **.py**. Он позволяет логически упорядочить и сгруппировать код.

Разделение на модули может быть, например, по такой логике:

- разделение данных, форматирования и логики кода
- группировка функций и других объектов по функционалу

Модули хороши тем, что позволяют повторно использовать уже написанный код и не копировать его (например, не копировать когда-то написанную функцию).

ИМПОРТ МОДУЛЯ

ИМПОРТ МОДУЛЯ

В Python есть несколько способов импорта модуля:

- `import module`
- `import module as`
- `from module import object`
- `from module import *`

import module

```
In [1]: dir()
Out[1]:
['In',
 'Out',
 ...
 'exit',
 'get_ipython',
 'quit']

In [2]: import os

In [3]: os.getlogin()
Out[3]: 'natasha'
```

`import module`

Этот способ импорта хорош тем, что объекты модуля не попадают в именное пространство текущей программы. То есть, если создать функцию с именем getlogin(), она не будет конфликтовать с аналогичной функцией модуля os.

`import module as`

Конструкция **import module as** позволяет импортировать модуль под другим именем (как правило, более коротким):

```
In [1]: import subprocess as sp

In [2]: sp.check_output('ping -c 2 -n  8.8.8.8', shell=True)
Out[2]: 'PING 8.8.8.8 (8.8.8.8): 56 data bytes\n64 bytes from 8.8.8.8: icmp_seq=0 ttl=48 time=49.880 ms\n64
```

`from module import object`

Вариант **from module import object** удобно использовать, когда из всего модуля нужны только одна-две функции:

```
In [1]: from os import getlogin, getcwd

In [2]: dir()
Out[2]:
['In',
 'Out',
 ...
 'exit',
 'get_ipython',
 'getcwd',
 'getlogin',
 'quit']
```

from module import object

Их можно вызывать без имени модуля:

```
In [3]: getlogin()
Out[3]: 'natasha'

In [4]: getcwd()
Out[4]: '/Users/natasha/Desktop/Py_net_eng/code_test'
```

`from module import *`

Вариант **from module import *** импортирует все имена модуля в текущее именное пространство:

```
In [1]: from os import *

In [2]: dir()
Out[2]:
['EX_CANTCREAT',
 'EX_CONFIG',
 ...
 'wait',
 'wait3',
 'wait4',
 'waitpid',
 'walk',
 'write']

In [3]: len(dir())
Out[3]: 218
```

`from module import *`

В модуле os очень много объектов, поэтому вывод сокращен. В конце указана длина списка имен текущего именного пространства.

Такой вариант импорта лучше не использовать. При таком импорте по коду непонятно, что какая-то функция взята, например, из модуля os. Это заметно усложняет понимание кода.

СОЗДАНИЕ СВОИХ МОДУЛЕЙ

СОЗДАНИЕ СВОИХ МОДУЛЕЙ

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

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

l3int_template = ['no switchport', 'ip address']
```

Файл sw_data.py:

```
sw1_fast_int = {
                'access':{
                        '0/12':'10',
                        '0/14':'11',
                        '0/16':'17'}}
```

СОЗДАНИЕ СВОИХ МОДУЛЕЙ

```
import sw_int_templates as sw_temp
from sw_data import sw1_fast_int

def generate_access_cfg(sw_dict):
    result = []
    for intf, vlan in sw_dict['access'].items():
        result.append('interface FastEthernet' + intf)
        for command in sw_temp.access_template:
            if command.endswith('access vlan'):
                result.append(' {} {}'.format(command, vlan))
            else:
                result.append(' {}'.format(command))
    return result


print('\n'.join(generate_access_cfg(sw1_fast_int)))
```

СОЗДАНИЕ СВОИХ МОДУЛЕЙ

Результат выполнения скрипта:

```
$ python generate_sw_int_cfg.py
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
interface FastEthernet0/16
 switchport mode access
 switchport access vlan 17
 spanning-tree portfast
 spanning-tree bpduguard enable
```

```
if __name__ == "__main__"
```

`if __name__ == "__main__"`

Иногда скрипт, который вы создали, может выполняться и самостоятельно, и может быть импортирован как модуль другим скриптом.

Добавим ещё один скрипт к предыдущему примеру, который будет импортировать функцию из файла generate_sw_int_cfg.py.

if __name__ == "__main__"

Файл sw_cfg_templates.py с шаблонами конфигурации:

```
basic_cfg = """
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
"""

lines_cfg = """
!
line con 0
 logging synchronous
 history size 100
line vty 0 4
 logging synchronous
 history size 100
 transport input ssh
!
"""
```

if __name__ == "__main__"

В файле generate_sw_cfg.py импортируются шаблоны из sw_cfg_templates.py и функции из предыдущих файлов:

```
from sw_data import sw1_fast_int
from generate_sw_int_cfg import generate_access_cfg
from sw_cfg_templates import basic_cfg, lines_cfg


print(basic_cfg)
print('\n'.join(generate_access_cfg(sw1_fast_int)))
print(lines_cfg)
```

В результате должны отобразиться такие части конфигурации, по порядку: шаблон basic_cfg, настройка интерфейсов, шаблон lines_cfg.

if __name__ == "__main__"

Результат выполнения:

```
$ python generate_sw_cfg.py
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
interface FastEthernet0/16
 switchport mode access
 switchport access vlan 17
 spanning-tree portfast
 spanning-tree bpduguard enable
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
interface FastEthernet0/12
 switchport mode access
 switchport access vlan 10
 spanning-tree portfast
 spanning-tree bpduguard enable
interface FastEthernet0/14
```

`if __name__ == "__main__"`

Когда скрипт импортирует какой-то модуль, всё, что находится в модуле, выполняется. И, так как в данном случае, в файле generate_sw_int_cfg.py есть строка с print, на стандартный поток вывода попадает результат выполнения этого выражения при запуске файла generate_sw_int_cfg.py.

В Python есть специальный прием, который позволяет указать, что какой-то код должен выполняться, только когда файл запускается напрямую.

if __name__ == "__main__"

Файл generate_sw_int_cfg2.py:

```
import sw_int_templates
from sw_data import sw1_fast_int


def generate_access_cfg(sw_dict):
    result = []
    for intf, vlan in sw_dict['access'].items():
        result.append('interface FastEthernet' + intf)
        for command in sw_int_templates.access_template:
            if command.endswith('access vlan'):
                result.append(' {} {}'.format(command, vlan))
            else:
                result.append(' {}'.format(command))
    return result

if __name__ == '__main__':
    print('\n'.join(generate_access_cfg(sw1_fast_int)))
```

`if __name__ == "__main__"`

```
if __name__ == '__main__':
    print('\n'.join(generate_access_cfg(sw1_fast_int)))
```

Переменная `__name__` - это специальная переменная, которая выставляется равной `"__main__"`, если файл запускается как основная программа, и выставляется равной имени модуля, если модуль импортируется.

Таким образом, условие `if __name__ == '__main__'` проверяет, был ли файл запущен напрямую.