PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

КОНТРОЛЬ ХОДА ПРОГРАММЫ

IF/ELIF/ELSE

IF/ELIF/ELSE

Конструкция if/elif/else дает возможность выполнять различные действия в зависимости от условий.

```
In [1]: a = 9

In [2]: if a == 10:
   ...:     print('a равно 10')
   ...: elif a < 10:
   ...:     print('a меньше 10')
   ...: else:
   ...:     print('a больше 10')
   ...:
a меньше 10
```

IF/ELIF/ELSE

Примеры условий:

```
In [7]: 5 > 3
Out[7]: True

In [8]: 5 == 5
Out[8]: True

In [9]: 'vlan' in 'switchport trunk allowed vlan 10,20'
Out[9]: True

In [10]: 1 in [ 1, 2, 3 ]
Out[10]: True

In [11]: 0 in [ 1, 2, 3 ]
Out[11]: False
```

TRUE И FALSE

В Python:

- True (истина)
  - любое ненулевое число
  - любая не пустая строка
  - любой не пустой объект
- False (ложь)
  - 0
  - None
  - пустая строка
  - пустой объект

TRUE И FALSE

Так как пустой список это ложь, проверить пустой ли список можно таким образом:

```
In [12]: list_to_test = [1, 2, 3]

In [13]: if list_to_test:
   ....:     print("В списке есть объекты")
   ....:
В списке есть объекты
```

Тот же результат можно было бы получить таким образом:

```
In [14]: if len(list_to_test) != 0:
   ....:     print("В списке есть объекты")
   ....:
В списке есть объекты
```

ОПЕРАТОРЫ СРАВНЕНИЯ

```
In [3]: 5 > 6
Out[3]: False

In [4]: 5 > 2
Out[4]: True

In [5]: 5 < 2
Out[5]: False

In [6]: 5 == 2
Out[6]: False

In [7]: 5 == 5
Out[7]: True

In [8]: 5 >= 5
Out[8]: True

In [9]: 5 <= 10
Out[9]: True

In [10]: 8 != 10
Out[10]: True
```

ОПЕРАТОР IN

Оператор **in** позволяет выполнять проверку на наличие элемента в последовательности (например, элемента в списке или подстроки в строке):

```
In [8]: 'Fast' in 'FastEthernet'
Out[8]: True

In [9]: 'Gigabit' in 'FastEthernet'
Out[9]: False

In [10]: vlan = [10, 20, 30, 40]

In [11]: 10 in vlan
Out[11]: True

In [12]: 50 in vlan
Out[12]: False
```

ОПЕРАТОР IN

При использовании со словарями условие **in** выполняет проверку по ключам словаря:

```
In [15]: r1 = {
   ....:  'IOS': '15.4',
   ....:  'IP': '10.255.0.1',
   ....:  'hostname': 'london_r1',
   ....:  'location': '21 New Globe Walk',
   ....:  'model': '4451',
   ....:  'vendor': 'Cisco'}

In [16]: 'IOS' in r1
Out[16]: True

In [17]: '4451' in r1
Out[17]: False
```

ОПЕРАТОРЫ AND, OR, NOT

```
In [15]: r1 = {
   ....:  'IOS': '15.4',
   ....:  'IP': '10.255.0.1',
   ....:  'hostname': 'london_r1',
   ....:  'location': '21 New Globe Walk',
   ....:  'model': '4451',
   ....:  'vendor': 'Cisco'}

In [18]: vlan = [10, 20, 30, 40]

In [19]: 'IOS' in r1 and 10 in vlan
Out[19]: True

In [20]: '4451' in r1 and 10 in vlan
Out[20]: False

In [21]: '4451' in r1 or 10 in vlan
Out[21]: True

In [22]: not '4451' in r1
Out[22]: True

In [23]: '4451' not in r1
Out[23]: True
```

ОПЕРАТОР AND

В Python оператор and возвращает не булево значение, а значение одного из операторов.

Если оба операнда являются истиной, результатом выражения будет последнее значение:

```
In [24]: 'string1' and 'string2'
Out[24]: 'string2'

In [25]: 'string1' and 'string2' and 'string3'
Out[25]: 'string3'
```

Если один из операторов является ложью, результатом выражения будет первое ложное значение:

```
In [26]: '' and 'string1'
Out[26]: ''

In [27]: '' and [] and 'string1'
Out[27]: ''
```

ОПЕРАТОР OR

Оператор or, как и оператор and, возвращает значение одного из операторов.

При оценке операндов, возвращается первый истинный операнд:

```
In [28]: '' or 'string1'
Out[28]: 'string1'

In [29]: '' or [] or 'string1'
Out[29]: 'string1'

In [30]: 'string1' or 'string2'
Out[30]: 'string1'
```

Если все значения являются ложью, возвращается последнее значение:

```
In [31]: '' or [] or {}
Out[31]: {}
```

ОПЕРАТОР OR

Важная особенность работы оператора or - операнды, которые находятся после истинного, не вычисляются:

```
In [33]: '' or sorted([44,1,67])
Out[33]: [1, 44, 67]

In [34]: '' or 'string' or sorted([44,1,67])
Out[34]: 'string'
```

ПРИМЕР IF/ELIF/ELSE

Пример скрипта check_password.py, который проверяет длину пароля и есть ли в пароле имя пользователя:

```
# -*- coding: utf-8 -*-

username = input('Введите имя пользователя: ' )
password = input('Введите пароль: ' )

if len(password) < 8:
    print('Пароль слишком короткий')
elif username in password:
    print('Пароль содержит имя пользователя')
else:
    print('Пароль для пользователя {} установлен'.format(username))
```

ПРИМЕР IF/ELIF/ELSE

Проверка скрипта:

```
$ python check_password.py
Введите имя пользователя: nata
Введите пароль: nata1234
Пароль содержит имя пользователя

$ python check_password.py
Введите имя пользователя: nata
Введите пароль: 123nata123
Пароль содержит имя пользователя

$ python check_password.py
Введите имя пользователя: nata
Введите пароль: 1234
Пароль слишком короткий

$ python check_password.py
Введите имя пользователя: nata
Введите пароль: 123456789
Пароль для пользователя nata установлен
```

ЦИКЛ FOR

ЦИКЛ FOR

Цикл for проходится по указанной последовательности и выполняет действия, которые указаны в блоке for.

Примеры последовательностей, по которым может проходиться цикл for:

- строка
- список
- словарь
- итератор range()
- любой другой итератор (например, enumerate())

ПРИМЕР ЦИКЛА FOR

```
In [1]: for letter in 'Test string':
   ...:     print(letter)
   ...:
T
e
s
t

s
t
r
i
n
g
```

ПРИМЕР ЦИКЛА FOR

```
In [2]: for i in range(10):
   ...:     print('interface FastEthernet0/{}'.format(i))
   ...:
interface FastEthernet0/0
interface FastEthernet0/1
interface FastEthernet0/2
interface FastEthernet0/3
interface FastEthernet0/4
interface FastEthernet0/5
interface FastEthernet0/6
interface FastEthernet0/7
interface FastEthernet0/8
interface FastEthernet0/9
```

ПРИМЕР ЦИКЛА FOR

```
In [3]: vlans = [10, 20, 30, 40, 100]
In [4]: for vlan in vlans:
   ...:     print('vlan {}'.format(vlan))
   ...:     print(' name VLAN_{}'.format(vlan))
   ...:
vlan 10
 name VLAN_10
vlan 20
 name VLAN_20
vlan 30
 name VLAN_30
vlan 40
 name VLAN_40
vlan 100
 name VLAN_100
```

ПРИМЕР ЦИКЛА FOR

Когда цикл идет по словарю, он проходится по ключам:

```
In [5]: r1 = {
 'IOS': '15.4',
 'IP': '10.255.0.1',
 'hostname': 'london_r1',
 'location': '21 New Globe Walk',
 'model': '4451',
 'vendor': 'Cisco'}

In [6]: for k in r1:
   ....:     print(k)
   ....:
vendor
IP
hostname
IOS
location
model
```

ПРИМЕР ЦИКЛА FOR

Если необходимо выводить пары ключ-значение в цикле:

```
In [7]: for key in r1:
   ....:     print(key + ' => ' + r1[key])
   ....:
vendor => Cisco
IP => 10.255.0.1
hostname => london_r1
IOS => 15.4
location => 21 New Globe Walk
model => 4451
```

ПРИМЕР ЦИКЛА FOR

В словаре есть специальный метод items, который позволяет проходится в цикле сразу по паре ключ, значение:

```
In [8]: r1.items()
Out[8]: dict_items([('IOS', '15.4'), ('IP', '10.255.0.1'), ('hostname', 'london_r1'),
('location', '21 New Globe Walk'), ('model', '4451'), ('vendor', 'Cisco')])

In [9]: for key, value in r1.items():
   ....:     print(key + ' => ' + value)
   ....:
vendor => Cisco
IP => 10.255.0.1
hostname => london_r1
IOS => 15.4
location => 21 New Globe Walk
model => 4451
```

ПРИМЕР ЦИКЛА FOR

```
In [7]: commands = ['switchport mode access', 'spanning-tree portfast', 'spanning-tree bpduguard enable']
In [8]: fast_int = ['0/1','0/3','0/4','0/7','0/9','0/10','0/11']

In [9]: for intf in fast_int:
   ...:     print('interface FastEthernet {}'.format(intf))
   ...:     for command in commands:
   ...:         print(' {}'.format(command))
   ...:
interface FastEthernet 0/1
 switchport mode access
 spanning-tree portfast
 spanning-tree bpduguard enable
interface FastEthernet 0/3
 switchport mode access
 spanning-tree portfast
 spanning-tree bpduguard enable
interface FastEthernet 0/4
 switchport mode access
 spanning-tree portfast
 spanning-tree bpduguard enable
...
```

ПРИМЕР СОВМЕЩЕНИЯ FOR И IF

Файл generate_access_port_config.py:

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

fast_int = {'access': { '0/12': 10,
                        '0/14': 11,
                        '0/16': 17,
                        '0/17': 150 }}

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

ПРИМЕР СОВМЕЩЕНИЯ FOR И IF

Результат выполнения скрипта:

```
$ python generate_access_port_config.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
```

ИТЕРАТОР ENUMERATE()

Иногда, при переборе объектов в цикле for, нужно не только получить сам объект, но и его порядковый номер. Это можно сделать, создав дополнительную переменную, которая будет расти на единицу с каждым прохождением цикла.

Но, гораздо удобнее это делать с помощью итератора **enumerate()**.

Базовый пример:

```
In [1]: list1 = ['str1', 'str2', 'str3']

In [2]: for position, string in enumerate(list1):
   ...:     print(position, string)
   ...:
0 str1
1 str2
2 str3
```

ИТЕРАТОР ENUMERATE()

enumerate() умеет считать не только с нуля, но и с любого значение, которое ему указали после объекта:

```
In [1]: list1 = ['str1', 'str2', 'str3']

In [2]: for position, string in enumerate(list1, 100):
   ...:     print(position, string)
   ...:
100 str1
101 str2
102 str3
```

ЦИКЛ WHILE

ЦИКЛ WHILE

В цикле while, как и в выражении if, надо писать условие. Если условие истинно, выполняются действия внутри блока while. Но, в отличии от if, после выполнения while возвращается в начало цикла.

При использовании циклов while, необходимо обращать внимание на то, будет ли достигнуто такое состояние, при котором условие цикла будет ложным.

ПРИМЕР ЦИКЛА WHILE

```
In [1]: a = 5

In [2]: while a > 0:
   ...:     print(a)
   ...:     a -= 1 # Эта запись равнозначна a = a - 1
   ...:
5
4
3
2
1
```

ПРИМЕР ЦИКЛА WHILE

Файл check_password_with_while.py:

```
# -*- coding: utf-8 -*-

username = input('Введите имя пользователя: ' )
password = input('Введите пароль: ' )

pass_OK = False

while not pass_OK:
    if len(password) < 8:
        print('Пароль слишком короткий\n')
        password = input('Введите пароль еще раз: ' )
    elif username in password:
        print('Пароль содержит имя пользователя\n')
        password = input('Введите пароль еще раз: ' )
    else:
        print('Пароль для пользователя {} установлен'.format( username ))
        pass_OK = True
```

ПРИМЕР ЦИКЛА WHILE

```
$ python check_password_with_while.py
Введите имя пользователя: nata
Введите пароль: nata
Пароль слишком короткий

Введите пароль еще раз: natanata
Пароль содержит имя пользователя

Введите пароль еще раз: 123345345345
Пароль для пользователя nata установлен
```

BREAK, CONTINUE, PASS

ОПЕРАТОР BREAK

Оператор break позволяет досрочно прервать цикл:

- break прерывает текущий цикл и продолжает выполнение следующих выражений
- если используется несколько вложенных циклов, break прерывает внутренний цикл и продолжает выполнять выражения следующие за блоком
- break может использоваться в циклах for и while

ОПЕРАТОР BREAK

Пример с циклом for:

```
In [1]: for num in range(10):
   ...:     if num < 7:
   ...:         print(num)
   ...:     else:
   ...:         break
   ...:
0
1
2
3
4
5
6
```

ОПЕРАТОР BREAK

Пример с циклом while:

```
In [2]: i = 0
In [3]: while i < 10:
   ...:     if i == 5:
   ...:         break
   ...:     else:
   ...:         print(i)
   ...:         i += 1
   ...:
0
1
2
3
4
```

ОПЕРАТОР BREAK

Пример с циклом while:

```
# -*- coding: utf-8 -*-

username = input('Введите имя пользователя: ' )
password = input('Введите пароль: ' )

while True:
    if len(password) < 8:
        print('Пароль слишком короткий\n')
        password = input('Введите пароль еще раз: ' )
    elif username in password:
        print('Пароль содержит имя пользователя\n')
        password = input('Введите пароль еще раз: ' )
    else:
        print('Пароль для пользователя {} установлен'.format( username ))
        break
```

ОПЕРАТОР CONTINUE

Оператор continue возвращает управление в начало цикла. То есть, continue позволяет "перепрыгнуть" оставшиеся выражения в цикле и перейти к следующей итерации.

ОПЕРАТОР CONTINUE

Пример с циклом for:

```
In [4]: for num in range(5):
   ...:     if num == 3:
   ...:         continue
   ...:     else:
   ...:         print(num)
   ...:
0
1
2
4
```

ОПЕРАТОР CONTINUE

Пример с циклом while:

```
In [5]: i = 0
In [6]: while i < 6:
   ....:     i += 1
   ....:     if i == 3:
   ....:         print("Пропускаем 3")
   ....:         continue
   ....:         print("Это никто не увидит")
   ....:     else:
   ....:         print("Текущее значение: ", i)
   ....:
Текущее значение:  1
Текущее значение:  2
Пропускаем 3
Текущее значение:  4
Текущее значение:  5
Текущее значение:  6
```

ОПЕРАТОР PASS

Оператор pass ничего не делает. Фактически это такая заглушка для объектов.

Например, pass может помочь в ситуации, когда нужно прописать структуру скрипта. Его можно ставить в циклах, функциях, классах. И это не будет влиять на исполнение кода.

Пример использования pass:

```
In [6]: for num in range(5):
   ....:     if num < 3:
   ....:         pass
   ....:     else:
   ....:         print(num)
   ....:
3
4
```

РАБОТА С ИСКЛЮЧЕНИЯМИ

TRY/EXCEPT

Примеры исключений:

```
In [1]: 2/0
---------------------------------------------------
ZeroDivisionError: division by zero

In [2]: 'test' + 2
---------------------------------------------------
TypeError: must be str, not int
```

В данном случае, возникло два исключения: **ZeroDivisionError** и **TypeError**.

TRY/EXCEPT

Для работы с исключениями используется конструкция try/except:

```
In [3]: try:
   ...:     2/0
   ...: except ZeroDivisionError:
   ...:     print("You can't divide by zero")
   ...:
You can't divide by zero
```

TRY/EXCEPT

Конструкция try работает таким образом:

- сначала выполняются выражения, которые записаны в блоке try
- если при выполнения блока try, не возникло никаких исключений, блок except пропускается. И выполняется дальнейший код
- если во время выполнения блока try, в каком-то месте, возникло исключение, оставшаяся часть блока try пропускается
  - если в блоке except указано исключение, которое возникло, выполняется код в блоке except
  - иначе выполнение программы прерывается и выдается ошибка

TRY/EXCEPT

```
In [4]: try:
   ...:     print "Let's divide some numbers"
   ...:     2/0
   ...:     print('Cool!')
   ...: except ZeroDivisionError:
   ...:     print("You can't divide by zero")
   ...:
Let's divide some numbers
You can't divide by zero
```

TRY/EXCEPT

```
# -*- coding: utf-8 -*-

try:
    a = input("Введите первое число: ")
    b = input("Введите второе число: ")
    print("Результат: ", int(a)/int(b))
except ValueError:
    print("Пожалуйста, вводите только числа")
except ZeroDivisionError:
    print("На ноль делить нельзя")
```

TRY/EXCEPT

Примеры выполнения скрипта:

```
$ python divide.py
Введите первое число: 3
Введите второе число: 1
Результат:  3

$ python divide.py
Введите первое число: 5
Введите второе число: 0
Результат:  На ноль делить нельзя

$ python divide.py
Введите первое число: qewr
Введите второе число: 3
Результат:  Пожалуйста, вводите только числа
```

TRY/EXCEPT

Если нет необходимости выводить различные сообщения на ошибки ValueError и ZeroDivisionError, можно сделать так (файл divide_ver2.py):

```
# -*- coding: utf-8 -*-

try:
    a = input("Введите первое число: ")
    b = input("Введите второе число: ")
    print("Результат: ", int(a)/int(b))
except (ValueError, ZeroDivisionError):
    print("Что-то пошло не так...")
```

Проверка:

```
$ python divide_ver2.py
Введите первое число: wer
Введите второе число: 4
Результат:  Что-то пошло не так...

$ python divide_ver2.py
Введите первое число: 5
Введите второе число: 0
Результат:  Что-то пошло не так...
```

TRY/EXCEPT/ELSE

В конструкции try/except есть опциональный блок else. Он выполняется в том случае, если не было исключения.

Например, если необходимо выполнять в дальнейшем какие-то операции с данными, которые ввел пользователь, можно записать их в блоке else (файл divide_ver3.py):

```
# -*- coding: utf-8 -*-

try:
    a = input("Введите первое число: ")
    b = input("Введите второе число: ")
    result = int(a)/int(b)
except (ValueError, ZeroDivisionError):
    print("Что-то пошло не так...")
else:
    print("Результат в квадрате: ", result**2)
```

TRY/EXCEPT/ELSE

Пример выполнения:

```
$ python divide_ver3.py
Введите первое число: 10
Введите второе число: 2
Результат в квадрате:  25

$ python divide_ver3.py
Введите первое число: werq
Введите второе число: 3
Что-то пошло не так...
```

TRY/EXCEPT/FINALLY

Блок finally это еще один опциональный блок в конструкции try. Он выполняется **всегда**, независимо от того, было ли исключение или нет.

Сюда ставятся действия, которые надо выполнить в любом случае. Например, это может быть закрытие файла.

Файл divide_ver4.py с блоком finally:

```
# -*- coding: utf-8 -*-

try:
    a = input("Введите первое число: ")
    b = input("Введите второе число: ")
    result = int(a)/int(b)
except (ValueError, ZeroDivisionError):
    print("Что-то пошло не так...")
else:
    print("Результат в квадрате: ", result**2)
finally:
    print("Вот и сказочке конец, а кто слушал - молодец.")
```

TRY/EXCEPT/FINALLY

Проверка:

```
$ python divide_ver4.py
Введите первое число: 10
Введите второе число: 2
Результат в квадрате:  25
Вот и сказочке конец, а кто слушал - молодец.

$ python divide_ver4.py
Введите первое число: qwerewr
Введите второе число: 3
Что-то пошло не так...
Вот и сказочке конец, а кто слушал - молодец.

$ python divide_ver4.py
Введите первое число: 4
Введите второе число: 0
Что-то пошло не так...
Вот и сказочке конец, а кто слушал - молодец.
```

РАБОТА С ФАЙЛАМИ

ОТКРЫТИЕ ФАЙЛОВ

ОТКРЫТИЕ ФАЙЛОВ

Для начала работы с файлом, его надо открыть.

```
file = open('file_name.txt', 'r')
```

В функции open():

- 'file_name.txt' - имя файла
  - тут можно указывать не только имя, но и путь (абсолютный или относительный)
- 'r' - режим открытия файла

Функция open() создает объект file, к которому потом можно применять различные методы, для работы с ним.

РЕЖИМЫ ОТКРЫТИЯ ФАЙЛОВ

- r - открыть файл только для чтения (значение по умолчанию)
- r+ - открыть файл для чтения и записи
- w - открыть файл для записи
  - если файл существует, то его содержимое удаляется. Если файла нет, создается новый
- w+ - открыть файл для чтения и записи
  - если файл существует, то его содержимое удаляется. Если файла нет, создается новый
- a - открыть файл для дополнение записи. Данные добавляются в конец файла
- a+ - открыть файл для чтения и записи. Данные добавляются в конец файла

ЧТЕНИЕ ФАЙЛОВ

ЧТЕНИЕ ФАЙЛОВ

В Python есть несколько методов чтения файла:

- read() - считывает содержимое файла в строку
- readline() - считывает файл построчно
- readlines() - считывает строки файла и создает список из строк

ЧТЕНИЕ ФАЙЛОВ

Посмотрим как считывать содержимое файлов, на примере файла r1.txt:

```
!
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
!
ip ssh version 2
!
```

ЧТЕНИЕ ФАЙЛОВ. МЕТОД READ()

Метод read() - считывает весь файл в одну строку.

Пример использования метода read():

```
In [1]: f = open('r1.txt')

In [2]: f.read()
Out[2]: '!\nservice timestamps debug datetime msec localtime show-timezone year\nservice timestamps log date

In [3]: f.read()
Out[3]: ''
```

ЧТЕНИЕ ФАЙЛОВ. МЕТОД READLINE()

Построчно файл можно считать с помощью метода readline():

```
In [4]: f = open('r1.txt')

In [5]: f.readline()
Out[5]: '!\n'

In [6]: f.readline()
Out[6]: 'service timestamps debug datetime msec localtime show-timezone year\n'
```

ЧТЕНИЕ ФАЙЛОВ. МЕТОД READLINE()

Но, чаще всего, проще пройтись по объекту file в цикле, не используя методы read...:

```
In [7]: f = open('r1.txt')

In [8]: for line in f:
   ...:     print(line)
   ...:
!

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

!

ip ssh version 2

!
```

ЧТЕНИЕ ФАЙЛОВ. МЕТОД READLINES()

Еще один полезный метод - readlines(). Он считывает строки файла в список:

```
In [9]: f = open('r1.txt')

In [10]: f.readlines()
Out[10]:
['!\n',
 'service timestamps debug datetime msec localtime show-timezone year\n',
 'service timestamps log datetime msec localtime show-timezone year\n',
 'service password-encryption\n',
 'service sequence-numbers\n',
 '!\n',
 'no ip domain lookup\n',
 '!\n',
 'ip ssh version 2\n',
 '!\n']
```

ЧТЕНИЕ ФАЙЛОВ. МЕТОД READLINES()

Если нужно получить строки файла, но без перевода строки в конце, можно воспользоваться методом split и как разделитель, указать символ \n:

```
In [11]: f = open('r1.txt')

In [12]: f.read().split('\n')
Out[12]:
['!',
 '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',
 '!',
 'ip ssh version 2',
 '!',
 '']
```

Обратите внимание, что последний элемент списка - пустая строка.

ЧТЕНИЕ ФАЙЛОВ. МЕТОД READLINES()

Если перед выполнением split(), воспользоваться методом rstrip(), список будет без пустой строки в конце:

```
In [13]: f = open('r1.txt')

In [14]: f.read().rstrip().split('\n')
Out[14]:
['!',
 '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',
 '!',
 'ip ssh version 2',
 '!']
```

SEEK()

До сих пор, файл каждый раз приходилось открывать заново, чтобы снова его считать. Так происходит из-за того, что после методов чтения, курсор находится в конце файла. И повторное чтение возвращает пустую строку.

Чтобы ещё раз считать информацию из файла, нужно воспользоваться методом seek, который перемещает курсор в необходимое положение.

```
In [15]: f = open('r1.txt')

In [16]: print(f.read())
!
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
!
ip ssh version 2
!
```

SEEK()

Если вызывать ещё раз метод read, возвращается пустая строка:

```
In [17]: print(f.read())
```

Но, с помощью метода seek, можно перейти в начало файла (0 означает начало файла):

```
In [18]: f.seek(0)
```

После того, как, с помощью seek, курсор был переведен в начало файла, можно опять считывать содержимое:

```
In [19]: print(f.read())
!
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
!
ip ssh version 2
!
```

ЗАПИСЬ ФАЙЛОВ

ЗАПИСЬ ФАЙЛОВ

При записи, очень важно определиться с режимом открытия файла, чтобы случайно его не удалить:

- append - добавить строки в существующий файл
- write - перезаписать файл
- оба режима создают файл, если он не существует

Для записи в файл используются такие методы:

- write() - записать в файл одну строку
- writelines() - позволяет передавать в качестве аргумента список строк

WRITE()

Метод write ожидает строку, для записи.

Для примера, возьмем список строк с конфигурацией:

```
In [1]: cfg_lines = ['!',
   ...:  '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',
   ...:  '!',
   ...:  'ip ssh version 2',
   ...:  '!']
```

WRITE()

Открытие файла r2.txt в режиме для записи:

```
In [2]: f = open('r2.txt', 'w')
```

Преобразуем список команд в одну большую строку с помощью join:

```
In [3]: cfg_lines_as_string = '\n'.join(cfg_lines)

In [4]: cfg_lines_as_string
Out[4]: '!\nservice timestamps debug datetime msec localtime show-timezone year\nservice timestamps log date
```

WRITE()

Запись строки в файл:

```
In [5]: f.write(cfg_lines_as_string)
```

Аналогично можно добавить строку вручную:

```
In [6]: f.write('\nhostname r2')
```

После завершения работы с файлом, его необходимо закрыть:

```
In [7]: f.close()
```

WRITE()

Так как ipython поддерживает команду cat, можно легко посмотреть содержимое файла:

```
In [8]: cat r2.txt
!
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
!
ip ssh version 2
!
hostname r2
```

WRITELINES()

Метод writelines() ожидает список строк, как аргумент.

```
In [1]: cfg_lines = ['!',
   ...:  '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',
   ...:  '!',
   ...:  'ip ssh version 2',
   ...:  '!']

In [9]: f = open('r2.txt', 'w')

In [10]: f.writelines(cfg_lines)

In [11]: f.close()

In [12]: cat r2.txt
!service timestamps debug datetime msec localtime show-timezone yearservice timestamps log datetime msec loc
```

WRITELINES()

```
In [13]: cfg_lines2 = []

In [14]: for line in cfg_lines:
   ....:     cfg_lines2.append( line + '\n' )
   ....:

In [15]: cfg_lines2
Out[15]:
['!\n',
 'service timestamps debug datetime msec localtime show-timezone year\n',
 'service timestamps log datetime msec localtime show-timezone year\n',
 'service password-encryption\n',
 'service sequence-numbers\n',
 '!\n',
 'no ip domain lookup\n',
 '!\n',
 'ip ssh version 2\n',
```

WRITELINES()

```
In [18]: f = open('r2.txt', 'w')

In [19]: f.writelines(cfg_lines3)

In [20]: f.close()

In [21]: cat r2.txt
!
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
!
ip ssh version 2
!
```

ЗАКРЫТИЕ ФАЙЛОВ

ЗАКРЫТИЕ ФАЙЛОВ

В реальной жизни, для закрытия файлов, чаще всего, используется конструкция with. Её намного удобней использовать, чем закрытия файла явно. Но, так как в жизни можно встретить и метод close, в этом разделе рассматривается его использование.

После завершения работы с файлом, его нужно закрыть. В некоторых случаях, Python может самостоятельно закрыть файл. Но лучше на это не расчитывать и закрывать файл явно.

CLOSE()

```
In [1]: f = open('r1.txt', 'r')
```

Теперь можно считать содержимое:

```
In [2]: print(f.read())
!
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
!
ip ssh version 2
!
```

CLOSE()

У объекта file есть специальный атрибут closed, который позволяет проверить закрыт файл или нет. Если файл открыт, он возвращает False:

```
In [3]: f.closed
Out[3]: False
```

Теперь закрываем файл и снова проверяем closed:

```
In [4]: f.close()

In [5]: f.closed
Out[5]: True
```

CLOSE()

Если попробовать прочитать файл, возникнет исключение:

```
In [6]: print(f.read())
---------------------------------------------------------------------
ValueError                          Traceback (most recent call last)
<ipython-input-53-2c962247edc5> in <module>()
----> 1 print f.read()

ValueError: I/O operation on closed file
```

ИСПОЛЬЗОВАНИЕ TRY/FINALLY ДЛЯ РАБОТЫ С ФАЙЛАМИ

ИСПОЛЬЗОВАНИЕ TRY/FINALLY ДЛЯ РАБОТЫ С ФАЙЛАМИ

С помощью обработки исключений, можно:

- перехватывать исключения, которые возникают, при попытке прочитать несуществующий файл
- закрывать файл, после всех операций, в блоке finally

Если попытаться открыть для чтения файл, которого не существует, возникнет такое исключение:

```
In [7]: f = open('r3.txt', 'r')
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-54-1a33581ca641> in <module>()
----> 1 f = open('r3.txt', 'r')

IOError: [Errno 2] No such file or directory: 'r3.txt'
```

ИСПОЛЬЗОВАНИЕ TRY/FINALLY ДЛЯ РАБОТЫ С ФАЙЛАМИ

С помощью конструкции try/except, можно перехватить это исключение и вывести своё сообщение:

```
In [8]: try:
  ....:     f = open('r3.txt', 'r')
  ....: except IOError:
  ....:     print('No such file')
  ....:
No such file
```

ИСПОЛЬЗОВАНИЕ TRY/FINALLY ДЛЯ РАБОТЫ С ФАЙЛАМИ

А с помощью части finally, можно закрыть файл, после всех операций:

```
In [9]: try:
  ....:     f = open('r1.txt', 'r')
  ....:     print(f.read())
  ....: except IOError:
  ....:     print('No such file')
  ....: finally:
  ....:     f.close()
  ....:
!
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
!
ip ssh version 2
!

In [10]: f.closed
Out[10]: True
```

КОНСТРУКЦИЯ WITH

КОНСТРУКЦИЯ WITH

```
In [1]: with open('r1.txt', 'r') as f:
  ....:     for line in f:
  ....:         print(line)
  ....:
!

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

!

ip ssh version 2

!
```

КОНСТРУКЦИЯ WITH

```
In [2]: with open('r1.txt', 'r') as f:
  ....:     for line in f:
  ....:         print(line.rstrip())
  ....:
!
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
!
ip ssh version 2
!

In [3]: f.closed
Out[3]: True
```

КОНСТРУКЦИЯ WITH

С конструкцией with можно использовать не только такой построчный вариант считывания, все методы, которые рассматривались до этого, также работают:

```
In [4]: with open('r1.txt', 'r') as f:
  ....:     print(f.read())
  ....:
!
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
!
ip ssh version 2
!
```

Конструкция with может использоваться не только с файлами.