Основы Python

Строки

Строки (Strings)

Строки это неизменяемый, упорядоченный тип данных. Строка в Python это последовательность символов, заключенная в кавычки.

```
In [6]: 'Hello'
Out[6]: 'Hello'

In [7]: "Hello"
Out[7]: 'Hello'

In [8]: """
   ...: interface Tunnel0
   ...:  ip address 10.10.10.1 255.255.255.0
   ...:  ip mtu 1416
   ...:  ip ospf hello-interval 5
   ...:  tunnel source FastEthernet1/0
   ...:  tunnel protection ipsec profile DMVPN
   ...: """

Out[8]: '\ninterface 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'
```

Строки (Strings)

То, что строки являются упорядоченным типом данных позволяет нам обращаться к символам в строке по номеру, начиная с нуля:

```
In [14]: string1 = 'interface FastEthernet1/0'

In [15]: string1[0]
Out[15]: 'i'

In [16]: string1[1]
Out[16]: 'n'
```

Нумерация всех символов в строке идет с нуля. Но, если нужно обратиться к какому-то по счету символу, начиная с конца, то можно указывать отрицательные значения (на этот раз с единицы).

```
In [17]: string1[-1]
Out[17]: '0'
```

Строки (Strings)

Кроме обращения к конкретному символу, можно делать срезы строки (срез выполняется по второе число, не включая его):

```
In [14]: string1 = 'interface FastEthernet1/0'

In [18]: string1[0:9]
Out[18]: 'interface'

In [19]: string1[10:22]
Out[19]: 'FastEthernet'
```

Если не указывается второе число, то срез будет до конца строки:

```
In [20]:  string1[10:]
Out[20]: 'FastEthernet1/0'
```

Срезать три последних символа строки:

```
In [21]: string1[-3:]
Out[21]: '1/0'
```

Строка в обратном порядке:

```
In [23]: a[::]
Out[23]: '0123456789'

In [24]: a[::-1]
Out[24]: '9876543210'
```

Полезные методы для работы со строками

По сути, конфигурационный файл это просто текстовый файл и поэтому, при автоматизации, очень часто надо будет работать со строками. Знание различных методов (то есть, действий), которые можно применять к строкам, очень сильно облегчает жизнь.

Конечно, чаще всего, эти методы будут применяться в циклах, при обработке файла. Но уже тут должно быть понятно, что многие вещи можно сделать довольно просто.

Очень важно обращать внимание на то, что часто методы возвращают преобразованную строку. И, значит, надо не забыть присвоить ее какой-то переменной (можно той же).

Это связано с тем, что строка в Python это неизменяемый тип данных.

Полезные методы для работы со строками

Преобразование регистра строки (методы upper(), lower(), swapcase(), capitalize()):

```
In [25]: string1 = 'FastEthernet'

In [26]: string1.upper()
Out[26]: 'FASTETHERNET'

In [27]: string1.lower()
Out[27]: 'fastethernet'

In [28]: string1.swapcase()
Out[28]: 'fASTeTHERNET'

In [29]: string2 = 'tunnel 0'

In [30]: string2.capitalize()
Out[30]: 'Tunnel 0'
```

Метод .lower() особенно полезен при получении данных от пользователя. Так как он позволяет преобразовать данные в нижний регистр и, соответственно не учитывать регистр далее, при обработке введенных данных.

Полезные методы для работы со строками

Метод count() используется для подсчета того, сколько раз символ или подстрока, встречаются в строке:

```
In [33]: string1 = 'Hello, hello, hello, hello'

In [34]: string1.count('hello')
Out[34]: 3

In [35]: string1.count('ello')
Out[35]: 4

In [36]: string1.count('l')
Out[36]: 8
```

Методу find() можно передать подстроку или символ и он покажет на какой позиции находится первый символ подстроки (для первого совпадения):

```
In [37]: string1 = 'interface FastEthernet0/1'

In [38]: string1.find('Fast')
Out[38]: 10

In [39]: string1[string1.find('Fast')::]
Out[39]: 'FastEthernet0/1'
```

Полезные методы для работы со строками

Проверка на то начинается (или заканчивается) ли строка на определенные символы (методы startswith(), endswith()):

```
In [40]: string1 = 'FastEthernet0/1'

In [41]: string1.startswith('Fast')
Out[41]: True

In [42]: string1.startswith('fast')
Out[42]: False

In [43]: string1.endswith('0/1')
Out[43]: True

In [44]: string1.endswith('0/2')
Out[44]: False
```

Замена последовательности символов в строке, на другую последовательность (метод replace()):

```
In [45]: string1 = 'FastEthernet0/1'

In [46]: string1.replace('Fast', 'Gigabit')
Out[46]: 'GigabitEthernet0/1'
```

Полезные методы для работы со строками

Часто, при обработки файла, файл открывается построчно. Но в конце каждый строки, как правило есть какие-то спецсимволы (а могут быть и вначале). Например, перевод строки.

Для того чтобы избавиться от них, очень удобно использовать метод strip():

```
In [47]: string1 = '\n\tinterface FastEthernet0/1\n'

In [48]: print string1

        interface FastEthernet0/1


In [49]: string1
Out[49]: '\n\tinterface FastEthernet0/1\n'

In [50]: string1.strip()
Out[50]: 'interface FastEthernet0/1'
```

Метод strip() убрал спецсимволы и вначале и в конце. Если необходимо убрать символы только слева или только справа, можно использовать, соответственно, методы lstrip() и rstrip().

Полезные методы для работы со строками

Метод split() разбивает строку на части, используя как разделитель какой-то символ (или символы). По умолчанию, в качестве разделителя используется пробел. Но в скобках можно указать любой разделитель, который нужен.

В итоге строка будет разбита на части и представлена в виде частей, которые содержатся в списке.

```
In [51]: string1 = 'switchport trunk allowed vlan 10,20,30,100-200'

In [52]: string1.split()
Out[52]: ['switchport', 'trunk', 'allowed', 'vlan', '10,20,30,100-200']
```

Еще один пример:

```
In [53]: string1 = ' switchport trunk allowed vlan 10,20,30,100-200\n'

In [54]: commands = string1.strip().split()

In [55]: print commands
['switchport', 'trunk', 'allowed', 'vlan', '10,20,30,100-200']

In [56]: vlans = commands[-1].split(',')

In [57]: print vlans
['10', '20', '30', '100-200']
```

Python для сетевых инженеров

Автор курса: Наташа Самойленко
nataliya.samoylenko@gmail.com