Python
2-месяц 5-урок

Тема: Регулярные выражения

Модуль re

Регулярные выражения — специальная последовательность символов, которая помогает сопоставлять или находить строки **python** с использованием специализированного синтаксиса, содержащегося в шаблоне. Регулярные выражения распространены в мире UNIX.

Регулярками называются шаблоны, которые используются для поиска соответствующего фрагмента текста и сопоставления символов.

Грубо говоря, у нас есть input-поле, в которое должен вводиться email-адрес. Но пока мы не зададим проверку валидности введённого email-адреса, в этой строке может оказаться совершенно любой набор символов, а нам это не нужно.

Основные функции модуля re

Основные функции модуля **re**:

- match - ищет последовательность в начале строки
- search - ищет первое совпадение с шаблоном
- findall - ищет все совпадения с шаблоном. Возвращает результирующие строки в виде списка
- finditer - ищет все совпадения с шаблоном. Возвращает итератор
- compile - компилирует регулярное выражение. К этому объекту затем можно применять все перечисленные функции
- fullmatch - вся строка должна соответствовать описанному регулярному выражению

Кроме функций для поиска совпадений, в модуле есть такие функции:

- re.sub - для замены в строках
- re.split - для разделения строки на части

Специальные символы модуля re

Спец. символ	Зачем нужен
.	Задает **один** произвольный символ
[ ]	Заменяет символ из квадратных скобок
-	Задает один символ, которого не должно быть в скобках
[^]	Задает один символ из **не** содержащихся в квадратных скобках
^	Обозначает начало последовательности
$	Обозначает окончание строки
*	Обозначает произвольное число повторений одного символа
?	Обозначает строго одно повторение символа
+	Обозначает один символ, который повторяется несколько раз
|	Логическое **ИЛИ**. Либо выражение до, либо выражение после символа
\	Экранирование. Для использования метасимволов в качестве обычных
( )	Группирует символы внутри
{ }	Указывается число повторений предыдущего символа

Дополнительные конструкции

Также есть дополнительные конструкции, которые позволяют сокращать регулярные выражения:

- **\d** — соответствует любой одной цифре и заменяет собой выражение **[0-9]**;
- **\D** — исключает все цифры и заменяет **[^0-9]**;
- **\w** — заменяет любую цифру, букву, а также знак нижнего подчёркивания;
- **\W** — любой символ кроме латиницы, цифр или нижнего подчёркивания;
- **\s** — соответствует любому пробельному символу;
- **\S** — описывает любой непробельный символ.

Примеры использования

re.match(pattern, string)

Этот метод ищет по заданному шаблону в начале строки. Например, если мы вызовем метод match() на строке «AV Analytics AV» с шаблоном «AV», то он завершится успешно. Но если мы будем искать «Analytics», то результат будет отрицательный:

```
import re

result = re.match(r'AV', 'AV Analytics Vidhya AV')

print result
```

Примеры использования

Искомая подстрока найдена. Чтобы вывести её содержимое, применим метод group() (мы используем «r» перед строкой шаблона, чтобы показать, что это «сырая» строка в Python):

```
result = re.match(r'AV', 'AV Analytics Vidhya AV')

print result.group(0)
```

Результат:

AV

Теперь попробуем найти «Analytics» в данной строке. Поскольку строка начинается на «AV», метод вернет None:

```
result = re.match(r'Analytics', 'AV Analytics Vidhya AV')

print result
```

Результат:

None

Примеры использования

Также есть методы start() и end() для того, чтобы узнать начальную и конечную позицию найденной строки.

```
result = re.match(r'AV', 'AV Analytics Vidhya AV')

print result.start()

print result.end()
```

Результат:

```
0

2
```

Примеры использования

re.search(pattern, string)

Метод похож на match(), но ищет не только в начале строки. В отличие от предыдущего, search() вернёт объект, если мы попытаемся найти «Analytics»:

result = re.search(r'Analytics', 'AV Analytics Vidhya AV')

print result.group(0)

Результат:

Analytics

Метод search() ищет по всей строке, но возвращает только первое найденное совпадение.

Примеры использования

re.findall(pattern, string)

Возвращает список всех найденных совпадений. У метода findall() нет ограничений на поиск в начале или конце строки. Если мы будем искать «AV» в нашей строке, он вернет все вхождения «AV». Для поиска рекомендуется использовать именно findall(), так как он может работать и как re.search(), и как re.match().

```
result = re.findall(r'AV', 'AV Analytics Vidhya AV')

print result
```

Результат:

```
['AV', 'AV']
```

Примеры использования

re.split(pattern, string, [maxsplit=0])

Этот метод разделяет строку по заданному шаблону.

result = re.split(r'y', 'Analytics')

print result

Результат:

['Anal', 'tics']

В примере мы разделили слово «Analytics» по букве «y». Метод split() принимает также аргумент maxsplit со значением по умолчанию, равным 0. В данном случае он разделит строку столько раз, сколько возможно, но если указать этот аргумент, то разделение будет произведено не более указанного количества раз. Давайте посмотрим на примеры Python RegEx:

result = re.split(r'i', 'Analytics Vidhya')

print result

Примеры использования

Результат:

['Analyt', 'cs V', 'dhya'] # все возможные участки.
result = re.split(r'i', 'Analytics Vidhya',maxsplit=1)

print result

Результат:

['Analyt', 'cs Vidhya']

Мы установили параметр maxsplit равным 1, и в результате строка была разделена на две части вместо трех.

Примеры использования

re.sub(pattern, repl, string)

Ищет шаблон в строке и заменяет его на указанную подстроку. Если шаблон не найден, строка остается неизменной.

result = re.sub(r'India', 'the World', 'AV is largest Analytics community of India')

print result

Результат:

'AV is largest Analytics community of the World'

Примеры использования

e.compile(pattern, repl, string)

Мы можем собрать регулярное выражение в отдельный объект, который может быть использован для поиска. Это также избавляет от переписывания одного и того же выражения.

```
pattern = re.compile('AV')

result = pattern.findall('AV Analytics Vidhya AV')

print result

result2 = pattern.findall('AV is largest analytics community of India')

print result2
```

Результат:

```
['AV', 'AV']

['AV']
```

Домашняя работа

Условия Задания :

1. Файл MOCK_DATA, надо проанализировать сначала
2. Используя модуль re Из файла MOCK_DATA вытащить хаотичные данные и упорядочить по пунктам (ФИО , почта , расширение файла, код цвета)
3. Использую метод работы с файлом распределить в новые файлы , в MOCK_DATA есть 4 разделений на ФИО, почту , расширение , код цвета
4. В итоге у вас будет 4 файла , каждый со своими данными

Доп Задание :

1. Сделать все в классах