Практика 2.3.2. Предварительная обработка текста на Python: инструменты и алгоритмы (Часть 2 – Регулярные выражения)

Для дальнейшей работы нам понадобится понятие – регулярные выражения.

2.1. Основы регулярных выражений

Термин «*Регулярные выражения*» является переводом английского словосочетания «Regular expressions». Перевод не очень точно отражает смысл, правильнее было бы «*шаблонные выражения*», т.е. регулярными выражениями называются шаблоны, которые используются для поиска соответствующего фрагмента текста и сопоставления символов. Регулярное выражение – это строка, задающая шаблон поиска подстрок в тексте.

Регулярное выражение состоит из обычных символов и специальных командных последовательностей. Например, \d задаёт любую цифру, а \d+ – задает любую последовательность из одной или более цифр. Работа с регулярными выражениями реализована во всех современных языках программирования.

Таблица 1. Примеры регулярных выражений:

Регулярное выражение	Его смысл
`\d{5}`	Последовательности из 5 цифр `\d` означает любую цифру `{5}` – ровно 5 цифр
`\d\d/\d\d/\d{4}`	Даты в формате ДД/ММ/ГГГГ (и другие варианты, на них похожие, например, 98/76/5432)
`\b\w{3}\b`	Слова в точности из трёх букв `\b` означает границу слова (с одной стороны буква, а с другой – нет) `\w` – любая буква, `{3}` – ровно три буквы
`[-+]?\d+`	Целое число, например, 7, +17, -42, 0013 (возможны ведущие нули) `[-+]? :` либо –, либо +, либо пусто `\d+` – последовательность из 1 или более цифр
`[-+]?(?:\d+(?:\.\d*)?|\.\d+)(?:[eE][-+]?\d+)?`	Действительное число, возможно, в экспоненциальной записи. Например, 0.2, +5.45, -.4, 6e23, -3.17E-14.

С помощью регулярных выражений (шаблонов) в Python мы можем:

– осуществлять поиск, замену и извлечение символов;
– находить слова, которые начинаются на определенные буквы;
– проверять формат телефонного номера или *email*-адреса;
– разбивать строку на подстроки по нескольким разделителям;
– извлекать информацию из *html*-файла и многое другое.

Синтаксис таких выражений в основном стандартизирован, так что их следует понять, чтобы использовать в любом языке программирования. Но регулярные выражения не всегда оптимальны, и для простых операций часто достаточно встроенных в Python функций.

2.2. Синтаксис *Regular Expressions* (RegEx)

Рассмотрим регулярные выражения в Python, начиная с синтаксиса и заканчивая примерами использования.

В Python для работы с регулярными выражениями (шаблонами) есть модуль `re`. Его нужно просто импортировать:

```
import re
```

Рассмотрим наиболее популярные методы, которые предоставляет модуль:

- `re.match()`
- `re.search()`
- `re.findall()`
- `re.split()`
- `re.sub()`
- `re.compile()`

1) Метод `re.match(pattern, string)`

Этот метод ищет заданный шаблон (**`pattern`**) в начале строки (**`string`**). 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'ИИ', 'ИИ Искусственный Интеллект ИИ')
print(result)
```

Результат:

```
<re.Match object; span=(0, 2), match='ИИ'>
```

Это сообщение, что искомая подстрока найдена. Чтобы вывести её содержимое, применяют метод `group()`. При этом в круглых скобках ставится символ `r` перед строкой шаблона, чтобы показать, что это «сырая» строка в Python ("сырые" строки подавляют экранирование. **Экранированные последовательности** – это последовательности, которые начинаются с символа "\" за которым следует один или более символов.):

```
result = re.match(r'ИИ', 'ИИ Искусственный Интеллект ИИ')
print(result.group(0))
```

Результат:

```
ИИ
```

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

```
result = re.match(r'Интеллект', 'ИИ Искусственный Интеллект ИИ')
print(result)
```

Результат:
```
None
```

2) Метод `re.search(pattern, string)`

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

```
result = re.search(r'Интеллект', 'ИИ Искусственный Интеллект ИИ')
print(result.group(0))
```

Результат:
```
Интеллект
```

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

3) Метод `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 «ИИ» в нашей строке, он вернет все вхождения «ИИ». Для поиска рекомендуется использовать именно `findall()`, так как он может работать и как `re.search()`, и как `re.match()`.

```
result = re.findall(r'ИИ', 'ИИ Искусственный Интеллект ИИ')
print(result)
```

Результат:
```
['ИИ', 'ИИ']
```

4) Метод `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 «у». 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.

```
result = re.split(r'н', 'Искусственный Интеллект')
print(result)
```

Результат:

```
[' Искусстве', '', 'ый И', 'теллект'] # все возможные участки.
```

```
result = re.split(r'т', 'Искусственный Интеллект',maxsplit
= 2)
print(result)
```

Результат:

```
['Искусс', 'венный Ин', 'еллект']
```

Мы установили параметр `maxsplit` равным 2, и в результате строка была разделена на три части.

5) Метод `re.sub(pattern, repl, string)`

Ищет шаблон (`pattern`) в строке и заменяет его на указанную подстроку (`repl`). Если шаблон не найден, строка остается неизменной.

```
result = re.sub(r' 'Алтайского края', 'России', 'Слушатели
ДПО – самые продвинутые слушатели Алтайского края!')
print(result)
```

Результат:

```
Слушатели ДПО - самые продвинутые слушатели России!
```

6) Метод `re.compile(pattern, repl, string)`

Мы можем выделить регулярное выражение в отдельный объект и использовать его для поиска в разных строках. Это избавляет от переписывания одного и того же выражения.

```
pattern = re.compile('ИИ')
result = pattern.findall('ИИ Искусственный ИИнтеллект ИИ')
print(result)
result2 = pattern.findall('Развитие ИИ – стратегическая
линия развития России')
print(result2)
```

Результат:

```
['ИИ', 'ИИ', 'ИИ']
['ИИ']
```

2.3. Использование специальных символов в регулярных выражениях

До сих пор мы рассматривали поиск определенной последовательности символов. Но что, если у нас нет определенного шаблона, и нам надо вернуть набор символов из строки, отвечающий определенным правилам? Такая задача часто стоит при извлечении информации из строк. Это можно сделать, написав выражение с использованием специальных символов – конструкций, которые позволяют сокращать регулярные выражения. Наиболее часто используемые из них приведены в таблице 2:

Таблица 2. Описание специальных символов

Спец. символ	Описание
`.`	Задает один произвольный символ, кроме новой строки `\n`.
`?`	Обозначает 0 или 1 вхождение шаблона слева
`+`	1 и более вхождений шаблона слева
`*`	0 и более вхождений шаблона слева
`\w`	Любая цифра или буква или знак нижнего подчеркивания (\W – все, кроме буквы или цифры и знака нижнего подчеркивания)
`\d`	Любая цифра [0-9] (`\D` – все, кроме цифры – нецифровой символ или знака подчеркивания)
`\s`	Любой пробельный символ (`\S` – любой непробельный символ)
`\b`	Граница слова
`[..]`	Один из символов в скобках (`[^..]` – любой символ, кроме тех, что в скобках)
`\`	Экранирование специальных символов (`\.` означает точку или `\+` – знак «плюс»)
`^` и `$`	Начало и конец строки соответственно
`{n,m}`	От `n` до `m` вхождений (`{,m}` – от 0 до `m`)
`a|b`	Соответствует `a` или `b`
`()`	Группирует символы и возвращает найденный текст
`\t, \n, \r`	Символ табуляции, новой строки и возврата каретки соответственно

Рассмотрим примеры практического применения Python регулярных выражений и специальных символов.

Пример 1. Вернуть первое слово из строки

1.1. Сначала попробуем вытащить каждый символ, используя `'.'`

```
result = re.findall(r'.', ' Развитие искусственного
интеллекта – стратегическая линия развития России ')
print(result)
```

Результат:

```
[' ', 'Р', 'а', 'з', 'в', 'и', 'т', 'и', 'е', ' ', 'и', 'с', 'к', 'у', 'с',
'с', 'т', 'в', 'е', 'н', 'н', 'о', 'г', 'о', ' ', 'и', 'н', 'т', 'е', 'л',
'л', 'е', 'к', 'т', 'а', ' ', '–', ' ', 'с', 'т', 'р', 'а', 'т', 'е', 'г',
'и', 'ч', 'е', 'с', 'к', 'а', 'я', ' ', 'л', 'и', 'н', 'и', 'я', ' ', 'р',
'а', 'з', 'в', 'и', 'т', 'и', 'я', ' ', 'Р', 'о', 'с', 'с', 'и', 'и', ' ']
```

1.2. Для того, чтобы в конечный результат не попал пробел, используем вместо `.` `\w`.

```
result = re.findall(r'\w', ' Развитие искусственного
интеллекта – стратегическая линия развития России ')
print(result)
```

Результат:

```
['Р', 'а', 'з', 'в', 'и', 'т', 'и', 'е', 'и', 'с', 'к', 'у', 'с', 'с', 'т',
'в', 'е', 'н', 'н', 'о', 'г', 'о', 'и', 'н', 'т', 'е', 'л', 'л', 'е', 'к',
'т', 'а', 'с', 'т', 'р', 'а', 'т', 'е', 'г', 'и', 'ч', 'е', 'с', 'к', 'а',
'я', 'л', 'и', 'н', 'и', 'я', 'р', 'а', 'з', 'в', 'и', 'т', 'и', 'я', 'Р',
'о', 'с', 'с', 'и', 'и']
```

1.3. Теперь попробуем достать каждое слово (используя `*` или `+`)

```
result = re.findall(r'\w*', ' Развитие искусственного
интеллекта – стратегическая линия развития России ')
print(result)
```

Результат:

```
['', 'Развитие', '', 'искусственного', '', 'интеллекта', '', '', '',
'стратегическая', '', 'линия', '', 'развития', '', 'России', '', '']
```

В результат попали пробелы, так как `*` означает «ноль или более символов».

1.4. Для того, чтобы убрать пробелы, используем `+`:

```
result = re.findall(r'\w+', ' Развитие искусственного
интеллекта – стратегическая линия развития России ')
print(result)
```

Результат:

```
['Развитие', 'искусственного', 'интеллекта', 'стратегическая',
'линия', 'развития', 'России']
```

1.5. Теперь вытащим первое слово, используя `^`:

```
result = re.findall(r'^\w+', 'Развитие искусственного
интеллекта – стратегическая линия развития России')
print(result)
```

Результат:
```
['Развитие']
```

1.6. Если использовать `$` вместо `^`, то получим последнее слово, а не первое:

```
result = re.findall(r'\w+$', 'Развитие искусственного
интеллекта – стратегическая линия развития России')
print(result)
```

Результат:
```
['России']
```

Пример 2. Вернуть первые два символа каждого слова

2.1. Вариант 1: используя `\w`, вытащить два последовательных символа, кроме пробельных, из каждого слова:

```
result = re.findall(r'\w\w', 'Развитие искусственного
интеллекта – стратегическая линия развития России')
print(result)
```

Результат:
```
['Ра', 'зв', 'ит', 'ие', 'ис', 'ку', 'сс', 'тв', 'ен', 'но', 'го', 'ин', 'те',
'лл', 'ек', 'та', 'ст', 'ра', 'те', 'ги', 'че', 'ск', 'ая', 'ли', 'ни', 'ра',
'зв', 'ит', 'ия', 'Ро', 'сс', 'ии']
```

2.2. Вариант 2: вытащить первые два последовательных символа, используя символ границы слова (`\b`):

```
result = re.findall(r'\b\w.', 'Развитие искусственного
интеллекта – стратегическая линия развития России')
print(result)
```

Результат:
```
['Ра', 'ис', 'ин', 'ст', 'ли', 'ра', 'Ро']
```

Пример 3. Извлечь дату из строки

3.1. Используем `\d` для извлечения цифр.

```
result = re.findall(r'\d{2}-\d{2}-\d{4}', 'Amit 34-3456 12-
05-2007, XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-2009')
print(result)
```

Результат:
```
['12-05-2007', '11-11-2011', '12-01-2009']
```

3.2. Для извлечения только года нам опять помогут скобки:

```
result = re.findall(r'\d{2}-\d{2}-(\d{4})', 'Amit 34-3456
12-05-2007, XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-
2009')
print(result)

Результат:
['2007', '2011', '2009']
```

3.3. Пример с датой рождения

```
import re
born = "05-03-1987 # Дата рождения"
# Удалим комментарий из строки
dob = re.sub(r'#.*$', "", born)
print("Дата рождения:", dob)
# Заменим дефисы на точки
f_dob = re.sub(r'-', ".", born)
print(f_dob)
# Запускаем скрипт и получаем вывод:

[Out]: Дата рождения: 05-03-1987
05.03.1987 # Дата рождения
```

Переходите к выполнению практического задания 2.3.2.