Поисковые задачи

- выяснить, содержится ли регулярное выражение в строке
- найти часть строки, подходящую под регулярное выражение
- заменить часть строки по регулярному выражению

Классы символов

- \s — любой «пробелообразный» символ (пробел, табуляция, новая строка)
- \w — любая буква
- \d — любая цифра

Обратные ссылки

- обратная ссылка (backreference) — это способ использовать уже найденную группу в дальнейшем поиске

- значение группы номер *x* внутри регулярного выражения обозначается как \x

- максимум 99 групп

Пример

```
([аеёиоуыэюя]).*\\1.*\\1
```

- так можно найти слова с 3 одинаковыми гласными

Пример

```
(а?)б\\1
```

- сработает на строке «б»

```
(а)?б\\1
```

- не сработает на строке «б»

Замена

- функция **sub:**

```
m = re.sub(u'[^;]+', u'', s)
```

Петя;1991;123-45-67;petya@petya.ru

;;;

Замена

- замена кошек на собак

```
m = re.sub(u'кошк', u'собак', s)
```

Эти кошки гуляют сами по себе.

Эти собаки гуляют сами по себе.

— А «кошек»?!

Замена

- замена кошек на собак

```
m = re.sub(u'коше?к', u'собак', s)
```

Эти кошки крупнее тех кошек.

Эти собаки крупнее тех собак.

— А «Кошки»?!

Замена

- замена кошек на собак

```
m = re.sub(u'[Кк]оше?к', u'собак')
```

Кошки мельче собак.

собаки мельче собак.

— А «лукошки»? А маленькая буква в начале предложения?!

Замена

- замена повторяющихся слов

```
m = re.sub(u'(\\w+) \\1', u'\\1', s,
   flags = re.U)
```

Я думаю, что что он прав.

Я думаю, что он прав.

не нервничайте перед экзаменом

Замена

- замена повторяющихся слов

```
m = re.sub(u'(^|[^\\w])
  (\\w+) \\2([^\\w]|$)',
 u'\\1\\2\\3', s, flags = re.U)
```

не нервничай, всё всё будет хорошо.

не нервничай, всё будет хорошо.

ОК.

Замена

- удвоение всех слов

```
m = re.sub(u'(\\w+)', u'\\1 \\1', s,
  flags = re.U)
```

Привет, я питон!

Привет Привет, я я питон питон!

Замена

- контекстное удаление: ABC → AC

```
m = re.sub(u'([ие])(.)([ие])',
  u'\\1\\3', s, flags = re.U)
```

Привет, я питон!

Приет, я пион!

Замена

- убрать все HTML-тэги из документа

```
m = re.sub(u'<.*?>', u'', s, flags =
  re.U)
```

<html> Привет! </html>

Привет!

<img alt="x > y" src="...">

Язык регулярных выражений

- ^ и $ — начало и конец строки
- [^абв] — **инвертированный** класс символов (все символы, кроме перечисленных)
- внутри квадратных скобок не работают круглые скобки, *, . и т. п.

Язык регулярных выражений

- ^ и $ могут использоваться много раз:

дела(ть$|ющ.{,3}$|е)

- {,} = *, {1,} = +, {,1} = ?

Язык регулярных выражений

- {5} — предыдущее выражение должно повториться ровно 5 раз
- {3,5} — предыдущее выражение должно повториться от 3 до 5 раз
- {,5} — max 5 раз, {3,} — min 3 раза

Язык регулярных выражений

- *, + и {} не означают повторение абсолютно одинаковых фрагментов:

```
([аеёиоуыэюя].*){3}
```

Найдутся: «карнавал», «бегемот», «самовар» и т. п.

Пример

([бгдзкпст][^бгдзкпст]){3,}

- Кошка, или домашняя кошка (лат. Felis silvestris catus) — домашнее животное, одно из наиболее популярных (наряду с собакой) «животных-компаньонов»

Жадные квантификаторы

- *, +, ?, {min,max} — **жадные** квантификаторы (greedy)
- они пытаются съесть как можно больше символов:

.*([аеёиоуыэюя])

собака

Нежадные квантификаторы

- *?, +?, ??, {min,max}? — **нежадные** квантификаторы
- они пытаются съесть как можно меньше символов:

.*?([аеёиоуыэюя])

собака

Использование нежадности

- как найти текст в скобках?
- вариант 1:

```
\\([^\\)]*\\)
```

- вариант 2:

```
\\(.*?\\)
```

Построение выражения из частей

- выражения можно составлять из частей
- пример: поиск слов с ≥ 4 гласными

```
v = u'[аеёиоуыэюя]'
m = re.search((v + u'.*') * 4, word)
```