

Регулярные выражения
ЛКШ.2014.Зима, параллель В'+

Антон Полднев

30 декабря 2014 г.,
3 января 2015 г.

Что такое регулярное выражение?

Регулярное выражение — строка-шаблон, представляющая собой формальную запись для множества строк.

Примеры:

- `Winter` — соответствует лишь строке `Winter`
- `Goo+gle` — соответствует строкам `Google`, `Gooogle`, `Gooooogle` ...

Поддержка в языках программирования

- *Python*: модуль `re`
- *C++*: библиотека `<regex>` (C++11, gcc \geq 4.9)
- *Perl, Javascript*: встроена в язык

Пример: поиск совпадения с шаблоном

Пытаемся найти подстроку,
соответствующую шаблону `l[aio]st`,
в строках `listing`, `the last day`
и `Lost`:

- `listing`
- `the last day`
- `Lost`

Пример: поиск совпадения с шаблоном

Пытаемся найти подстроку,
соответствующую шаблону `l[aio]st`,
в строках `listing`, `the last day`
и `Lost`:

- listing
- the last day
- Lost

Пример: поиск совпадения с шаблоном

Пытаемся найти подстроку,
соответствующую шаблону `l[aio]st`,
в строках `listing`, `the last day`
и `Lost`:

- listing
- the last day
- Lost

Пример: поиск совпадения с шаблоном

Пытаемся найти подстроку,
соответствующую шаблону `l[aio]st`,
в строках `listing`, `the last day`
и `Lost`:

- listing
- the last day
- `Lost`: нет совпадений, так как регистр имеет значение

Пример: поиск совпадения с шаблоном (Python)

```
>>> import re
>>> re.search(r'l[aio]st', 'listing')
<_sre.SRE_Match object at 0x02AB60C8>
>>> re.search(r'l[aio]st', 'the last day')
<_sre.SRE_Match object at 0x02AB6090>
>>> re.search(r'l[aio]st', 'Lost')
>>> bool(re.search(r'l[aio]st', 'listing'))
True
>>> bool(re.search(r'l[aio]st', 'the last day'))
True
>>> bool(re.search(r'l[aio]st', 'Lost'))
False
```

Пример: поиск совпадения с шаблоном (C++)

```
#include <iostream>
#include <regex>

using namespace std;

int main() {
    regex r(R"(l[aio]st)");

    if (regex_match("listing", r)) {
        cout << "Case 1: matched" << endl; // printed
    }
    if (regex_match("the last day", r)) {
        cout << "Case 2: matched" << endl; // printed
    }
    if (regex_match("Lost", r)) {
        cout << "Case 3: matched" << endl; // not printed
    }
    return 0;
}
```

Пример: поиск совпадения с шаблоном (Javascript)

```
>>> /l[aio]st/.test('listing')
true
>>> r = /l[aio]st/
/l[aio]st/
>>> r.test('listing')
true
>>> r.test('the last day')
true
>>> r.test('Lost')
false
```

Ещё примеры на Python

```
>>> import re
>>> bool(re.search(r'(?i)l[aio]st', 'Lost'))
True
>>> bool(re.search(
...     r'l[aio]st',
...     'Lost',
...     flags = re.I
... ))
True
```

Ещё примеры на Python

```
>>> import re
>>> bool(re.search(r'(?i)l[aio]st', 'Lost'))
True
>>> bool(re.search(
...     r'l[aio]st',
...     'Lost',
...     flags = re.I
... ))
True
>>> re.findall(r'l[aio]st', 'Last lost lists')
```

Ещё примеры на Python

```
>>> import re
>>> bool(re.search(r'(?i)l[aio]st', 'Lost'))
True
>>> bool(re.search(
...     r'l[aio]st',
...     'Lost',
...     flags = re.I
... ))
True
>>> re.findall(r'l[aio]st', 'Last lost lists')
['lost', 'list']
```

Специальные символы

- Проблема: $1+2$ не найдётся в строке $41+29$, так как $+$ имеет специальное значение

Специальные символы

- Проблема: `1+2` не найдётся в строке `41+29`, так как `+` имеет специальное значение
- Экранируем специальные символы с помощью `\`

Специальные символы

- Проблема: `1+2` не найдётся в строке `41+29`, так как `+` имеет специальное значение
- Экранируем специальные символы с помощью `\`
- `1\+2` найдётся в строке `41+29`

Специальные символы

- Проблема: `1+2` не найдётся в строке `41+29`, так как `+` имеет специальное значение
- Экранируем специальные символы с помощью `\`
- `1\+2` найдётся в строке `41+29`
- Специальные символы: `(,), [,], |, \, /, ?, *, +, ^, $, ., {, }`

Тонкости экранирования

```
>>> print('\t')
```

Тонкости экранирования

```
>>> print('\t')
```

Тонкости экранирования

```
>>> print('\t')
```

```
>>> print('\\t')
```

```
\t
```

Как получить регулярное выражение, соответствующее одному символу `\`?

Тонкости экранирования

```
>>> print('\t')
```

```
>>> print('\\t')
```

```
\t
```

Как получить регулярное выражение, соответствующее одному символу `\`?

```
>>> print('\\\\')
```

```
\\
```

Слишком много бэкслешей!

Если использовать сырые строки, `\` не будет специальным символом:

```
>>> print(r'\t')
\t
>>> print(r'\\t')
\\t
>>> print(r'\\\\\\')
\\\\\\
```

Чему соответствует регулярное выражение, полученное из строки `r'\\\\\\'` ?

Сырые строки в C++

```
#include <iostream>
#include <regex>

using namespace std;

int main() {
    cout << R"(\t)" << endl;    // \t
    cout << R"(\t)" << endl;    // \t
    cout << R"(\t)" << endl;    // \t

    return 0;
}
```

Якоря начала и конца строки: \wedge и $\$$

- \wedge в начале регулярного выражения означает, что искать нужно в начале строки
- Где найдётся $\wedge eni$? `eniki`, `beniki`

Якоря начала и конца строки: ^ и \$

- `^` в начале регулярного выражения означает, что искать нужно в начале строки
- Где найдётся `^eni`? `eniki`, `beniki`

Якоря начала и конца строки: ^ и \$

- `^` в начале регулярного выражения означает, что искать нужно в начале строки
- Где найдётся `^eni`? `eniki`, `beniki`

Якоря начала и конца строки: ^ и \$

- `^` в начале регулярного выражения означает, что искать нужно в начале строки
- Где найдётся `^eni`? `eniki`, `beniki`
- `$` в конце регулярного выражения означает, что искать нужно в конце строки
- Где найдётся `s$`? `ship`, `ships`

Якоря начала и конца строки: ^ и \$

- `^` в начале регулярного выражения означает, что искать нужно в начале строки
- Где найдётся `^eni`? `eniki`, `beniki`
- `$` в конце регулярного выражения означает, что искать нужно в конце строки
- Где найдётся `s$`? `ship`, `ships`

Якоря начала и конца строки: ^ и \$

- `^` в начале регулярного выражения означает, что искать нужно в начале строки
- Где найдётся `^eni`? `eniki`, `beniki`
- `$` в конце регулярного выражения означает, что искать нужно в конце строки
- Где найдётся `s$`? `ship`, `ships`

Якоря начала и конца строки: \wedge и $\$$

- Как с помощью функции поиска подстроки, соответствующей регулярному выражению, просто проверить, соответствует ли *вся* строка шаблону?

Якоря начала и конца строки: `^` и `$`

- Как с помощью функции поиска подстроки, соответствующей регулярному выражению, просто проверить, соответствует ли *вся* строка шаблону?
- Используем `^` и `$`

Якоря начала и конца строки: \wedge и $\$$

- Как с помощью функции поиска подстроки, соответствующей регулярному выражению, просто проверить, соответствует ли *вся* строка шаблону?
- Используем \wedge и $\$$
- Ищем $\wedge str \$$: `str`, `substr`

Якоря начала и конца строки: ^ и \$

- Как с помощью функции поиска подстроки, соответствующей регулярному выражению, просто проверить, соответствует ли *вся* строка шаблону?
- Используем ^ и \$
- Ищем `^str$`: `str`, `substr`

Якоря начала и конца строки: ^ и \$

- Как с помощью функции поиска подстроки, соответствующей регулярному выражению, просто проверить, соответствует ли *вся* строка шаблону?
- Используем ^ и \$
- Ищем `^str$`: `str`, `substr`

Любой символ: .

- . в регулярном выражении означает один любой символ (как правило, кроме перевода строки `\n`)
- Где найдётся `b.d`?

abcde

1b8d3

abde

abccde

bbdd

Любой символ: .

- . в регулярном выражении означает один любой символ (как правило, кроме перевода строки `\n`)
- Где найдётся `b.d`?

`abcde`, `1b8d3`, `abde`, `abccde`, `bbdd`

Любой символ: .

- . в регулярном выражении означает один любой символ (как правило, кроме перевода строки `\n`)
- Где найдётся `b.d`?

`abcde`, `1b8d3`, `abde`, `abccde`, `bbdd`

Любой символ: .

- . в регулярном выражении означает один любой символ (как правило, кроме перевода строки `\n`)
- Где найдётся `b.d`?

`abcde`

`1b8d3`

`abde`

`abccde`

`bbdd`

Любой символ: .

- . в регулярном выражении означает один любой символ (как правило, кроме перевода строки `\n`)
- Где найдётся `b.d`?

`abcde`

`1b8d3`

`abde`

`abccde`

`bbdd`

Любой символ: .

- `.` в регулярном выражении означает один любой символ (как правило, кроме перевода строки `\n`)
- Где найдётся `b.d`?

`abcde`

`1b8d3`

`abde`

`abccde`

`bbdd`

Любой символ: .

- . в регулярном выражении означает один любой символ (как правило, кроме перевода строки `\n`)
- Где найдётся `b.d`?

`abcde`

`1b8d3`

`abde`

`abccde`

`bbdd`

Альтернатива: |, круглые скобки

- Хотим, чтобы регулярное выражение соответствовало лишь двум строкам:

`test` и `twist`

Альтернатива: |, круглые скобки

- Хотим, чтобы регулярное выражение соответствовало лишь двум строкам:
`test` и `twist`
- Используем символ `|`: `test|twist`

Альтернатива: |, круглые скобки

- Хотим, чтобы регулярное выражение соответствовало лишь двум строкам:
`test` и `twist`
- Используем символ `|`: `test|twist`
- Для группировки частей можно использовать круглые скобки:
`t(e|wi)st` соответствует тем же строкам

Альтернатива: |, круглые скобки

- Где найдётся `twist|test`?

`twist`, `antitest`

Альтернатива: |, круглые скобки

- Где найдётся `twist|test`?
`twist`, `antitest`

Альтернатива: |, круглые скобки

- Где найдётся `twist|test`?
twist, antitest

Альтернатива: |, круглые скобки

- Где найдётся `twist|test`?
`twist`, `antitest`
- Хотим, чтобы находилось соответствие только целой строке → `^twist|test$`

Альтернатива: |, круглые скобки

- Где найдётся `twist|test`?
- twist, antitest
- Хотим, чтобы находилось соответствие только целой строке → `^twist|test$`
- В чём проблема?

Альтернатива: |, круглые скобки

- Где найдётся `twist|test`?
`twist`, `antitest`
- Хотим, чтобы находилось соответствие только целой строке → `^twist|test$`
- В чём проблема?
- `antitest`

Альтернатива: |, круглые скобки

- Где найдётся `twist|test`?
`twist`, `antitest`
- Хотим, чтобы находилось соответствие только целой строке → `^twist|test$`
- В чём проблема?
- `antitest`
- Исправляем с помощью скобок:
`^(twist|test)$`

Квантификаторы ?, *, +, {}

Квантификатор после группы символов определяет, сколько раз она может повторяться:

- **?** : 0 или 1 раз
- ***** : 0 и более раз
- **+** : 1 и более раз
- **{2,5}** : от 2 до 5 раз
- **{2,}** : 2 и более раз
- **{,5}** : от 0 до 5 раз
- **{2}** : ровно 2 раза

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`:

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`: Yandex,

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`: Yandex, Yooondex,

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`: Yandex, Yooondex,
Yoaondex

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`: Yandex, Yooondex,
Yoaondex

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`: Yandex, Yooondex,
Yoaondex
- `^Y.*ndex$`: Yndex, Yundex,
Y1i883ef-ndex, Yande

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`: Yandex, Yooondex, Yoaondex
- `^Y.*ndex$`: Yndex, Yundex, Y1i883ef-ndex, Yande
- `Y.*ndex`: 0Yandexandex0

Квантификаторы ?, *, +, {}

- `^Goo+gle$`: Gogle, Google, Goooogle
- `^G(oo)?gle$`: Gogle, Google, Ggle
- `^Y(a|o)+ndex$`: Yandex, Yooondex, Yoaondex
- `^Y.*ndex$`: Yndex, Yundex, Y1i883ef-ndex, Yande
- `Y.*ndex`: 0Yandexandex0
- Как видим, квантификаторы по умолчанию *жадные*, то есть «захватывают» максимально возможное число символов

Жадные и ленивые квантификаторы

Ленивые квантификаторы «захватывают» минимально возможное число символов:

- `*?`
- `+?`
- `{3,}??`

Жадные и ленивые квантификаторы

Ленивые квантификаторы «захватывают» минимально возможное число символов:

- `*?`
- `+?`
- `{3,}??`
- `Yaa+?..` : `Yaaandex`

Жадные и ленивые квантификаторы

Ленивые квантификаторы «захватывают» минимально возможное число символов:

- `*?`
- `+?`
- `{3,}??`
- `Yaa+?..` : Yaaindex

Классы символов: []

- `[AEIOUaeiou]`: одна любая гласная буква латинского алфавита
- `[0123456789]` или `[0-9]`: любая цифра
- `[^a-zA-Z]`: любой символ, кроме буквы латинского алфавита
- `[-ad]`, `[ad-]` или `[a\ -d]`: один символ `a`, `d` или `-`

Специальные классы символов: `\w`, `\d` и пр.

- `\d`: любая цифра
- `\D`: любой символ, кроме цифры
- `\w`: то же, что `[a-zA-Z0-9_]`
- `\W`: то же, что `[^a-zA-Z0-9_]`
- `\s`: любой пробельный символ
(`[\f\n\r\t\v]`)
- `\S`: любой непробельный символ

Примеры

1. Проверить, что строка выглядит как номер телефона: +7 916 520-76-49

Примеры

1. Проверить, что строка выглядит как номер телефона: +7 916 520-76-49

`^\+\d(\d{3}){2}(-\d\d){2}$`

(здесь видно, что пробел не проглатывается)

Примеры

1. Проверить, что строка выглядит как номер телефона: +7 916 520-76-49

```
^\+\d(\d{3}){2}(-\d\d){2}$
```

(здесь видно, что пробел не проглатывается)

2. Проверить, что строка является корректной записью времени:

```
22:34:04
```

Примеры

1. Проверить, что строка выглядит как номер телефона: +7 916 520-76-49

```
^\+\d(\d{3}){2}(-\d\d){2}$
```

(здесь видно, что пробел не проглатывается)

2. Проверить, что строка является корректной записью времени:

```
22:34:04
```

```
^([01]\d|2[0-3])(:[0-5]\d){2}$
```

Избыточность языка

1. Можно оставить только круглые скобки, якоря

`^` и `$`, `|` и `*`

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$, |$ и $*$
2. Как обойтись без $??$ $ab?c$ \Leftrightarrow

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$
3. Как обойтись без $+?$ $ab+c \Leftrightarrow$

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$
3. Как обойтись без $+?$ $ab+c \Leftrightarrow abb*c$

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$
3. Как обойтись без $+?$ $ab+c \Leftrightarrow abb*c$
4. На что заменить $\{,2\}$? $ab\{,2\}c \Leftrightarrow$

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$
3. Как обойтись без $+?$ $ab+c \Leftrightarrow abb*c$
4. На что заменить $\{,2\}?$ $ab\{,2\}c \Leftrightarrow a(|b|bb)c$

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$
3. Как обойтись без $+?$ $ab+c \Leftrightarrow abb*c$
4. На что заменить $\{,2\}?$ $ab\{,2\}c \Leftrightarrow a(|b|bb)c$
5. На что заменить $[0-9]?$

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$
3. Как обойтись без $+?$ $ab+c \Leftrightarrow abb*c$
4. На что заменить $\{,2\}?$ $ab\{,2\}c \Leftrightarrow a(|b|bb)c$
5. На что заменить $[0-9]?$
На $0|1|2|3|4|5|6|7|8|9$

Избыточность языка

1. Можно оставить только круглые скобки, якоря \wedge и $\$$, $|$ и $*$
2. Как обойтись без $??$ $ab?c \Leftrightarrow a(|b)c$
3. Как обойтись без $+?$ $ab+c \Leftrightarrow abb*c$
4. На что заменить $\{,2\}?$ $ab\{,2\}c \Leftrightarrow a(|b|bb)c$
5. На что заменить $[0-9]?$
На $0|1|2|3|4|5|6|7|8|9$
6. Вместо $.$ можно перечислить все символы, но лень

Сохраняющие круглые скобки

- Сохраняющие круглые скобки запоминают, какая часть строки соответствует части регулярного выражения в круглых скобках
- По умолчанию все круглые скобки сохраняющие
- `(\d{2}) : (\d{2}) : (\d{2})` : 54 : 18 : 33

Сохраняющие круглые скобки

- Сохраняющие круглые скобки запоминают, какая часть строки соответствует части регулярного выражения в круглых скобках
- По умолчанию все круглые скобки сохраняющие
- `(\d{2}):(\d{2}):(\d{2})` : 54:18:33
- `(\d{2}:?){3}` и `54:18:33` :

Сохраняющие круглые скобки

- Сохраняющие круглые скобки запоминают, какая часть строки соответствует части регулярного выражения в круглых скобках
- По умолчанию все круглые скобки сохраняющие
- `(\d{2}):(\d{2}):(\d{2})` : 54:18:33
- `(\d{2}:?){3}` и `54:18:33` : 33 и больше ничего (сохранилось последнее вхождение, но могло сохраниться и первое)

Сохраняющие круглые скобки

- Сохраняющие круглые скобки запоминают, какая часть строки соответствует части регулярного выражения в круглых скобках
- По умолчанию все круглые скобки сохраняющие
- `(\d{2}) : (\d{2}) : (\d{2})` : 54 : 18 : 33
- `(\d{2}:?){3}` и `54:18:33` : 33 и больше ничего (сохранилось последнее вхождение, но могло сохраниться и первое)
- Несохраниющие круглые скобки: `(?:...)`
- `(?: (\d\d) \. (\d\d) \.)? (\d{4})` : 31 . 12 . 2014 ,
2015

Пример: сохраняющие круглые скобки (Python)

```
>>> r = r'(?:(\d\d)\.(\d\d)\.?)?(\d{4})'  
>>> re.search(r, 'Date is 31.12.2014').groups()  
( '31', '12', '2014' )  
>>> re.search(r, 'Date is 31.12.2014').group()  
'31.12.2014'  
>>> re.search(r, 'Year is 2015').groups()  
(None, None, '2015')  
>>> re.search(r, 'Year is 2015').group()  
'2015'
```

Пример: сохраняющие круглые скобки (Javascript)

```
>>> r = /(?:\d\d)\.(\d\d)\.?)?(\d{4})/  
>>> r.exec('31.12.2014')  
["31.12.2014", "31", "12", "2014"]  
>>> r.exec('2015')  
["2015", undefined, undefined, "2015"]
```

Пример: сохраняющие круглые скобки (C++)

```
string s("31.12.2014");
regex r(R"((?:(\\d\\d)\\. (\\d\\d)\\. )?(\\d{4}))");
smatch sm;
regex_match(s, sm, r);

for (unsigned i = 0; i < sm.size(); ++i) {
    cout << sm[i] << endl;
}
// "31.12.2014", "31", "12", "2014"
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\.(?P<month>\d\d)\.)(?P<year>\d4) '  
>>> re.search(r, '31.12.2014').group('month')
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\. (?P<month>\d\d)\. )?(?P<year>\d4) '  
>>> re.search(r, '31.12.2014').group('month')  
'12'
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\. (?P<month>\d\d)\. )?(?P<year>\d4) '  
>>> re.search(r, '31.12.2014').group('month')  
'12'  
>>> re.search(r, '31.12.2014').groupdict()
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\.?(?P<month>\d\d)\.)?(?P<year>\d4) '  
>>> re.search(r, '31.12.2014').group('month')  
'12'  
>>> re.search(r, '31.12.2014').groupdict()  
{'day': '31', 'year': '2014', 'month': '12'}
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\.?(?P<month>\d\d)\.)?(?P<year>\d4) '
>>> re.search(r, '31.12.2014').group('month')
'12'
>>> re.search(r, '31.12.2014').groupdict()
{'day': '31', 'year': '2014', 'month': '12'}
>>> re.search(r, '2015').groupdict()
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\.?(?P<month>\d\d)\.)?(?P<year>\d4) '
>>> re.search(r, '31.12.2014').group('month')
'12'
>>> re.search(r, '31.12.2014').groupdict()
{'day': '31', 'year': '2014', 'month': '12'}
>>> re.search(r, '2015').groupdict()
{'day': None, 'year': '2015', 'month': None}
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\.?(?P<month>\d\d)\.)?(?P<year>\d4) '
>>> re.search(r, '31.12.2014').group('month')
'12'
>>> re.search(r, '31.12.2014').groupdict()
{'day': '31', 'year': '2014', 'month': '12'}
>>> re.search(r, '2015').groupdict()
{'day': None, 'year': '2015', 'month': None}
>>> for match in re.finditer(r, '31.12.2014, 01.01.2015'):
...     print(match.groupdict())
... 
```

Пример: именованные группы (Python)

```
>>> r = r'(?:(?P<day>\d\d)\.?(?P<month>\d\d)\.?)?(?P<year>\d4) '
>>> re.search(r, '31.12.2014').group('month')
'12'
>>> re.search(r, '31.12.2014').groupdict()
{'day': '31', 'year': '2014', 'month': '12'}
>>> re.search(r, '2015').groupdict()
{'day': None, 'year': '2015', 'month': None}
>>> for match in re.finditer(r, '31.12.2014, 01.01.2015'):
...     print(match.groupdict())
...
{'day': '31', 'year': '2014', 'month': '12'}
{'day': '01', 'year': '2015', 'month': '01'}
```

Чем не нравится регулярное выражение?

Повышаем читаемость (Python)

re.X. This flag allows you to write regular expressions that look nicer. Whitespace within the pattern is ignored, except when in a character class or preceded by an unescaped backslash, and, when a line contains a '#' neither in a character class or preceded by an unescaped backslash, all characters from the leftmost such '#' through the end of the line are ignored.

```
>>> r = r"""
...     (?:  
...         (?P<day>  
...             \d\d  
...         )  
...     \.  
...     (?P<month>  
...         \d\d  
...     )  
...     \.  
...     # dot between month and year  
...     )?  
...     # day and month are optional  
...     (?P<year>  
...         \d{4}  
...     )  
... """
>>> re.search(r, '31.12.2014', flags = re.X).groupdict()
{'day': '31', 'year': '2014', 'month': '12'}
```

Где используются регулярные выражения?

- Проверка корректности данных в HTML-формах (e-mail, номер телефона, количество денег...)
- Поиск в тексте слов определённого вида или заданной длины
- «Умная» замена в тексте
- Попытка угадать часть речи для слова
(ed\$ ⇒ глагол в прошедшем времени,
est\$ ⇒ прилагательное в превосходной степени)

Обратная связь

- Можно сослаться на сохранённую группу по номеру с помощью $\backslash 1, \dots, \backslash 9$
- $(\backslash d+) - \backslash 1 = 0$: $54 - 981 = 0$,
 $493 - 493 = 0$, $589 - 89 = 0$

Обратная связь

- Можно сослаться на сохранённую группу по номеру с помощью $\backslash 1, \dots, \backslash 9$
- $(\backslash d+) - \backslash 1 = 0$: $54 - 981 = 0$,
 $493 - 493 = 0$, $589 - 89 = 0$

Обратная связь

- Можно сослаться на сохранённую группу по номеру с помощью

$\backslash 1, \dots, \backslash 9$

- $(\backslash d+) - \backslash 1 = 0$: $54 - 981 = 0$,
 $493 - 493 = 0$, $589 - 89 = 0$

Обратная связь

- Можно сослаться на сохранённую группу по номеру с помощью

$\backslash 1, \dots, \backslash 9$

- $(\backslash d+) - \backslash 1 = 0$: $54 - 981 = 0$,
 $493 - 493 = 0$, $589 - 89 = 0$

Замена

- В строке `12+34, 98-32, 1/98`
заменяем `(\d+) [-+*/] (\d+)`
на `\2*\1`

Замена

- В строке 12+34, 98-32, 1/98
заменяем $(\backslash d+)[-+*/](\backslash d+)$
на $\backslash 2*\backslash 1$

Замена

- В строке 12+34, 98-32, 1/98
заменяем $(\backslash d+)[-+*/](\backslash d+)$
на $\backslash 2*\backslash 1$
- $34*12$, $32*98$, $98*1$

Замена

- В строке 12+34, 98-32, 1/98
заменяем $(\d+)[-+*/](\d+)$
на $\2*\1$
- $34*12$, $32*98$, $98*1$
- А если заменяли $(\d+)[-+*/](\d+?)$?

Замена

- В строке 12+34, 98-32, 1/98
заменяем $(\backslash d+)[-+*/](\backslash d+)$
на $\backslash 2*\backslash 1$
- 34*12, 32*98, 98*1
- А если заменяли $(\backslash d+)[-+*/](\backslash d+)?$?
- 3*124, 3*982, 9*18

Пример: замена (Python)

```
>>> re.sub(
...     r'(\d+)[-+*/](\d+)',
...     r'\2*\1',
...     '12+34, 98-32, 1/98'
... )
'34*12, 32*98, 98*1'
>>> re.sub(
...     r'(\d+)[-+*/](\d+?)',
...     r'\2*\1',
...     '12+34, 98-32, 1/98'
... )
'3*124, 3*982, 9*18'
```

Пример: замена (Javascript)

```
>>> "12+34, 98-32, 1/98".replace(
...   /(\d+)[-+*/](\d+)/g,
...   "$2*$1"
... )
"34*12, 32*98, 98*1"
>>> "12+34, 98-32, 1/98".replace(
...   /(\d+)[-+*/](\d+?)/g,
...   "$2*$1"
... )
"3*124, 3*982, 9*18"
```

Пример: ещё замена (Python)

```
>>> re.sub(  
...     r'(\d)(\d)(?:(-)(\d)(\d))?',  
...     r'\2\1\3\5\4',  
...     '56-54'  
... )
```

Пример: ещё замена (Python)

```
>>> re.sub(  
...     r'(\d)(\d)(?:(-)(\d)(\d))?',  
...     r'\2\1\3\5\4',  
...     '56-54'  
... )  
'65-45'
```

Пример: ещё замена (Python)

```
>>> re.sub(
...     r'(\d)(\d)(?:(-)(\d)(\d))?',
...     r'\2\1\3\5\4',
...     '56-54'
... )
'65-45'
>>> re.sub(
...     r'(\d)(\d)(?:(-)(\d)(\d))?',
...     r'\2\1\3\5\4',
...     '56'
... )
```

Пример: ещё замена (Python)

```
>>> re.sub(
...     r'(\d)(\d)(?:(-)(\d)(\d))?',
...     r'\2\1\3\5\4',
...     '56-54'
... )
'65-45'
>>> re.sub(
...     r'(\d)(\d)(?:(-)(\d)(\d))?',
...     r'\2\1\3\5\4',
...     '56'
... )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Python341\lib\re.py", line 175, in sub
    return _compile(pattern, flags).sub(repl, string, count)
  File "C:\Python341\lib\re.py", line 319, in filter
    return sre_parse.expand_template(template, match)
  File "C:\Python341\lib\sre_parse.py", line 853, in expand_template
    raise error("unmatched group")
sre_constants.error: unmatched group
```

Пример: ещё замена (Javascript)

```
>>> '56-54'.replace(
...   /(\d)(\d)(?:(-)(\d)(\d))?/g,
...   '$2$1$3$5$4'
... )
'65-45'
>>> '56'.replace(
...   /(\d)(\d)(?:(-)(\d)(\d))?/g,
...   '$2$1$3$5$4'
... )
'65'
```

Задачи

1. Строки вида `aabbaa`

Задачи

1. Строки вида `aabbaa`
2. Изменить формат номера телефона:
с `+7 916 520-76-49`
на `8 (916) 520 76 49`

Задачи

1. Строки вида `aabbaa`
2. Изменить формат номера телефона:
с `+7 916 520-76-49`
на `8 (916) 520 76 49`
3. Оставить в строке со странными символами только буквы, цифры, `-`
и `_`