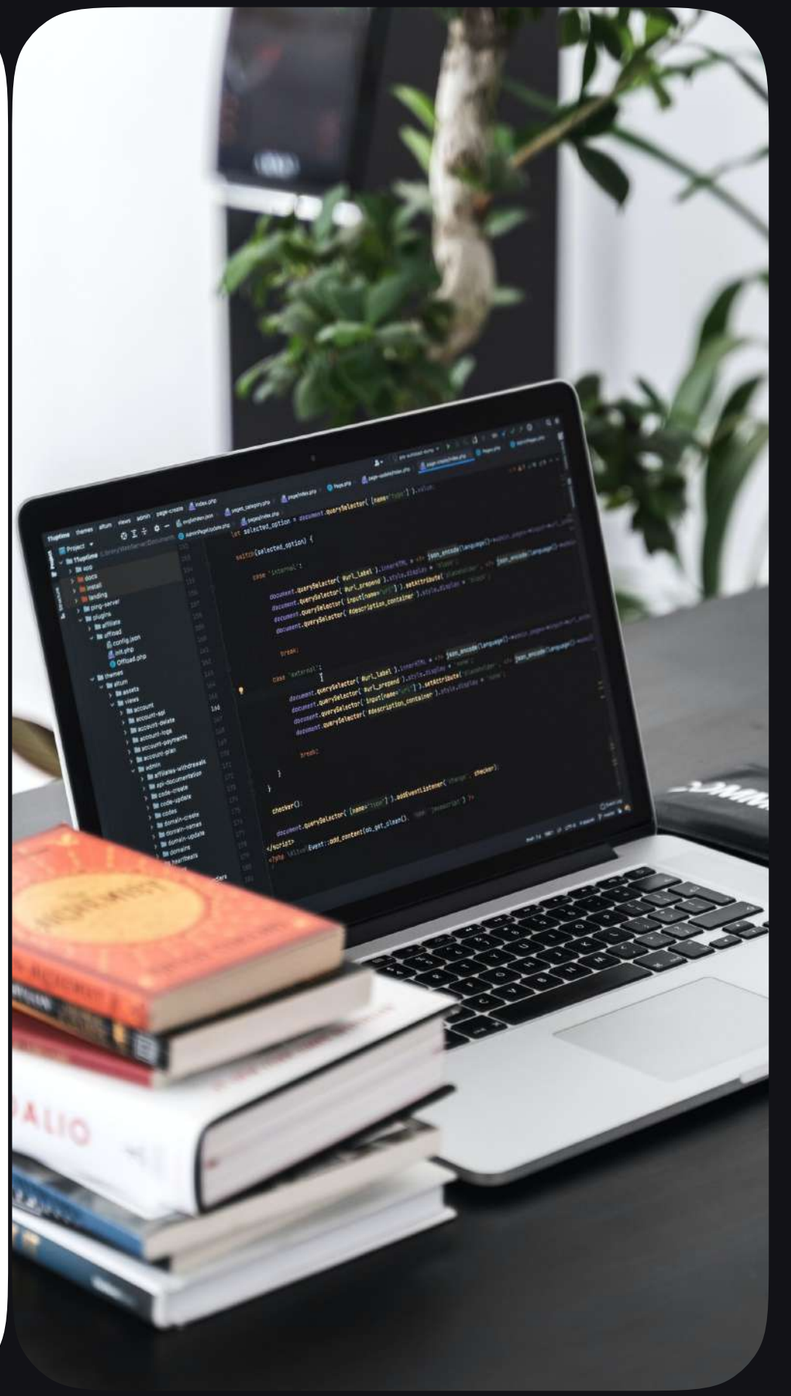
Модуль 3 Занятие 7

Регулярные выражения



Регулярные выражения

Модуль re

Регулярные выражения

Регулярные выражения (Regex, Regular expressions) — это строки, задающие шаблон для поиска подстрок в тексте.

Регулярное выражение или «регулярка» состоит из обычных символов и «метасимволов» – специальных символов позволяющих использовать условия и повторения в шаблоне.

Например:

- **\d{5}** задает последовательность из 5 цифр
- **\w+** задает последовательность из одной или более буквы, цифры или «_»

regex101

Для работы с регулярными выражениями можно воспользоваться удобными online сервисами:

- отладчик регулярных выражений regex101.com
- визуализация регулярных выражений debuggex.com

Поиск текста

Самая простая «регулярка» — это любая строка (в которой нет спецсимволов .^$*+?{}[]\|()). Работает как простой поиск —ищет полное совпадение строки.

Например, найдем все вхождения кот в строку:

regex: кот

текст: Кот кот котик каток слякоть кит пакет

результат: Кот кот котик каток слякоть кит пакет

Поиск любого символа

Метасимвол . в шаблоне — найдет один любой символ (кроме новой строки \n).

Например, найдем все слова, которые начинаются на к и заканчиваются на т из трех букв:

Например, найдем все вхождения кот в строку:

regex: к.т

текст: Кот кот котик каток слякоть кит пакет к тополю

результат: Кот кот котик каток слякоть кит пакет к тополю

Поиск любого символа

Для поиска спецсимвола, например, точки, его нужно экранировать — добавить перед спецсимволом обратный слэш (\).

Например, найдем все расширения «.txt» в строке:

regex: \.txt

текст: str.txt 5txt 5.txt 4rtxt .txt

результат: str.txt 5txt 5.txt 4rtxt .txt

Метасимволы

В регулярных выражениях можно использовать специальные метасимволы, которые заменяют собой конкретные наборы значений:

Символ	Значение	Пример regex	Результат
.	Один любой символ (кроме новой строки \n)	к.т.к	каток котик к1т1к
\d	Любая цифра	x\d	x5 x3 xy x23 xx
\D	Любой символ кроме цифры	x\D	x5 x3 xx x23 xx x_
\s	Любой пробельный символ (пробел, табуляция, конец строки и т.д.)	кот\s	кот котик кот кот.
\S	Любой непробельный символ	кот\S	кот котик кот кот.
\w	Любая буква, цифра и _	\w\w21	AA21 _521 A+21 __21
\W	Любой символ не буква, не цифра и не _	\w\W21	AA21 _521 A+21 __21

Пример

Для поиска имени файла перед расширением воспользуемся метасимволом \w.

Например, найдем все файлы имеющие расширение «.txt»:

regex: \w\.txt

текст: str.txt 5txt 5.txt 4rtxt .txt

результат: str.txt 5txt 5.txt 4rtxt .txt

Квантификаторы

Квантификаторы — это специальные символы в регулярных выражениях, которые указывают количество повторений текста.

Символ	Значение	Пример regex	Результат
?	Ноль или одно вхождение	кот?	кот коты ко ко котик
*****	Ноль или более повторений	x\d*	x x7 x777 yx7
+	Одно или более повторений	x\d+	x x7 x777 yx7
{n}	Ровно n повторений	\d{4}	1 12 123 1234 12345
{m,n}	От m до n повторений включительно	\d{2,4}	1 12 123 1234 12345
{m,}	Не менее m повторений	\d{3,}	1 12 123 1234 12345
{,n}	Не более n повторений	\d{,2}	1 12 123

Квантификаторы

По умолчанию, квантификаторы жадные, т.е. они захватывают максимально возможное число символов. Добавление ? делает их ленивыми и они захватывают минимально возможное число символов.

regex: \(.*\)

текст: (a+b)+(c+d)+(d+e)

результат: (a+b)+(c+d)+(d+e)

regex: \(.*?\)

текст: (a+b)+(c+d)+(d+e)

результат: (a+b)+(c+d)+(d+e)

Диапазон допустимых значений

Для поиска конкретных допустимых значений в строке можно воспользоваться квадратными скобками. Внутри [] можно перечислить конкретные символы или указать диапазон:

Пример regex	Значение
[аои]	Только один из трех символов: «а» «о» или «и»
[а-я]	Все русские буквы в нижнем регистре от «а» до «я» (кроме «ё»)
[А-Я]	Все заглавные русские буквы (кроме «Ё»)
[А-Яа-яЁё]	Любые русские буквы
[0-9]	Любая цифра
^ внутри [] означает исключение:	
[^467]	Любой символ кроме цифр 4, 6 и 7

Перечисления

Чтобы найти строку, удовлетворяющую одному из шаблонов, можно воспользоваться аналогом логического оператора ИЛИ, который записывается с помощью символа |.

Например, найдем все даты в формате ДД.ММ.ГГГГ или ДД.ММ.ГГ:

regex: \d{1,2}\.\d{1,2}\.(?:\d{4}|\d{2})

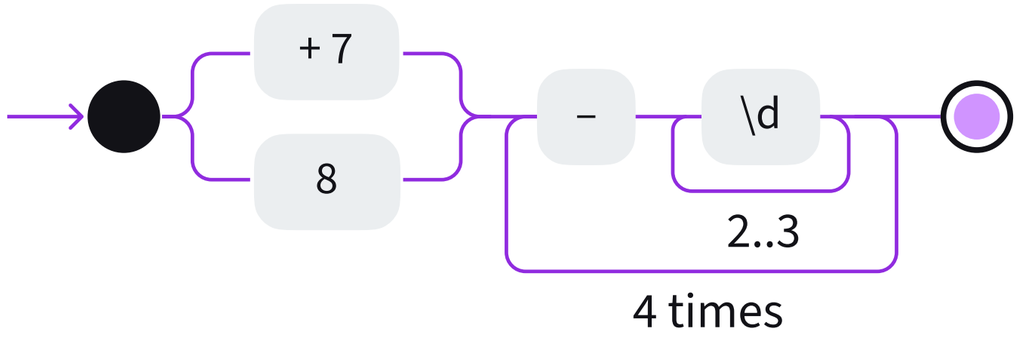
текст: 02.12.2023 9.12.23 9.12.3 9.12.3333

результат: 02.12.2023 9.12.23 9.12.3 9.12.3333

Круглые скобки (?:REGEXP) в данном примере позволяют локализовать часть шаблона, внутри которого происходит перечисление.

Группы

Если строка состоит из повторяющихся строк, то для поиска таких групп используют круглые скобки (?:REGEXP). К (?:REGEXP) можно применять квантификаторы, указывая, сколько именно раз должна повториться группа.



regex: (?:\+7|8)(?:-\d{2,3}){4}

текст: 8-924-77-77-77 +7-123-456-78-90 8-317-444-47-45

результат: 8-924-77-77-77 +7-123-456-78-90 8-317-444-47-45

Ссылки на группы

Если в регулярном выражении используются круглые скобки () без ?:, то они становятся группирующими. Использование групп добавляет возможность использовать в шаблоне ссылки на найденные группы, при помощи \1, \2, \3 и так далее.

regex: (\w+)\-\1

текст: том-ям том-том ам-ам мама дом-мод ту-ту ку-ку

результат: том-ям том-том ам-ам мама дом-мод ту-ту ку-ку

Regex в Python

Для работы с регулярными выражениями в Python существует встроенный модуль re, который содержит следующие функции:

- **re.search**(pattern, string) — поиск в строке string подстроки pattern. Возвращает первое совпадение или None
- **re.match**(pattern, string) — поиск в начале строки string подстроки pattern. Возвращает первое совпадение или None
- **re.findall**(pattern, string) — поиск подстроки pattern по всей строке string. Возвращает список со всеми найденными совпадениями
- **re.split**(pattern, string) — разделяет строку string по подстрокам, соответствующим pattern. Возвращает список строк
- **re.sub**(pattern, repl, string) — заменяет в строке string все подстроки pattern на repl. Возвращает измененную строку

r-строки

Так как символ \ в Python строках необходимо экранировать, то для написания регулярных выражений в Python используют r-строки (сырые строки).

Синтаксически r-строки записываются так:

```
r'...'
```

Пример

Чтобы в результат вошли строки в другом регистре, применяют флаг re.IGNORECASE, например:

re.search(pattern, string, re.IGNORECASE)

```
import re

pattern1 = r'\d\d\.\d\d'
pattern2 = r'\d\d\.\d\d\.\d{4}'
string = '01.12 должно было произойти что-то, но произойдет 02.12.2023'

match1 = re.search(pattern1, string)
match2 = re.search(pattern2, string)
print(match1)  # <re.Match object; span=(0, 5), match='01.12'>
print(match2)  # <re.Match object; span=(50, 60), match='02.12.2023'>

match3 = re.match(pattern1, string)
match4 = re.match(pattern2, string)
print(match3)  # <re.Match object; span=(0, 5), match='01.12'>
print(match4)  # None
```

Пример

Чтобы в результат вошли строки в другом регистре, применяют флаг re.IGNORECASE, например:

re.search(pattern, string, re.IGNORECASE)

```
import re

pattern = r'\d\d\.\d\d'
string = '01.12 должно было произойти что-то, но произойдет 02.12.2023'

match = re.findall(pattern, string)
print(match)  # ['01.12', '02.12']

result = re.split(pattern, string)
print(result)  # ['', ' должно было произойти что-то, но произойдет ', '.2023']

new_string = re.sub(pattern, '-удалено-', string)
print(new_string)  # -удалено- должно было произойти что-то, но произойдет -удалено-.2023
```

Match объекты

Функция re.match и re.search возвращают Match объекты:

<re.Match object; span=(0, 5), match='01.12'>

- **span** — это индекс начала и конца найденной подстроки в тексте, по которому мы искали совпадение (второй индекс не включается в подстроку)
- **match** — найденная подстрока

Вывести найденную подстроку можно с помощью метода group() или обратиться к объекту по индексу 0:

```
print(match.group(0))
print(match[0])
```

Пример

```
import re

pattern = r'[\w\.-]+@[\w\.-]+\w+'
string = """Обратная связь: onlineshop@gmail.com,
vasya-shop@yandex.ru. Для сотрудничества:
ad.online_shop@gmail.com!"""

emails = re.findall(pattern, string)
for email in emails:
    print(email)
```

Найдем все email строке.

Будем считать, что email состоит из одного символа @, текста до собачки и текста после собачки.
Текст — это любая буква, цифра, знак подчеркивания или точка.

Вывод:

```
onlineshop@gmail.com
vasya-shop@yandex.ru
ad.online_shop@gmail.com
```

Пример

```
import re

pattern = r'(\d{4})/(\d{1,2})/(\d{1,2})'
string = "1999/12/31 2022/1/1 1987/12/7"
new_string = re.sub(pattern, r'\3.\2.\1', string)
print(new_string)
```

Вывод:

```
31.12.1999 1.1.2022 7.12.1987
```

Переведем все даты в тексте из формата ГГГГ/ММ/ДД в формат ДД.ММ.ГГГГ.

Итоги

- Регулярные выражения — это очень мощный инструмент для поиска в тексте
- В регулярных выражения можно использовать метасимволы и квантификаторы
- Для работы с регулярными выражениями в Python существует встроенный модуль re
- Для работы с регулярными выражениями можно воспользоваться удобным online-отладчиком регулярных выражений regex101.com