IT CAREER HUB

Урок 31.
Регулярные выражения

IT CAREER HUB

Регулярные выражения

Регулярные выражения (RegEx) – это инструмент для поиска, сравнения, замены и проверки строк по заданному шаблону.

Регулярные выражения используются для:

- **Поиска текста по шаблону** (например, найти все email-адреса в документе).
- **Проверки формата строки** (например, соответствует ли телефонный номер формату `+7 (999) 123-45-67`).
- **Замены текста** (например, изменить все даты из формата `01-02-2025` в `01.02.2025`).
- **Разделения строк** (например, разбить строку по нескольким пробелам или запятым).

Где применяются регулярные выражения?

- **Обработка данных** – фильтрация, поиск и замена информации в текстах и файлах.
- **Формы ввода** – валидация email, номеров телефонов, паролей.
- **Парсинг текста** – извлечение нужных данных из веб-страниц, логов, документов.
- **Поиск в коде** – нахождение и рефакторинг кода в редакторах и IDE.

IT CAREER HUB

Модуль re

Модуль `re` предоставляет инструменты для **работы с регулярными выражениями**.

Основные функции модуля re:

- `re.search(pattern, string)` – ищет первое совпадение шаблона в строке.
- `re.match(pattern, string)` – проверяет, начинается ли строка с шаблона.
- `re.findall(pattern, string)` – возвращает список всех совпадений.
- `re.finditer(pattern, string)` – возвращает итератор с совпадениями.
- `re.sub(pattern, repl, string)` – заменяет найденные совпадения.
- `re.split(pattern, string)` – разделяет строку по шаблону.

IT CAREER HUB

Функция findall

Функция re.findall() ищет все совпадения шаблона в строке и возвращает их в виде списка.

Синтаксис

```
Python
re.findall(pattern, string)
```

- `pattern` – регулярное выражение (шаблон поиска).
- `string` – строка, в которой выполняется поиск.

Пример: поиск всех чисел в строке

```
Python
import re

text = "Python 3.9, Java 17, C++ 14"

numbers = re.findall(r"\d+", text)  # Поиск всех чисел
print(numbers)
```

IT CAREER HUB

Основные символы в регулярных выражениях

Регулярные выражения используют специальные символы для **поиска букв, цифр, пробелов и других элементов текста**.

Обозначения основных символов

Символ	Описание
`\d`	Любая цифра (`0-9`)
`\D`	Любой символ, кроме `\d`
`\w`	Буквы, цифры и `_`
`\W`	Любой символ, кроме `\w`
`\s`	Пробел, `\t`, `\n`
`\S`	Любой символ, кроме `\s`
`.`	Любой символ, кроме `\n`

Пример

Python

```python
import re

text = "\tPython 3.12, Java 17, C++ 14!\n"

print("Цифры:", re.findall(r"\d", text))
print("Двузначные цифры:", re.findall(r"\d\d", text))

print("НЕ цифры:", re.findall(r"\D", text))

print("Буквы, цифры, _:", re.findall(r"\w", text))

print("НЕ буквы, цифры, _:", re.findall(r"\W", text))
```

```
print("Пробелы:", re.findall(r"\s", text))

print("НЕ пробелы:", re.findall(r"\S", text))

print("Все символы (кроме \\n):", re.findall(r".", text))
```

Символ 'r' перед строкой шаблона

В регулярных выражениях **используется много специальных символов**, таких как `\d`, `\w`, `\s` и другие.

Python обрабатывает `\` **(обратный слэш) как управляющий символ**, из-за чего могут возникнуть ошибки.
Чтобы избежать этого, перед строкой **рекомендуется ставить** `r` **("сырая" строка, raw string)**.

Пример: если не использовать r

```python
import re

pattern = "\d"  # ОШИБКА: \d будет воспринято как управляющий символ
print(re.findall(pattern, "Price: 123"))
```

Вывод

```
/home/tanya/PycharmProjects/pythonProgramItch/.venv/bin/python /home/tanya/PycharmProjects/pythonProgramItch
['1', '2', '3']
/home/tanya/PycharmProjects/pythonProgramItch/_notes/test.py:3: SyntaxWarning: invalid escape sequence '\d'
  pattern = "\d"  # ОШИБКА: \d будет воспринято как управляющий символ

Process finished with exit code 0
```

IT CAREER HUB

Классы символов

Классы символов ([ ]) используются для поиска любого из указанных символов. Они позволяют задать набор символов, который должен встречаться в искомом фрагменте.

Внутри [ ] можно использовать:

- `-` **(дефис)** для указания **диапазона символов** (например, `[a-z]` — все буквы от `a` до `z`).
- `^` **(каретку) в начале** для **исключения символов** (например, `[^0-9]` — всё, кроме цифр).

Обозначения классов символов

Запись	Описание
`[afc]`	Один из символов `a`, `f` или `c`
`[0-9]`	Любая цифра от `0` до `9` (аналог `\d`)
`[a-z]`	Любая строчная буква от `a` до `z`
`[A-Z]`	Любая заглавная буква от `A` до `Z`
`[a-zA-Z]`	Любая буква (заглавная или строчная)
`[^abc]`	Любой символ, **кроме** a, b или c
`[^0-9]`	Любой символ, **кроме** цифры

Пример

```python
import re

text = "Report, report, report2, report10"
```

```
print("Буквы r или R в слове:", re.findall(r"[rR]eport", text))

print("Все цифры:", re.findall(r"[0-9]", text))

print("Заглавные буквы:", re.findall(r"[A-Z]", text))

print("Строчные буквы:", re.findall(r"[a-z]", text))

print("Все буквы:", re.findall(r"[a-zA-Z]", text))

print("Все, кроме цифр:", re.findall(r"[^0-9]", text))
```

Квантификаторы

Квантификаторы в регулярных выражениях определяют количество повторений символов или групп. Они позволяют указывать, сколько раз подряд должен встречаться символ или шаблон.

Обозначения квантификаторов

Квантификатор	Описание
+	Один или более раз (1, 2, 3...)
*	Ноль или более раз (0, 1, 2...)
?	Ноль или один раз (0, 1)
{n}	Ровно n раз
{n,}	n и более раз
{n,m}	От n до m раз

Примеры

```python
import re

text = """
Orders: ID123, ID4567, ID89
Numbers: 123-45-67, 321-45-67
Prices: 100$, 199.50$, 99.99€, 0.49€, .99€
File names: report.txt, report2.txt, report10.txt
"""

print("Одна или более цифр:", re.findall(r"\d+", text))
```

```
print("Телефонные номера (формата xxx-xx-xx):", re.findall(r"\d{3}-\d{2}-\d{2}", text))

print("Цены (числа с десятичной точкой):", re.findall(r"\d+\.\d+", text))

print("ID-коды:", re.findall(r"ID\d{2,}", text))

print("Имена файлов 0+ цифр:", re.findall(r"report\d*.txt", text))

print("Имена файлов 0/1 цифр:", re.findall(r"report\d?.txt", text))

print("Имена файлов 1/2 цифр:", re.findall(r"report\d{1,2}.txt", text))
```

IT CAREER HUB

Жадные и ленивые квантификаторы

Квантификаторы (`*`, `+`, `{n,m}`) по умолчанию работают **жадно** – они стараются захватить как можно **больше символов**.
Но иногда нужно, чтобы они захватывали **минимально возможное** количество символов – в этом случае используются **ленивые квантификаторы**.

Чтобы сделать квантификатор **ленивым**, нужно добавить ? после него.

Обозначения жадных и ленивых квантификаторов

Квантификатор	Тип	Описание
`*`	Жадный	Захватывает **максимально возможное** количество символов
`*?`	Ленивый	Захватывает **минимально возможное** количество символов
`+`	Жадный	Захватывает **минимум 1 символ**, но **как можно больше**
`+?`	Ленивый	Захватывает **минимум 1 символ**, но **как можно меньше**
`{n,m}`	Жадный	Захватывает **от** `n` **до** `m`, но **как можно больше**
`{n,m}?`	Ленивый	Захватывает **от** `n` **до** `m`, но **как можно меньше**

Примеры

Python

```python
import re
text = "<div>Hello</div><div>World</div>"

greedy = re.findall(r"<.*>", text)  # Жадный
lazy = re.findall(r"<.*?>", text)   # Ленивый
```

```
print(greedy)
print(lazy)
```

IT CAREER HUB

Задания для закрепления 1

1. Сопоставьте шаблон с тем, что он найдёт:

1. `\d+`
2. `[a-zA-Z0-9]+`
3. `\s+`
4. `[^a-zA-Z0-9]+`

a. Несколько подряд идущих пробелов
b. Последовательность букв и цифр
c. Последовательность цифр
d. Последовательность НЕ букв и НЕ цифр

Посмотреть ответ

2. Найдите ошибку в шаблоне:

```python
import re
re.findall("\d+", "Value: 123")
```

Посмотреть ответ

3. В каком шаблоне используется ленивый квантификатор?

a. `<.+>`
b. `<.*>`
c. `<.*?>`
d. `*[a-z]`

Посмотреть ответ

IT CAREER HUB

Экранирование специальных символов

В регулярных выражениях есть **символы, которые имеют особое значение** (`.` `+` `*` `{}` `[]` `()` `|` `^` `$`).
Если нужно **найти их как обычные символы**, их **нужно экранировать**, добавляя `\` перед символом.

Пример

```python
import re

text = "report.txt, report2.txt, report10.txt, some_txt_report, some_report_txt"

print("Имена файлов с txt:", re.findall(r"\w+.txt", text))

# Имена файлов с расширением .txt
print("Имена файлов с расширением .txt:", re.findall(r"\w+\.txt", text))

# Имена файлов в папке
print("Имена файлов в папке:", re.findall(r"\w+\\\w+\.\w+", r"reports\report.txt, report2.txt"))
```

Якоря

Якоря используются в регулярных выражениях для указания позиции совпадения в строке.
Они не ищут символы, а определяют, где именно должен находиться искомый фрагмент.

Обозначения якорей

Якорь	Описание
^	Начало строки
$	Конец строки
\b	Граница слова
\B	Не граница слова

Пример

Python

```python
import re

text = "Hello world! Welcome to world"

print("Слово в начале строки:", re.findall(r"^\w+", text))
print("Слово в конце строки:", re.findall(r"\w+$", text))

text2 = "category wildcat education _cat_ catalog"

print("Слова с 'cat' внутри:", re.findall(r"\w+cat\w+", text2))
print("Слова с 'cat' в начале слов:", re.findall(r"\bcat\w*", text2))
print("Слова с 'cat' в конце слов:", re.findall(r"\w*cat\b", text2))

text3 = "X123X 234 4567X X999"
print("Числа внутри строк:", re.findall(r"\B\d+\B", text3))
```

IT CAREER HUB

Альтернативы

Оператор | **(ИЛИ)** позволяет **искать один из нескольких вариантов**.

Пример

```python
import re

text = "Meeting on 2024-05-10 or 10/05/2024 at 14:30"

# Найдём даты в формате YYYY-MM-DD или DD/MM/YYYY
print("Даты:", re.findall(r"(\d{4}-\d{2}-\d{2}|\d{2}/\d{2}/\d{4})", text))
```

Группы

Группы (()) используются для объединения нескольких символов в одну логическую часть.
Они позволяют извлекать части совпадений, применять квантификаторы к целым выражениям и работать с подстроками отдельно.

Пример: извлечение данных с помощью групп

Функции `re.search()` и `re.match()` позволяют **доступ к частям совпадений** через `group()`.

```python
import re

text = "Order ID: 12345, Invoice No: 67890"

# Найдём ID заказа и счёта
match = re.search(r"Order ID: (\d+), Invoice No: (\d+)", text)

if match:
    print("ID заказа:", match.group(1))
    print("Номер счёта:", match.group(2))
```

Пример: использование негруппирующих скобок

Если группа нужна для логики, но не для извлечения данных, можно использовать `(?:...)`.

```python
import re
```

```
text = "USD 100, EUR 200, GBP 300"

# Найдём суммы, не выделяя валюту
matches = re.findall(r"(?:USD|EUR|GBP) (\d+)", text)
print("Суммы:", matches)

# Найдём суммы и валюту
matches = re.findall(r"(USD|EUR|GBP) (\d+)", text)
print("Суммы:", matches)
```

IT CAREER HUB

Функции модуля re

Функция re.match

Функция re.match() проверяет, начинается ли строка с заданного шаблона. Если совпадение найдено, функция возвращает объект Match, который содержит информацию о совпадении, иначе None.

Объект Match

Объект `Match` содержит **информацию о найденном фрагменте**, включая:

- `.group()` – само совпадение.
- `.start()` – индекс начала совпадения.
- `.end()` – индекс конца совпадения.
- `.span()` – кортеж `(start, end)`, показывающий границы совпадения.

Пример

```python
import re

text = "ID12345 is confirmed. ID23456 is confirmed"

# Проверяем, начинается ли строка с "ID" + цифры
match = re.match(r"ID\d+", text)

if match:
    print("Объект Match:", match)
    print("Само совпадение:", match.group())
    print("Диапазон совпадения:", match.span())
else:
    print("Нет совпадения.")
```

IT CAREER HUB

Функция re.search

Функция re.search() ищет первое совпадение регулярного выражения в любой части строки и возвращает объект Match.

Пример

```python
import re

text = "Order ID: 12345, Invoice No: 67890, Ref: ABC9876"

# Найдём первое число в тексте
match = re.search(r"\d+", text)

if match:
    print("Объект Match:", match)
    print("Само совпадение:", match.group())
    print("Индекс начала:", match.start())
    print("Индекс конца:", match.end())
    print("Диапазон совпадения:", match.span())
else:
    print("Нет совпадения.")
```

Флаг re.IGNORECASE

Флаг re.IGNORECASE (или сокращённо re.I) делает поиск регистронезависимым — шаблон будет находить совпадения в любом регистре, даже если написан в нижнем или верхнем.

IT CAREER HUB

Пример

```python
import re

text = "Python is popular."

# Найдём слово "python" без учёта регистра
match = re.search(r"python", text, re.IGNORECASE)

if match:
    print("Найдено:", match.group())
```

Функция re.finditer

Функция re.finditer() ищет все совпадения регулярного выражения в строке и возвращает итератор объектов Match.

Пример

```python
import re

text = "Order ID: 12345, Invoice No: 67890, Ref: ABC9876"

# Найдём все числа в тексте
matches = re.finditer(r"\d+", text)

for match in matches:
    print("Объект Match:", match)
    print("Само совпадение:", match.group())
    print("Диапазон совпадения:", match.span())
    print()
```

Функция re.split

Функция re.split() разделяет строку на части, используя регулярное выражение как разделитель.
Она работает аналогично `str.split()`, но позволяет **разбивать текст по сложным шаблонам**, а не только по одному символу.

Пример

```python
import re

text = """Python is popular. It is used in web development, data science,
and automation. Many developers choose Python for its simplicity."""

# Разделяем строку по запятым, пробелам и точкам
words = re.split(r"[,\s.]+", text)

print("Список слов:", words)
```

Функция re.sub

Функция re.sub() заменяет все найденные совпадения на указанную строку или результат функции.
Это полезно для **форматирования текста, удаления лишних символов и исправления данных**.

Синтаксис

```python
re.sub(pattern, repl, string)
```

- `pattern` – шаблон для поиска.
- `repl` – строка, на которую будет заменено совпадение.

- `string` – исходный текст.

Пример

```python
import re

text = "apple,   banana ,  orange ,grape"

# Удаляем лишние пробелы перед и после запятых
clean_text = re.sub(r"\s*,\s*", ", ", text)

print("Отформатированный текст:", clean_text)
```

Функция re.compile

Функция `re.compile()` позволяет предварительно скомпилировать регулярное выражение, чтобы затем использовать его многократно без повторного пересоздания.

Это полезно, если **одно и то же регулярное выражение используется несколько раз** в коде.

Синтаксис

```python
pattern = re.compile(regular_expression)
```

Теперь можно использовать **переменную** `pattern` с методами:

- `pattern.match(string)` – аналог `re.match()`
- `pattern.search(string)` – аналог `re.search()`
- `pattern.findall(string)` – аналог `re.findall()`
- `pattern.finditer(string)` – аналог `re.finditer()`
- `pattern.split(string)` – аналог `re.split()`

- `pattern.sub(repl, string)` – аналог `re.sub()`

Пример

Python

```python
import re

texts = [
    "Order ID: 12345, Invoice No: 67890, Ref: ABC9876",
    "Shipment ID: 54321, Tracking No: 98765, Customer Ref: XYZ123",
    "Invoice 22222 processed successfully."
]

# Компилируем регулярное выражение для поиска числовых идентификаторов
number_pattern = re.compile(r"\d+")

# Применяем шаблон к разным текстам
for text in texts:
    match = number_pattern.findall(text)
    if match:
        print(f"Совпадение в тексте: {match}")
```

IT CAREER HUB

Задания для закрепления 2

1. **Сопоставь якорь с его назначением:**

1. ^
2. $
3. \b
4. \B

a. Указывает на конец строки
b. Указывает на начало строки
c. Обозначает границу слова
d. Указывает, что позиция не является границей слова

Посмотреть ответ

2. В чём отличие `re.match()` **от** `re.search()`**?**

a. `match()` ищет по всей строке, `search()` — только в начале
b. `match()` работает быстрее
c. `match()` проверяет только начало, `search()` — в любой части строки

Посмотреть ответ

IT CAREER HUB

Полезные материалы

Чтобы быстрее разбираться с шаблонами и экспериментировать с ними, можно использовать специальные онлайн-инструменты. Они позволяют:

- Проверять регулярные выражения в реальном времени
- Подсвечивать совпадения в строках
- Пошагово объяснять работу каждого элемента шаблона
- Использовать синтаксис разных языков (включая Python)
- Сохранять и делиться шаблонами
- Обращаться к встроенной справке и примерам

Наиболее удобные из них:

- regex101.com
- regexr.com

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Сопоставление	Ответ: 1-c, 2-b, 3-a, 4-d
2. Ошибка в шаблоне	Ответ: строка шаблона должна быть `c r`
3. Ленивый квантификатор	Ответ: c
Задания на закрепление 2	Вернуться к заданиям
1. Сопоставление	Ответ: 1-b, 2-a, 3-c, 4-d
2. Отличия `re.match()` и `re.search()`	Ответ: c

IT CAREER HUB

Практическая работа

1. Проверка пароля

Реализуйте программу, которая должна проверить, соответствует ли введённый пароль следующим требованиям:

- Минимум 8 символов
- Есть хотя бы одна заглавная буква
- Есть хотя бы одна строчная буква
- Есть хотя бы одна цифра

Пример вывода:

```
Python
Введите пароль: Pass1234
Пароль надёжен.

---

Введите пароль: k2n6bd7
Пароль не соответствует требованиям.
```

Решение:

```python
password = input("Введите пароль: ")

import re

if (re.search(r"[A-Z]", password) and
    re.search(r"[a-z]", password) and
    re.search(r"\d", password) and
    len(password) >= 8):
    print("Пароль надёжен.")
else:
    print("Пароль не соответствует требованиям.")
```

IT CAREER HUB

2. Извлечение IP-адресов

Программа должна найти все IPv4-адреса в строке.
IPv4-адрес состоит из четырёх чисел от 0 до 255, разделённых точками.

Данные:

```python
text = "Server1: 192.168.1.1, Server2: 10.0.0.254, Invalid: 999.123.456.78"
```

Пример вывода:

```python
192.168.1.1
10.0.0.254
```

Решение:

```python
import re

text = "Server1: 192.168.1.1, Server2: 10.0.0.254, Invalid: 999.123.456.78"

pattern = r"\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b"
candidates = re.findall(pattern, text)

# Фильтруем корректные IP (0–255)
valid_ips = [ip for ip in candidates if all(0 <= int(part) <= 255 for part in
ip.split("."))]

for ip in valid_ips:
    print(ip)
```