

Python

# Регулярные выражения



# Преподаватель

Портрет

**Имя Фамилия**

Текущая должность

Количество лет опыта

Какой у Вас опыт - ключевые кейсы


Самые яркие проекты


Дополнительная информация по вашему усмотрению


Корпоративный e-mail


Социальные сети (по желанию)


# Важно

- 

Камера должна быть включена на протяжении всего занятия
- 







В течение занятия вопросы задавать в чате или когда преподаватель спрашивает, есть ли у Вас вопросы
- 

Вести себя уважительно и этично по отношению к остальным участникам занятия
- 

Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях
- 

Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя

# Повторение

-  JSON
-  Модуль json
-  Сравнение типов Python и JSON
-  Форматирование JSON
-  JSONDecodeError
-  Модуль datetime

# План занятия

- Регулярные выражения
- Модуль re
- Функция findall
- Основные символы в регулярных выражениях
- Символ 'r' перед строкой шаблона
- Классы символов
- Квантификаторы
- Жадные и ленивые квантификаторы
- Экранирование специальных символов
- Якоря
- Альтернативы
- Группы
- Функции модуля



# ОСНОВНОЙ БЛОК





# Регулярные выражения



## Регулярные выражения (RegEx)

Это инструмент для поиска, сравнения, замены и проверки строк по заданному шаблону

# Назначение регулярных выражений



Поиск текста по шаблону



Проверка формата строки



Замена текста



Разделение строк

# Области применения регулярных выражений



Обработка данных



Формы ввода



Парсинг текста



Поиск в коде



# ВОПРОСЫ





Модуль re



## Модуль re

Этот модуль 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



# ВОПРОСЫ





# Функция findall



## Функция findall

Эта функция ищет все совпадения шаблона в строке и возвращает их в виде списка

# Функция findall



## Синтаксис

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

## Пояснения

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

# Пример: поиск всех чисел в строке

```
import re

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

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



# ВОПРОСЫ





# Основные символы в регулярных выражениях

# Важно



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 использования специальных СИМВОЛОВ

```
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))
```



# ВОПРОСЫ





Символ 'r' перед  
строкой шаблона

# Важно



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

```
import re

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

```
/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
```



# ВОПРОСЫ





# Классы символов



## Классы символов

Они обозначаются квадратными скобками `[]` и используются для поиска любого из указанных внутри символов. Они позволяют задать набор символов, который должен встречаться в искомом фрагменте

# Внутри [] можно использовать

Символ	Пояснение	Назначение	Пример
-	дефис	для указания диапазона символов	[a-z] — все буквы от a до z
^	каретка	для исключения символов	[^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, <b>кроме</b> a, b или c
[^0-9]	Любой символ, <b>кроме</b> цифры

# Пример использования класса СИМВОЛОВ

```
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))
```



# ВОПРОСЫ





# Квантификаторы



## 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

# Пример использования квантификаторов

```
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))
```



# ВОПРОСЫ





# Жадные и ленивые квантификаторы

# Важно



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.

# Важно



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

# Обозначения жадных и ленивых квантификаторов

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

# Пример

```
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)
```



# ВОПРОСЫ





# ЗАДАНИЯ





## Сопоставьте шаблон с тем, что он найдёт

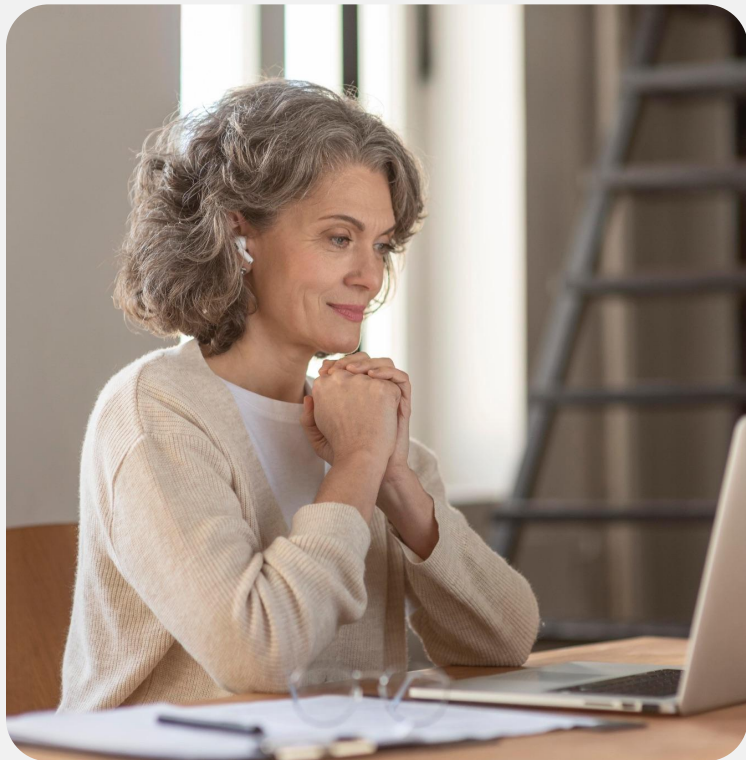
1. `\d+`
  2. `[a-zA-Z0-9]+`
  3. `\s+`
  4. `[^a-zA-Z0-9]+`
- 
- a. Несколько подряд идущих пробелов
  - b. Последовательность букв и цифр
  - c. Последовательность цифр
  - d. Последовательность НЕ букв и НЕ цифр



## 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

**Ответ:** 1-с, 2-b, 3-a, 4-d



## Найдите ошибку в шаблоне

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



## Найдите ошибку в шаблоне

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

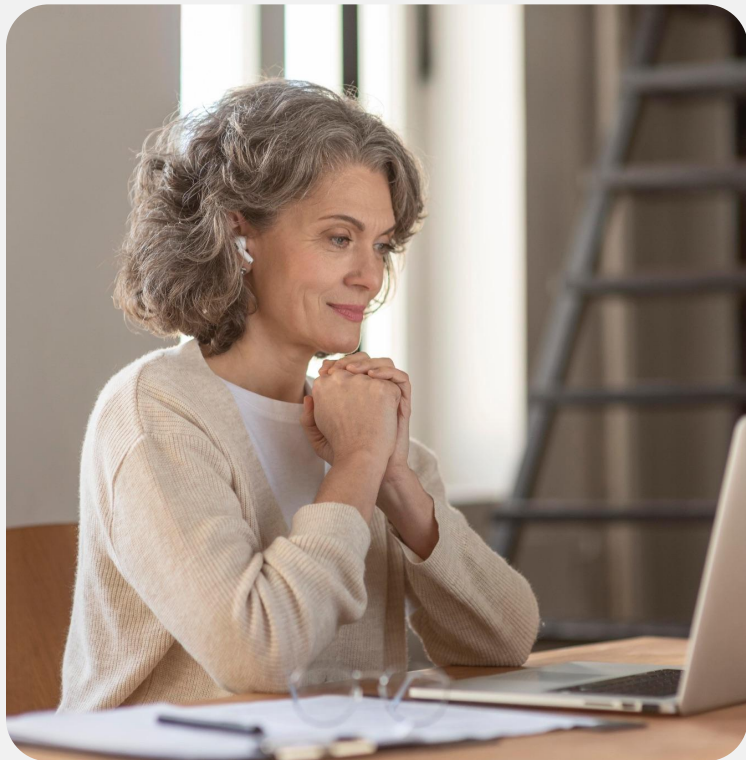
**Ответ:** строка шаблона должна быть с `r`



## Выберите верный вариант ответа

В каком шаблоне используется ленивый  
квантификатор?

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



## Выберите верный вариант ответа

В каком шаблоне используется ленивый  
квантификатор?

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



# ВОПРОСЫ





# Экранирование специальных символов

# Важно



В регулярных выражениях есть символы, которые имеют особое значение (. + \* {} [] () | ^ \$).

Если нужно найти их как обычные символы, их нужно экранировать, добавляя \ перед символом.

# Пример экранирования

```
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"))
```



# ВОПРОСЫ





Якоря



## 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 использования якорей

```
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))
```



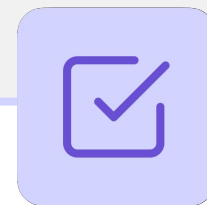
# ВОПРОСЫ





# Альтернативы

# Важно



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

# Пример

```
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))
```



# ВОПРОСЫ





Группы



## Группы

Группы обозначаются круглыми скобками () и используются для объединения нескольких символов в одну логическую часть.

Они позволяют извлекать части совпадений, применять квантификаторы к целым выражениям и работать с подстроками отдельно

# Извлечение данных с помощью групп



## Пример

```
import re

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

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

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

## Пояснения

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

# Использование негруппирующих скобок



## Пример

```
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)
```

## Пояснения

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



# ВОПРОСЫ





# Функции модуля re



## Функция `re.match()`

Эта функция проверяет, начинается ли строка с заданного шаблона.

Если совпадение найдено, функция возвращает объект `Match`, который содержит информацию о совпадении, иначе `None`

# Важно



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

```
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.")
```



## Функция `re.search()`

Эта функция ищет первое совпадение регулярного выражения в любой части строки и возвращает объект `Match`

# Пример

```
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

Этот флаг сокращённо обозначается `re.I`, он делает поиск регистронезависимым — шаблон будет находить совпадения в любом регистре, даже если написан в нижнем или верхнем

# Пример

```
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()`

Эта функция ищет все совпадения регулярного выражения в строке и возвращает итератор объектов `Match`

# Пример

```
import re

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

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

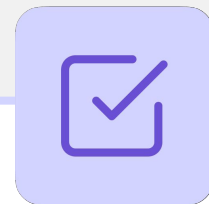
for match in matches:
    print("Объект Match:", match)
    print("Само совпадение:", match.group())
    print("Диапазон совпадения:", match.span())
    print()
```



## Функция `re.split()`

Эта функция разделяет строку на части, используя регулярное выражение как разделитель

# Важно



Функция `re.split()` работает аналогично `str.split()`, но позволяет **разбивать текст по сложным шаблонам**, а не только по одному символу

# Пример

```
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()`

Эта функция заменяет все найденные совпадения на указанную строку или результат функции. Это полезно для форматирования текста, удаления лишних символов и исправления данных

# Функция re.sub()



## Синтаксис

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

## Пояснения

- pattern – шаблон для поиска
- repl – строка, на которую будет заменено совпадение
- string – исходный текст

# Пример

```
import re

text = "apple,  banana ,  orange ,grape"

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

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



## Функция `re.compile()`

Эта функция 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.

# Функция re.compile()



## Синтаксис

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

## Пояснения

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

```
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}")
```



# ВОПРОСЫ





# ЗАДАНИЯ





## 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



## 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

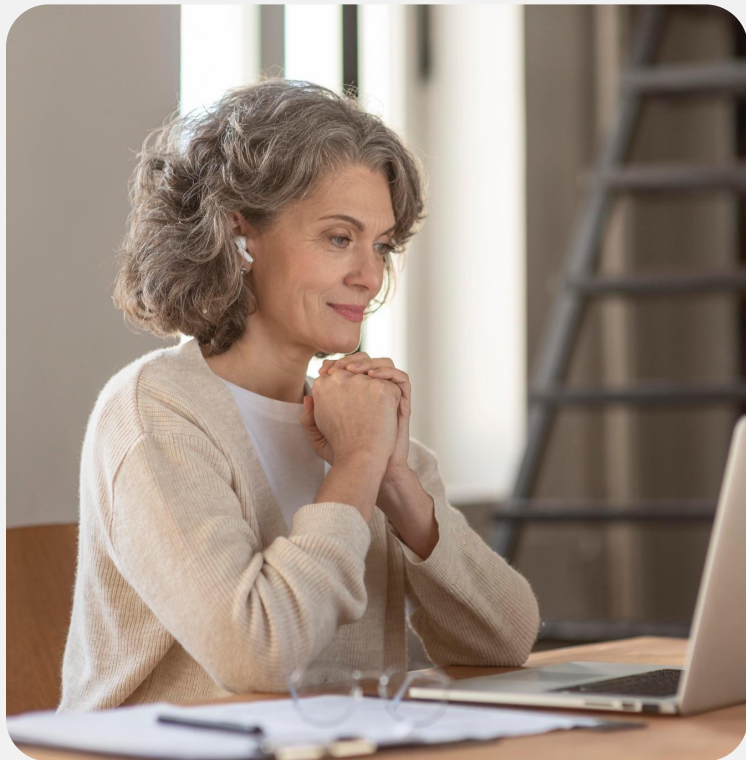
**Ответ:** 1-b, 2-a, 3-c, 4-d



## Выберите верный вариант ответа

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



## Выберите верный вариант ответа

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



# ВОПРОСЫ





# ПРАКТИЧЕСКАЯ РАБОТА



# 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:

Введите пароль: Pass1234

Пароль надёжен.

---

Введите пароль: k2n6bd7

Пароль не соответствует требованиям.

# Извлечение IP-адресов



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

## Данные:

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

## Пример вывода:

```
192.168.1.1
```

```
10.0.0.254
```



# ДОМАШНЕЕ ЗАДАНИЕ



# Домашнее задание

## 1. Извлечение дат

Реализуйте программу, которая должна:

- Найти в тексте все даты в форматах DD/MM/YYYY, DD-MM-YYYY и DD.MM.YYYY.

### Данные:

```
text = "The events N 123456 happened on  
15/03/2025, 01.12.2024 and 09-09-2023.  
Deadline: 28/02/2022."
```

### Пример вывода:

15/03/2025

01.12.2024

09-09-2023

28/02/2022

# Домашнее задание

## 2. Разделение списка тегов

Реализуйте программу, которая должна:

- Прочитать строку с тегами, введёнными пользователем.
- Разделить её на отдельные теги, независимо от того, чем они были разделены (запятые, точки с запятой, слэши или пробелы).
- Удалить лишние пробелы и пустые значения.

**Данные:**

```
tag_input = "python, data-science / machine-learning; AI  neural-networks"
```

**Пример вывода:**

```
['python', 'data-science', 'machine-learning', 'AI', 'neural-networks']
```

## Заключение

