ПМ3 Разработка модулей ПО.

РО 3.1 Понимать и применять принципы объектно-ориентированного и асинхронного программирования.

Тема 4. Регулярные выражения.

Лекция 19. Основы регулярных выражений

Цель занятия:

Познакомиться регулярными выражениями в JavaScript как инструментом для поиска и обработки текста.

Учебные вопросы:

1. Что такое регулярные выражения?
2. Создание регулярных выражений в JavaScript.
3. Флаги регулярных выражений.
4. Методы работы с регулярными выражениями в JavaScript.

1. Что такое регулярные выражения?

Регулярные выражения (RegExp) — это специальный язык шаблонов для работы со строками.

С их помощью можно искать, проверять, заменять и извлекать текст по заданному правилу.

Зачем нужны?

Они помогают автоматизировать работу со строками, там, где обычные методы уже не справляются.

Основные задачи:

- Проверка соответствия шаблону
- Поиск фрагментов текста
- Замена по шаблону
- Разделение строки по сложному разделителю

Когда используются в разработке?

- Валидация данных (email, телефон, пароль)
- Поиск и подсветка текста (поиск в редакторе, фильтрация)
- Обработка логов, CSV, HTML, JSON, markdown
- Нормализация и замена формата данных (например, даты или номера телефона)
- Написание парсеров и простых анализаторов текста

⚡ Итог:

Регулярные выражения — это универсальный инструмент для работы с текстом по правилам.

Они удобны, когда нужно не просто найти подстроку, а проверить или обработать текст по сложному условию.

2. Создание регулярных выражений в JavaScript.

1. Литерал регулярного выражения

- Записывается между косыми слэшами **/.../**
- Флаги указываются после второго слэша.

```
const re1 = /hello/;        // ищет "hello"
const re2 = /hello/i;       // ищет "hello", без учёта регистра
```

✅ Плюсы:

- Кратко и читаемо
- Используется чаще всего

❌ Минус:

- Нельзя динамически формировать строку-шаблон (например, из переменной)

2. Конструктор RegExp

Позволяет создавать регулярку из строки.

```
const word = "cat";
const re = new RegExp(word, "gi");
console.log("Cat catalog".match(re)); // ["Cat", "cat"]
```

✅ Плюсы:

- Удобно, если шаблон нужно собрать из переменных
- Флаги передаются как строка вторым аргументом

❌ Минус:

- Нужно экранировать \ дважды: в строках JS и в регулярке

```
const re = new RegExp("\\d+"); // эквивалент /\d+/
```

Где использовать регулярные выражения?

Регулярки применяются в методах:

- у объекта RegExp → .test(), .exec()
- у строки (String) → .match(), .matchAll(), .replace(), .replaceAll(), .search(), .split()

◇ Итог

- Для статических шаблонов (заранее известные выражения) используют литерал /.../.
- Для динамических шаблонов (собранных из переменных) используют new RegExp().

3. Флаги регулярных выражений.

Флаги меняют поведение поиска. Указываются после /.../

◇ Основные

g — global (глобальный поиск)

По умолчанию регулярное выражение ищет только первое совпадение.

С **g** будут найдены все совпадения.

```
// ищем одну цифру → находит только первую: ["1"]
"a1 b2 c3".match(/\d/);

// то же, но с флагом g → находит все цифры: ["1", "2", "3"]
"a1 b2 c3".match(/\d/g);
```

i — ignore case (без учёта регистра)

Поиск не различает большие и маленькие буквы.

```
/hello/.test("Hello");
// false, потому что ищем строго "hello" в нижнем регистре

/hello/i.test("Hello");
// true, потому что i игнорирует регистр
```

m — multiline (многострочный режим)

Когда в тексте есть перенос строки (**\n**), строка становится многострочной.

```
const text = "one\ntwo";
// это не одна строка "one\ntwo"
// а две строки:
//    1) "one"
//    2) "two"
```

Некоторые проверки работают только для всей строки целиком.

Из-за этого "two" может не находиться, если смотреть на текст как на одну длинную строку.

Флаг m включает режим, в котором каждая строка внутри текста проверяется отдельно.

Теперь "two" можно найти и во второй строке.

```
const text = "one\ntwo";

console.log(text.match(/^two$/));    // null
console.log(text.match(/^two$/m));   // ["two"]
```

Дополнительные (знать полезно, но не обязательно для старта):

- s — dotAll, позволяет . захватывать \n
- u — unicode, корректная работа с юникодом (например, эмодзи)
- y — sticky, ищет только с позиции lastIndex

4. Методы работы с регулярными выражениями в JavaScript.

Регулярные выражения в JS можно применять через методы объекта **RegExp** и строк (**String**).

◇ Методы у RegExp

test(str)

Проверяет: есть ли совпадение?

Возвращает true или false.

```
const re = /abc/;

console.log(re.test("123abc456")); // true (строка содержит "abc")
console.log(re.test("123456"));    // false (нет "abc")
```

exec(str)

Находит совпадение и возвращает подробную информацию:

- совпадение целиком,
- где оно нашлось (индекс),
- исходная строка.

```
const re = /abc/;
console.log(re.exec("123abc456"));

// [ 'abc', index: 3, input: '123abc456', groups: undefined ]
```

Если совпадений нет → null.

◇ Методы у String

match(re)

Ищет совпадения.

- Без флага g → возвращает первое совпадение и подробности.
- С флагом g → возвращает массив всех совпадений.

```
"abc abc".match(/abc/);
// ['abc', index: 0, input: 'abc abc', groups: undefined]

"abc abc".match(/abc/g);   // ['abc', 'abc']
```

search(re)

Возвращает индекс первого совпадения (позицию в строке) или -1, если не найдено.

```
"abc123".search(/123/);   // 3
"abc123".search(/zzz/);   // -1
```

replace(re, newStr)

Заменяет совпадение на другой текст.

По умолчанию заменяет только первое совпадение.
Чтобы заменить все — нужен флаг g.

```
"2025-09-17".replace("-", ".");     // "2025.09-17"
"2025-09-17".replace(/-/g, ".");    // "2025.09.17"
```

split(re)

Разбивает строку по разделителю.

Можно использовать регулярку, если разделителей несколько.

```
"a,b;c".split(/[,;]/);   // ['a', 'b', 'c']
```

📌 Краткие итоги:

- **test** — проверить, есть ли совпадение.
- **match** — получить совпадение/совпадения.
- **search** — узнать позицию.
- **replace** — замена.
- **split** — разбиение строки.

Практические примеры:

1. Проверка имени пользователя

Пусть нужно проверить, что в имени есть слово "Ivan".

```
const re = /Ivan/;

console.log(re.test("Ivan Petrov"));    // true
console.log(re.test("Petr Ivanov"));    // true (тоже содержит "Ivan")
console.log(re.test("Petrov"));         // false
```

2. Поиск всех имён "Anna" (без учёта регистра)

```
const text = "Anna likes coffee. anna goes to school. ANNA is happy.";
console.log(text.match(/anna/gi));
// ['Anna', 'anna', 'ANNA']
```

3. Замена домена в email

```
const email = "student@mail.com";
const result = email.replace(/mail\.com/, "university.edu");
console.log(result);
// "student@university.edu"
```

4. Форматирование телефона

```
const phone = "8-999-123-45-67";
// Заменим дефисы на пробелы:
console.log(phone.replace(/-/g, " "));
// "8 999 123 45 67"
```

5. Поиск года в тексте

```
const text = `я родился в 2000 году. сейчас мне 25 лет.
  Я окончил школу № 123 в 2007 году.`;
console.log(text.match(/\d{4}/g));
```

Как работает /\d{4}/g

- \d → цифра (0–9)
- {4} → ровно 4 раза подряд
- g → ищем все совпадения

6. Разделение списка email-адресов

```
const emails = "anna@mail.com; ivan@mail.com, petr@mail.com";
console.log(emails.split(/[,;]/));
// ["anna@mail.com", " ivan@mail.com", " petr@mail.com"]
```

7. Проверка, есть ли цифры в пароле

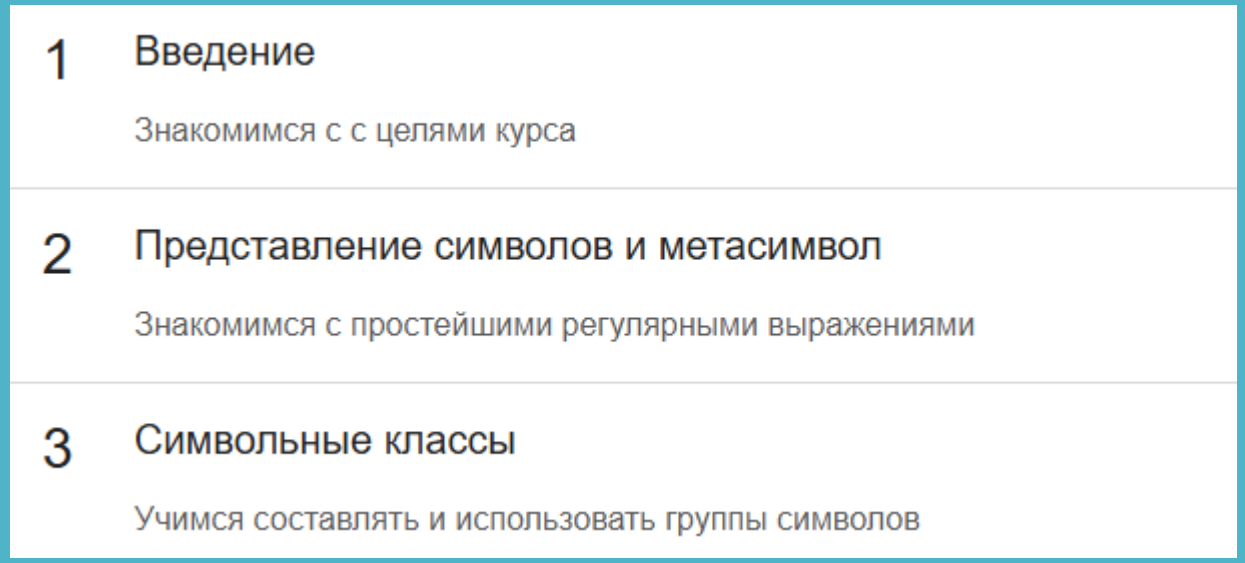
```
const password1 = "qwerty";
const password2 = "qwerty123";

console.log(/\d/.test(password1)); // false
console.log(/\d/.test(password2)); // true
```

Контрольные вопросы:

- Что такое регулярные выражения? Для чего они нужны?
- Какими двумя способами можно создать регулярное выражение в JavaScript?
- Что делают флаги g, i, m?
- Чем отличается test от match?
- В чём разница между replace(/.../, ...) и replace(/.../g, ...)?

Домашнее задание:

1. https://ru.hexlet.io/courses/regexp



2. Повторить материалы лекции.

Материалы лекций:

https://github.com/ShViktor72/Education2025

Обратная связь:

colledge20education23@gmail.com