PHASE 1

WEEK 1

DAY 4

ELBRUS/BOOTCAMP

План

1. Строки (strings)
2. Регулярные выражения (regular expressions)

Strings

Строки

Строковые литералы

- "Это строка"
- 'Это тоже строка'

Специальные символы

Специальные символы и экранирование

- 'Можно переносить\nна новую строку'
- 'Использовать \'кавычки\"
- 'Шестнадцатеричные ASCII коды: \x20'
- 'Unicode-символы: \u2014' // номер Unicode-символа в hex

Шаблонные литералы

Создаются через обратные кавычки (backticks).
В шаблонные строки можно вставлять JS-выражения.

```
const names = ['Олег', 'Анна', 'Майк'];


names.forEach((name) => {
   console.log(`Привет, ${name}`); // ← шаблонный литерал
});
```

Это также называется “интерполяция строк”.

Индексы строки

Получение символа по индексу:

```
const str = 'Строка';
str[1]; // "т"
str.charAt(1); // "т"
str.charCodeAt(1); // 1090 - десятичный номер символа Юникода
```

Строки немутабельные

Нельзя изменить символ в строке:

```
const str = 'Строка';
str[1] = 'o';
// исходная строка НЕ изменится
```

Строки итерируемые

По ним можно пройтись циклом или spread-оператором:

```
const str = 'Строка';
const array = [...str];
console.log(array);
// [ 'С', 'т', 'р', 'о', 'к', 'а' ]
```

toLowerCase, toUpperCase (методы строки)

Создают новую строку на основе старой — но уже в одном регистре:

```
const str = 'Строка';
const lowerCaseStr = str.toLowerCase(); // "строка"
const upperCaseStr = str.toUpperCase(); // "СТРОКА"
```

indexOf, lastIndexOf (методы строки)

```
const str = 'Widget with id';
```

Метод indexOf ищет первое совпадение

```
str.indexOf('id'); // 1
str.indexOf('id', 6); // 12 (поиск с 6-го индекса)
str.indexOf('foo'); // -1 (совпадений нет)
```

Метод lastIndexOf ищет от конца строки к началу

```
str.lastIndexOf('id'); // 12
```

slice (метод строки)

```
const str = "Просто пример";
```

Метод slice возвращает все символы от стартового индекса.

```
const s1 = str.slice(7);    // "пример"
```

Второй параметр — конечный индекс. Его значение не включается.

```
const s2 = str.slice(3, 6); // "сто"
```

substring (метод строки)

```
const str = "Просто пример";
```

Метод substring очень похож на slice.

```
const s3 = str.substring(7);    // "пример"
const s4 = str.substring(3, 6); // "сто"
```

replace (метод строки)

```
const str = 'JavaScript такой такой простой';
```

Метод replace возвращает новую строку с заменой первого совпадения на шаблон.

```
const newStr = str.replace('такой', 'не');
// "JavaScript не такой простой"
```

replace (метод строки)

Если шаблон — регулярное выражение, то можно заменить **все** совпадения с помощью флага g (global)

```
str.replace(/такой/g, 'не'); // "JavaScript не не простой"
```

replaceAll (метод строки)

Заменяет все вхождения на предоставленный шаблон.

```
const word = 'Super Word, Word!'

const replaceWord = word.replaceAll('Word', 'World!')

console.log(replaceWord); // Super World!, World!!
```

split / join (метод строки / массива)

Метод split превращает строку в массив, разделяя её на части. Первым параметром в этот метод передаётся разделитель. Удобен, чтобы разделять текст на строки или на слова, либо чтобы вычленять из текста какие-то данные.

```
const text = 'It’s a me, Mario!';
const words = text.split(' ');
console.log(words); // ["It’s", "a", "me,", "Mario!"]
```

Метод join противоположен split: превращает массив в строку, соединяя его элемент через переданный первым параметром разделитель.

```
const text = ["It’s", "a", "me,", "Mario!"];
const words = text.join(' ');
console.log(words); // "It’s a me, Mario!"]
```

RegExp

Регулярные выражения

Регулярные выражения

У вас есть проблема.

Вы решили использовать регулярные выражения, чтобы её решить.

Теперь у вас две проблемы.

Unknown Author

Создание

```
// через конструктор
const regexp1 = new RegExp('шаблон', 'флаги');


// литеральная запись (предпочтительнее)
const regexp2 = /шаблон/g;
```

search (метод строки)

Возвращает индекс первого совпадения в строке по регулярному выражению.

```
'Я люблю JavaScript!'.search(/лю/); // 2
```

match (метод строки)

Возвращает результат сопоставления строки с регулярным выражением.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/match

test (метод регулярного выражения)

Возвращает булевое значение, совпала ли строка с выражением.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/test

Флаги (синтаксис регулярных выражений)

`g (global)`

Поиск всех совпадений, а не только первого.

`i (case insensitive)`

Регистр игнорируется — например, “A” и “a” считаются одним и тем же.

`m (multi line)`

Символы начала и конца строки (`^`, `$`) работают с переносами на новую строку.

Полный список флагов:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions#advanced_searching_with_flags

Наборы символов (синтаксис регулярных выражений)

`[123]` — вхождение любого символа из перечисленных

`[0-9]` — вхождение любого символа из диапазона

`[^123]` — всё, кроме указанных символов

Пример:

```
/[взкд]ол/g   // сработает на слова: вол, кол, дол, зол
```

Наборы символов (синтаксис регулярных выражений)

`.` (точка) — любой символ

`\d` – `[0-9]` `\D` – `[^0-9]`

`\w` – `[a-zA-Z0-9_]` `\W` – `[^a-zA-Z0-9_]`

`\s` – `[\t\n\v\f\r ]` `\S` – `[^\t\n\v\f\r ]`

Экранирование (синтаксис регулярных выражений)

Синтаксис регулярных выражений использует специальные символы:

`[ ] \ ^ $ . | ? * + ( ) { }`

Если нужно найти эти символы как обычный текст, их надо **экранировать** через `\`.

`.` специальный символ — совпадает с любым текстовым символом

`\.` обычный символ — совпадает только с текстовой точкой

Повторения (синтаксис регулярных выражений)

`{n}` — ровно n вхождений

`\d{3}` — срабатывает на 3 цифры подряд — то же самое, что и `\d\d\d`

`{2,4}` — от 2 до 4 вхождений

`{3,}` — 3 и более вхождения

Повторения (синтаксис регулярных выражений)

`+` минимум одно вхождение — `{1,}`

`*` любое количество вхождений — `{0,}`

`?` одно либо ноль вхождений — `{0,1}`

```
/ка*[пй]?от/g    // сработает на: капот, кот, кайот
```

Границы строки (синтаксис регулярных выражений)

^ — начало строки

$ — конец строки

```
/\+7\d/     // найдёт 89+743
/^\+7\d/    // найдёт +75765
/\+7\d$/    // найдёт 64+75
```

Группы, ИЛИ (синтаксис регулярных выражений)

Группы обозначаются круглыми скобками ()

```
/(ко-)+/gi        // Ко-ко-ко-ко-ко-кот, ко-кот, ко-ко-кот
```

ИЛИ обозначается вертикальной линией |

```
/Работать надо (плохо|хорошо)/g  // Корректное использование групп + ИЛИ

/Работать надо хорошо|плохо/g    // Некорректное
```

Группы и replace (метод строки)

Если использовать регулярное выражение с группами в первом параметре метода строки `replace`, можно гибко управлять заменой текста.

Если использовать строку в качестве второго параметра, можно использовать запись `'$1'`, чтобы обратиться к первой группе регулярного выражения.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace#specifying_a_string_as_the_replacement

Группы и replace (метод строки)

Вторым параметром в метод `replace` можно передавать callback-функцию.

В параметрах этой callback-функции также можно получить доступ к группам из регулярного выражения.

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/replace#specifying_a_function_as_a_parameter