Федеральное государственное образовательное бюджетное учреждение высшего образования
«Финансовый университет при Правительстве Российской Федерации»
(Финансовый университет)

Колледж информатики и программирования

ПМ.08 Разработка кода информационных систем

Группа: 3ИСИП-622

УТВЕРЖДАЮ

Председатель предметно-цикловой комиссии информационных систем и программирования

_____ Т.Г. Аксёнова

«____» _____2025 г.

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №8

"Стрелочные функции в JavaScript

Преподаватель

_____Р. Р. Абзалимов

Исполнитель

_____Л. Д. Слепцов

Оценка: _____

«____» _____2025 г.

Москва

2025

Цель работы

Изучить синтаксис и особенности использования стрелочных функций в JavaScript, научиться применять их для решения практических задач.

Ход работы

1. В чём основное отличие стрелочных функций от обычных функций в JavaScript?

Основное отличие в том, что **у стрелочных функций нет своего контекста (this)**. Они наследуют this из внешней области видимости, тогда как у обычных функций this зависит от того, как они вызваны. Также у стрелочных функций нет arguments, super и new.target.

2. Когда можно опустить круглые скобки в стрелочной функции?

Скобки можно опустить, если у стрелочной функции **только один параметр**:

```
const square = x => x * x; // Правильно
const sum = (a, b) => a + b; // Скобки нужны при 2+ параметрах
```

3. Что такое неявный return в стрелочных функциях?

Если тело функции состоит из **одного выражения**, результат этого выражения автоматически возвращается:

```
const add = (a, b) => a + b; // Неявный return
const multiply = (a, b) => { return a * b; }; // Явный return
```

В фигурных скобках {} return писать обязательно.

4. Почему в функции isPrime мы проверяем делители только до корня из числа?

Потому что **если у числа есть делитель больше корня, то есть и меньший**. Достаточно проверять до sqrt(n), чтобы найти все возможные делители, что значительно ускоряет алгоритм.

Пример: для n = 25 делители 1, 5, 25. Достаточно проверить 1...5, так как 25 / 5 = 5.

5. Как работает spread-оператор (...) в функции findMin?

Оператор ... разворачивает массив в отдельные элементы:

const findMin = arr => Math.min(...arr);

Эквивалентно Math.min(arr[0], arr[1], ..., arr[n]). Без ... передать массив напрямую в Math.min() нельзя.

6. Какое преимущество даёт использование метода reduce в функции sumNumbers?

Метод reduce() позволяет **пройти по всему массиву и аккумулировать результат в одной строке**:

const sumNumbers = arr => arr.reduce((sum, num) => sum + num, 0);

Это **короче и производительнее**, чем использовать forEach или for.

7. Почему в функции celsiusToFahrenheit не требуются фигурные скобки?

Если стрелочная функция состоит из **одного выражения**, можно **опустить** {} и return:

const celsiusToFahrenheit = c => (c * 9/5) + 32;

Если добавить {}, то return нужно писать вручную.

8. Как можно переписать функцию getStringLength, используя обычный синтаксис функции?

Стрелочная функция:

```
const getStringLength = str => str.length;
```

Обычная функция:

```
function getStringLength(str) {
   return str.length;
}
```

Обычные функции удобнее, если код сложнее и требует this.

9. В каких случаях использование стрелочных функций может быть нежелательно?

1. **При работе с this** – у стрелочных функций нет своего this, поэтому они не подходят для методов объектов:

```
const obj = {
   value: 10,
   getValue: () => this.value // `this` будет `undefined`
};
```

2. **В качестве методов классов** – из-за отсутствия this методы на стрелочных функциях не работают.
3. **При использовании arguments** – в стрелочных функциях его нет.
4. **Когда нужна возможность вызвать с new** – стрелочные функции не являются конструкторами.

10. Что произойдёт, если в функцию sumNumbers передать пустой массив?

```
const sumNumbers = arr => arr.reduce((sum, num) => sum + num, 0);
console.log(sumNumbers([])); // 0
```

Метод reduce() использует **начальное значение 0**, поэтому при пустом массиве **вернётся 0**, а ошибки не будет.