Программа
«*Современное WEB-программирование*»

Входной ассемент

Лекция 3.

Модуль 3. «Введение в алгоритмизацию и основы языков web-программирования»
(*продолжение*)

и

Модуль 4. «Основы технологий web-программирования»
(*начало*)

Основы программирования на языке JavaScript

- **avaScript (JS)** — интерпретируемый язык программирования для веб-разработки.
- Работает в браузере (клиентская часть) и на сервере (Node.js).
- Позволяет:
  - взаимодействовать с пользователем;
  - изменять содержимое веб-страницы «на лету»;
  - отправлять запросы на сервер;
  - обрабатывать данные в реальном времени.

- **Ключевые особенности:**
  - динамическая типизация;
  - поддержка ООП;
  - асинхронная работа с API;
  - интеграция с DOM.

Введение в JavaScript и основы языка

Основные конструкции:

```
// Переменные
let name = "John";
const age = 25;
var isActive = true;

// Функции
function greet(name) {
    return `Hello, ${name}!`;
}

// Стрелочные функции
const multiply = (a, b) => a * b;
```

Массивы, объекты и методы работы

1. Создание массива:

```
let fruits = ['яблоко', 'банан', 'апельсин'];

let numbers = [1, 2, 3, 4, 5];
```

2. Основные методы:

- push() — добавить элемент в конец;
- pop() — удалить последний элемент;
- shift() — удалить первый элемент;
- unshift() — добавить элемент в начало;
- slice() — извлечь часть массива;
- splice() — изменить содержимое (добавить/удалить элементы);
- forEach() — перебрать элементы;
- map(), filter(), reduce() — функциональные методы для обработки данных.

3. Свойства:

•length — количество элементов;
•индексация начинается с 0.

Пример:

```
fruits.push('груша');
console.log(fruits); // ['яблоко', 'банан', 'апельсин', 'груша']
```

Массивы, объекты и методы работы

Массивы:

```
const numbers = [1, 2, 3, 4, 5];
const fruits = ['apple', 'banana', 'orange'];

// Методы массивов
numbers.map(x => x * 2);        // [2, 4, 6, 8, 10]
numbers.filter(x => x > 3);     // [4, 5]
numbers.reduce((sum, x) => sum + x, 0); // 15
```

Массивы, объекты и методы работы

- **Объект** — коллекция свойств (ключ-значение).

- **Создание объекта:**

```
let user = {
  name: 'Иван',
  age: 30,
  isStudent: false
};
```

- **Доступ к свойствам:**
  - через точку: user.name;
  - через квадратные скобки: user['name’].
- **Методы объекта:** функции, являющиеся свойствами объекта.
- **Добавление и изменение свойств:**

```
user.city = 'Москва'; // добавление
user.age = 31; // изменение
```

Цикл for...in для перебора свойств объекта.

Массивы, объекты и методы работы

Объекты:

```
const person = {
    name: "Alice",
    age: 30,
    greet() {
        return `Hi, I'm ${this.name}`;
    }
};

// Деструктуризация
const { name, age } = person;
```

Работа с DOM и асинхронность

1. DOM (Document Object Model):

•представление HTML-документа в виде дерева элементов;
•позволяет JS изменять структуру, стиль и содержимое страницы;
•основные методы: getElementById(), querySelector(), createElement(), appendChild().

Пример:

```
let elem = document.getElementById('myId');
elem.style.color = 'red’;
```

2. Асинхронность в JS:

•выполнение операций без блокировки основного потока (загрузка данных, таймеры);
•ключевые механизмы:
    •setTimeout(), setInterval() — таймеры;
    •Promise — обработка асинхронных результатов;
    •async/await — синтаксический сахар для работы с Promise.

Пример Promise:

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data));
```

Работа с DOM и асинхронность

DOM Manipulation:

```
// Выбор элементов
const element = document.getElementById('myId');
const elements = document.querySelectorAll('.m'myIdyClass');

// Изменение контента
element.textContent = 'New text';
element.innerHTML = '<strong>Bold text</strong>';

// Создание элементов
const newDiv = document.createElement('div');
document.body.appendChild(newDiv);
```

Обработка событий:

```
element.addEventListener('click', function(event) {
    console.log('Element clicked!');
    event.preventDefault();
});

// Формы
form.addEventListener('submit', (e) => {
    e.preventDefault();
    const data = new FormData(form);
});
```

Работа с DOM и асинхронность

Promise и async/await:

```
// Promise
fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error));

// Async/await
async function fetchData() {
    try {
        const response = await fetch('https://api.example.com/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error:', error);
    }
}
```

Модули, ООП и работа с API

Модули в JavaScript (ES6):
•организация кода в отдельные файлы;
•ключевые директивы: export, import.

Пример:
```
// module.js
export function greet() { return 'Привет!'; }

// main.js
import { greet } from './module.js';
console.log(greet());
```

Классы и ООП:
•синтаксис классов (ES6): class, constructor, методы;
•наследование, инкапсуляция, полиморфизм.

Пример класса:
```
class Animal {
  constructor(name) { this.name = name; }
  speak() { console.log(`${this.name} издаёт звук`); }
}
```

Модули, ООП и работа с API

Работа с API:

•fetch() — стандартный метод для HTTP-запросов;

•axios — библиотека для упрощения работы с API (обещания, интерцепторы).

Пример axios:

```
axios.get('https://api.example.com/users')
  .then(response => console.log(response.data));
```

Работа с API:

```
// Fetch API
const response = await fetch('/api/users', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: 'John' })
});

// Axios (библиотека)
const response = await axios.get('/api/users');
```

Замыкания, области видимости и продвинутые темы

1. Замыкания (closures):

- функция, запоминающая переменные из внешней области видимости;
- используется для сохранения состояния, приватных свойств.
  Пример:

```
function createCounter() {
  let count = 0;
  return function() { return ++count; };
}
let counter = createCounter();
console.log(counter()); // 1
```

2. Области видимости:

•global, local, block (let, const);
•лексическое замыкание.

3. Продвинутые темы:

- •**Итераторы** — объекты для перебора коллекций (Symbol.iterator);
- •**Генераторы** — функции с сохранением состояния (function*);
- •**Прокси** — обёртка для объектов с перехватом операций (new Proxy(target, handler)).

Замыкания, области видимости и продвинутые темы

Замыкания и области видимости:

```
function createCounter() {
    let count = 0; // private variable

    return {
        increment: () => ++count,
        decrement: () => --count,
        getCount: () => count
    };
}

const counter = createCounter();
counter.increment(); // 1
counter.increment(); // 2
```

Продвинутые темы:

```
// Генераторы
function* numberGenerator() {
    yield 1;
    yield 2;
    yield 3;
}

// Итераторы
const iterable = {
    [Symbol.iterator]: function* () {
        yield 1;
        yield 2;
    }
};
```

Работа с файлами, формами, веб-сокетами и оптимизация

1.Работа с формами:

- доступ к полям формы через DOM;
- валидация данных перед отправкой;
- обработка событий (submit, change).

2.Веб-сокеты (WebSocket):

- двусторонний обмен данными в реальном времени;
- отличие от HTTP: постоянное соединение;
- пример использования: чаты, онлайн-игры.

Пример 1:

```
let socket = new WebSocket('ws://example.com/socket');
socket.onmessage = function(event) { console.log(event.data); };
```

Пример 2:

```
const socket = new WebSocket('ws://localhost:8080');

socket.onopen = () => {
    socket.send('Hello Server!');
};

socket.onmessage = (event) => {
    console.log('Message from server:', event.data);
};

socket.onclose = () => {
    console.log('Connection closed');
};
```

Работа с файлами, формами, веб-сокетами и оптимизация

3. Оптимизация производительности:

- минимизация DOM-манипуляций;
- кэширование результатов вычислений;
- использование Web Workers для тяжёлых вычислений;
- анализ производительности в DevTools.

Пример:

```
// Дебаунсинг
function debounce(func, wait) {
    let timeout;
    return function executedFunction(...args) {
        const later = () => {
            clearTimeout(timeout);
            func(...args);
        };
        clearTimeout(timeout);
        timeout = setTimeout(later, wait);
    };
}

// Виртуализация списков
// Ленивая загрузка изображений
// Мемоизация дорогих вычислений
```

4. Итог:

- JavaScript — мощный инструмент для создания интерактивных веб-приложений;
- сочетает простоту и широкие возможности;
- требует понимания асинхронности и управления памятью.

Система контроля версий git

Цель:

познакомить с основами работы с Git, его ключевыми возможностями и рабочим процессом.

Проблема множества версий и роль Git

Проблема: при совместной разработке накапливаются множественные версии файлов, теряется история изменений, возникают конфликты.

Примеры проблем:

- разные версии кода у участников команды;
- потеря промежуточных изменений;
- сложность отслеживания, кто и когда что изменил;
- трудности при возврате к рабочим версиям.

Проблема множества версий:

project_v1_final.py
project_v2_new.py
project_v3_final_final.py
project_v4_with_fixes.py

Решение: использование системы контроля версий (VCS — Version Control System):

- Отслеживание истории изменений
- Возможность отката к любой версии
- Параллельная работа нескольких разработчиков
- Автоматическое слияние изменений

Git — распределённая система контроля версий, созданная Линусом Торвальдсом в 2005 г. для разработки ядра Linux.

Ключевые цели Git:

- скорость работы;
- поддержка нелинейного развития (тысячи параллельных веток);
- полная распределённость;
- эффективная работа с крупными проектами.

Основные понятия Git и начало работы

Ключевые термины:

- **Репозиторий** - хранилище проекта с историей
- **Коммит** - фиксация изменений с комментарием
- **Ветвление** - параллельные линии разработки

Начало работы:

```
# Настройка пользователя
git config --global user.name "Ваше Имя"
git config --global user.email "your@email.com"

# Создание репозитория
git init                    # новый репозиторий
git clone https://github.com/user/repo.git  # клонирование
```

Базовый workflow:

```
Рабочая директория → Staging Area → Репозиторий
    (modified)      (staged)     (committed)
```

Работа с коммитами, ветками и конфликтами

Основные команды:

```
git status                  # статус файлов
git add file.txt            # добавление в staging
git commit -m "Описание"    # создание коммита
git log                     # история коммитов
```

Ветвление:

```
git branch feature-branch     # создание ветки
git checkout feature-branch   # переключение
git merge feature-branch      # слияние в основную ветку
```

Разрешение конфликтов:

```
<<<<<<< HEAD
Ваша версия кода
=======
Версия из удаленного репозитория
>>>>>>> branch-name
```

- Ручное редактирование конфликтующих участков
- Коммит разрешенной версии

GitHub и инструменты совместной работы

Работа с удаленными репозиториями:

```
git remote add origin https://github.com/user/repo.git
git push -u origin main       # отправка изменений
git pull                      # получение изменений
git fetch                     # загрузка без слияния
```

Современные сервисы:

- **GitHub** - самый популярный, социальные функции
- **GitLab** - self-hosted решения, CI/CD
- **Bitbucket** - интеграция с Jira, бесплатные private репозитории

.gitignore:

```
# Игнорирование файлов
node_modules/
*.log
.env
.DS_Store
```

- Автоматическое исключение ненужных файлов
- Шаблоны для разных языков программирования

Продвинутые техники и DevOps

Stash - временное хранение:

```
git stash                # сохранить незакоммиченные изменения
git stash pop              # восстановить последние изменения
git stash list             # список сохраненных изменений
```

Рабочие процессы:

- **Git Flow** - строгая модель ветвления
- **GitHub Flow** - упрощенный процесс
- **Trunk Based Development** - частые коммиты в основную ветку

DevOps и Git:

- Непрерывная интеграция (CI)
- Автоматическое тестирование
- Деплоймент по коммитам
- Code review через Pull Requests

Лучшие практики:

- Частые маленькие коммиты
- Осмысленные сообщения коммитов
- Регулярная синхронизация с удаленным репозиторием
- Code review перед слиянием

Введение в базы данных PostgreSQL

Цель:
познакомить с основами работы с PostgreSQL, ключевыми понятиями моделирования данных и SQL.

Что такое база данных:

- Организованная коллекция данных
- Система управления базами данных (СУБД)
- Реляционные vs NoSQL базы данных

Модели данных:

- **Иерархическая** - древовидная структура
- **Сетевая** - сложные связи между объектами
- **Реляционная** - таблицы с отношениями
- **Объектно-ориентированная** - объекты и классы
- **Документная** - коллекции документов (NoSQL)

PostgreSQL:

- Продвинутая реляционная СУБД с открытым исходным кодом
- Поддержка ACID (атомарность, согласованность, изолированность, долговечность)
- Расширяемость и поддержка JSON

ER-диаграммы и проектирование базы данных

Сущности и атрибуты:

[Студент]	[Курс]
id (PK)	id (PK)
имя	название
фамилия	описание
email	

Типы связей:
•**1:1** (один к одному) - Паспорт ↔ Человек
•**1:N** (один ко многим) - Автор ↔ Книги
•**N:M** (многие ко многим) - Студенты ↔ Курсы

Ограничения:
•PRIMARY KEY - первичный ключ
•FOREIGN KEY - внешний ключ
•UNIQUE - уникальность
•NOT NULL - обязательное поле
•CHECK - проверка условия

SQL - создание схемы и работа с данными

DDL (Data Definition Language):

```
-- Создание таблицы
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE,
    birth_date DATE,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Создание связи
CREATE TABLE student_courses (
    student_id INTEGER REFERENCES students(id),
    course_id INTEGER REFERENCES courses(id),
    PRIMARY KEY (student_id, course_id)
);
```

DML (Data Manipulation Language):

```
-- Вставка данных
INSERT INTO students (name, email)
VALUES ('Иван Петров', 'ivan@example.com');

-- Обновление
UPDATE students SET email = 'new@email.com'
WHERE id = 1;

-- Удаление
DELETE FROM students WHERE id = 1;
```

SQL запросы и представления

Базовые запросы:

```sql
-- SELECT с фильтрацией
SELECT name, email
FROM students
WHERE birth_date > '2000-01-01'
ORDER BY name ASC;

-- JOIN таблиц
SELECT s.name, c.title
FROM students s
JOIN student_courses sc ON s.id = sc.student_id
JOIN courses c ON c.id = sc.course_id;

-- Агрегатные функции
SELECT
    COUNT(*) as total_students,
    AVG(age) as average_age
FROM students;
```

Представления (Views):

```sql
CREATE VIEW student_course_view AS
SELECT
    s.name as student_name,
    c.title as course_title,
    c.description
FROM students s
JOIN student_courses sc ON s.id = sc.student_id
JOIN courses c ON c.id = sc.course_id;

-- Использование представления
SELECT * FROM student_course_view;
```

Расширенные возможности - функции и триггеры

Пользовательские функции:

```
-- Функция для расчета возраста
CREATE OR REPLACE FUNCTION
calculate_age(birth_date DATE)
RETURNS INTEGER AS $$
BEGIN
    RETURN EXTRACT(YEAR FROM AGE(birth_date));
END;
$$ LANGUAGE plpgsql;

-- Использование функции
SELECT name, calculate_age(birth_date) as age
FROM students;
```

Триггеры:

```
-- Триггер для логирования изменений
CREATE OR REPLACE FUNCTION log_student_changes()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO student_audit (student_id, change_type,
changed_at)
    VALUES (NEW.id, 'UPDATE', NOW());
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER student_update_trigger
    AFTER UPDATE ON students
    FOR EACH ROW EXECUTE FUNCTION
log_student_changes();
```

ORM и NoSQL в PostgreSQL

ORM (Object-Relational Mapping):

- Преобразование объектов в реляционные данные
- Популярные ORM:
  - **Sequelize** (JavaScript)
  - **SQLAlchemy** (Python)
  - **Hibernate** (Java)
  - **Entity Framework** (C#)

Пример Sequelize:

```
const Student = sequelize.define('Student', {
    name: { type: DataTypes.STRING },
    email: { type: DataTypes.STRING }
});

// Автоматическое создание SQL
await Student.create({ name: 'John', email: 'john@example.com' });
```

NoSQL возможности в PostgreSQL:

```
-- Работа с JSON
INSERT INTO products (id, data) VALUES (1,
    '{"name": "Laptop", "specs": {"ram": "16GB", "storage": "512GB"}}'
);

-- Запрос по JSON полю
SELECT * FROM products
WHERE data->'specs'->>'ram' = '16GB';
```

Оптимизация и лучшие практики

Индексы для производительности:

```
-- Создание индексов
CREATE INDEX idx_students_email ON students(email);
CREATE INDEX idx_students_name ON students(name);
CREATE INDEX idx_courses_title ON courses(title);

-- Составные индексы
CREATE INDEX idx_student_courses_composite
ON student_courses(student_id, course_id);
```

Лучшие практики:

- Нормализация базы данных (1NF, 2NF, 3NF)
- Правильное использование транзакций
- Резервное копирование и репликация
- Мониторинг производительности
- Безопасность: права доступа, инъекции

Транзакции:

```
BEGIN TRANSACTION;
    UPDATE accounts SET balance = balance - 100 WHERE id = 1;
    UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
-- или ROLLBACK в случае ошибки
```