

Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кузбасский государственный технический университет
имени Т. Ф. Горбачева»

Институт профессионального образования
Кафедра информатики и информационных систем

Юлия Сергеевна Дементьева

ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ

Методические материалы к практическим занятиям,
лабораторным и самостоятельным работам

Рекомендовано цикловой методической комиссией
специальности СПО 09.02.07 Информационные системы и
программирование в качестве электронного издания
для использования в образовательном процессе

Кемерово 2024

Рецензент: Семенова О.С. –заведующий кафедрой информатики и информационных систем ИПО, доцент кафедры эксплуатации автомобилей ФГБОУ ВО «Кузбасский государственный технический университет имени Т. Ф. Горбачева»

Дементьева, Ю.С. Проектирование и разработка веб-приложений: методические материалы к практическим занятиям, лабораторным и самостоятельным работам: для студентов специальности СПО 09.02.07 «Информационные системы и программирование» /сост. Ю.С. Дементьева, Кузбасский государственный технический университет имени Т. Ф. Горбачева. – Кемерово, 2024. – Текст: электронный.

Методические материалы к практическим занятиям, лабораторным и самостоятельным работам по курсу «Проектирование и разработка веб-приложений» позволяют углубить знания, полученные в ходе аудиторных занятий; способствуют закреплению теоретических положений; развивают навыки по их практическому применению.

© Кузбасский государственный
технический университет
имени Т. Ф. Горбачева
© Дементьева Ю.С.,
составление, 2024

СОДЕРЖАНИЕ

| | |
|---|-----|
| ПРАКТИЧЕСКАЯ РАБОТА №1. СОСТАВЛЕНИЕ АНКЕТЫ И СБОР МАТЕРИАЛА ДЛЯ КОНКРЕТНОЙ ЗАДАЧИ ВЕБ-ПРИЛОЖЕНИЯ. ОПИСАНИЕ ЗАДАЧИ НА ЯЗЫКЕ UML..... | 5 |
| ПРАКТИЧЕСКАЯ РАБОТА №2. МОДЕЛИРОВАНИЕ ПРОЦЕССА РАЗРАБОТКИ ИНФОРМАЦИОННОГО РЕСУРСА СОВРЕМЕННЫМ ПО СРЕДСТВАМИ BPWIN ИЛИ ALLFUSION PROCESS MODELER | 20 |
| ПРАКТИЧЕСКАЯ РАБОТА №3. ПОСТРОЕНИЕ СЕТЕВОГО ГРАФИКА РАЗРАБОТКИ ВЕБ-ПРОЕКТА В MS PROJECT | 28 |
| ПРАКТИЧЕСКАЯ РАБОТА №4. СОЗДАНИЕ ПРОСТЕЙШИХ ПРОГРАММ НА JS: ВЫВОД РЕЗУЛЬТАТОВ И ВВОД ДАННЫХ, ПЕРЕМЕННЫЕ, КОНСТАНТЫ, ОПЕРАТОРЫ JAVASCRIPT..... | 36 |
| ПРАКТИЧЕСКАЯ РАБОТА №5. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ УСЛОВНЫХ ОПЕРАТОРОВ И ЦИКЛОВ..... | 44 |
| ПРАКТИЧЕСКАЯ РАБОТА №6. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ | 55 |
| ПРАКТИЧЕСКАЯ РАБОТА №7. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТНОЙ МОДЕЛИ ДОКУМЕНТА..... | 61 |
| ПРАКТИЧЕСКАЯ РАБОТА №8. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ПОЛЬЗОВАТЕЛЬСКИХ ОБЪЕКТОВ..... | 68 |
| ПРАКТИЧЕСКАЯ РАБОТА №9. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ЭЛЕМЕНТОВ ФОРМЫ..... | 76 |
| ПРАКТИЧЕСКАЯ РАБОТА №10. СОЗДАНИЕ ПРОГРАММ НА СЕРВЕРНОМ ЯЗЫКЕ ПРОГРАММИРОВАНИЯ, СОДЕРЖАЩИХ КОНСТАНТЫ, ПЕРЕМЕННЫЕ, ОПЕРАТОРЫ. 82 | |
| ПРАКТИЧЕСКАЯ РАБОТА №11. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ВЕТВЛЕНИЙ, ЦИКЛОВ И МАССИВОВ... 92 | |
| ПРАКТИЧЕСКАЯ РАБОТА №12. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ И ФУНКЦИЙ..... | 101 |

| | |
|---|-----|
| ПРАКТИЧЕСКАЯ РАБОТА №13. ОБРАБОТКА ДАННЫХ ФОРМЫ..... | 114 |
| ПРАКТИЧЕСКАЯ РАБОТА №14. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ СПЕЦИАЛЬНЫХ ФУНКЦИЙ..... | 125 |
| ПРАКТИЧЕСКАЯ РАБОТА №15. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ | 131 |
| ПРАКТИЧЕСКАЯ РАБОТА №16. SQL – ЗАПРОСЫ И ИХ ОБРАБОТКА С ПОМОЩЬЮ PHP | 139 |
| ЛАБОРАТОРНАЯ РАБОТА №1. ОСНОВЫ РАБОТЫ С JS-ФРЕЙМВОРКОМ: VUE.JS. ОСНОВЫ ДОСТУПА К ДАННЫМ. ВЫВОД ИНФОРМАЦИИ НА СТРАНИЦУ | 147 |
| ЛАБОРАТОРНАЯ РАБОТА №2. СОЗДАНИЕ ДИНАМИЧЕСКИХ АТТРИБУТОВ. ПОДКЛЮЧЕНИЕ И ОБРАБОТКА СОБЫТИЙ..... | 154 |
| ЛАБОРАТОРНАЯ РАБОТА №3. СОЗДАНИЕ ПРОСТЫХ КОМПОНЕНТОВ И ИХ ИСПОЛЬЗОВАНИЕ НА СТРАНИЦЕ | 161 |
| ЛАБОРАТОРНАЯ РАБОТА №4. РАБОТА СО СЛОЖНЫМИ КОМПОНЕНТАМИ | 169 |
| ЛАБОРАТОРНАЯ РАБОТА №5. РЕАЛИЗАЦИЯ ПРОГРАММ ПО МЕТОДОЛОГИИ ООП СРЕДСТВАМИ СЕРВЕРНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ..... | 178 |
| ЛАБОРАТОРНАЯ РАБОТА №6. СОЗДАНИЕ БАЗЫ ДАННЫХ MYSQL С ПОМОЩЬЮ УТИЛИТЫ PHPMYSQLADMIN | 193 |
| ЛАБОРАТОРНАЯ РАБОТА №7. РАЗРАБОТКА МОДУЛЯ АВТОРИЗАЦИИ | 200 |
| ЛАБОРАТОРНАЯ РАБОТА №8. РАЗРАБОТКА МОДУЛЯ РЕГИСТРАЦИИ | 209 |
| САМОСТОЯТЕЛЬНАЯ РАБОТА №1 | 222 |
| САМОСТОЯТЕЛЬНАЯ РАБОТА №2 | 223 |
| САМОСТОЯТЕЛЬНАЯ РАБОТА №3 | 224 |
| СПИСОК ЛИТЕРАТУРЫ | 225 |

ПРАКТИЧЕСКАЯ РАБОТА №1. СОСТАВЛЕНИЕ АНКЕТЫ И СБОР МАТЕРИАЛА ДЛЯ КОНКРЕТНОЙ ЗАДАЧИ WEB-ПРИЛОЖЕНИЯ. ОПИСАНИЕ ЗАДАЧИ НА ЯЗЫКЕ UML

Цель работы: изучение основ создания диаграмм состояний на языке UML, получение навыков построения диаграмм состояний, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

После запуска ArgoUML появится окно. Окно ArgoUML состоит из следующих компонентов: меню, панели инструментов и четырех панелей. Вверху слева находится Explorer, затем идет панель редактирования, затем на нижней панель деталей и To-Do панель.

В панели Explorer содержится список всех классов, интерфейсов и типов данных модели в виде дерева. На панели редактирования можно редактировать диаграммы.

В панели To-Do отображаются элементы моделей.

Описание пунктов меню:

Файл (File) — позволяет создавать новые проекты, сохранять и открывать проекты, импортировать, импортировать источники из другого расположения, скачать и сохранить модель из/в базу данных, печать модель, сохранить модель, сохранять конфигурацию модели и выход из программы.

Редактировать (Edit) -позволяет выбрать один или несколько UML элементов в диаграмме, а также повторить действия, перемещать элементы из диаграммы и так далее

Вид (View) -позволяет переключаться между диаграммами, искать артефакты в модели, масштабировать диаграмму, выберите особенное представление диаграмм и т.п.

Создать диаграмму (Create Diagram) -позволяет создавать любую диаграмму из семи UML диаграмм, поддерживаемых ARGOUML (class, use case, state, activity, collaboration, deployment и sequence).

Расставить (Arrange) – позволяет выровнять, распределить, упорядочить артефакты в диаграмме.

Генерация кода (Generation) -позволяет сгенерировать Java код для отдельных или всех классов.

Критика (Critique) –переключать авторецензирование, устанавливать уровень важности проблемы дизайна и дизайн целей и увидеть критических изменения. Инструменты (Tools) Помощь (Help)

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №1

Задание 1. Для создания диаграммы вариантов использования, выполните следующие действия:

- 1) Дважды щелкните на значке untitledModel.
- 2) Нажмите на значок Диаграмма вариантов использования в проводнике, чтобы открыть диаграмму Вариантов использования.
- 3) С помощью кнопки Use Case (вариант использования) на панели инструментов поместите на диаграмму новый вариант использования.
- 4) Назовите этот новый вариант «Ввести новый заказ», для этого вам нужно дважды нажать на варианте использования и ввести имя или активировать вкладку Свойства To-Do панели в поле Имя: введите имя.
- 5) Повторите шаги 3 и 4, чтобы разместить на диаграмме остальные варианты.
- 6) С помощью кнопки Actor (Действующее лицо) инструментов поместите на диаграмму новое действующее лицо
- 7) Назовите его «Разработчик», действия аналогичные вариантам использования.
- 8) Повторите шаги 6 и 7, поставив на диаграмме остальных актеров.

Задание 2. Для указания абстрактного варианта использования, выполните следующие действия:

- 1) ЛКМ нажмите на вариант использования «Отклонить заказ» на диаграмме.

2) В панели Свойств в области modifiers выберите контрольный переключатель isAbstract (Абстрактный), чтобы сделать этот вариант использования абстрактным.

Задание 3. Для добавления ассоциации выполните следующие действия:

1) Нажав на кнопку Association (Ассоциация), выберите UniAssociation (Однонаправленная Ассоциация), изобразите ассоциацию между актером Продавец и вариантом использования «Ввести новый заказ».

2) Повторите этот шаг, чтобы разместить на диаграмме остальные ассоциации.

Задание 4. Для добавления связи расширения выполните следующие действия:

Нажав на кнопку Extend на панели инструментов нарисуйте связь между вариантом использования «Отклонить заказы» и вариантом использования «Оформить заказ». Стрелка должна быть протянута от первого варианта использования ко второму. Связь расширения означает, что вариант использования «Отклонить заказ», при необходимости, дополняет функциональные возможности варианта использования «Оформить заказ». Диаграмма должна иметь вид, как на рисунке 1.1.

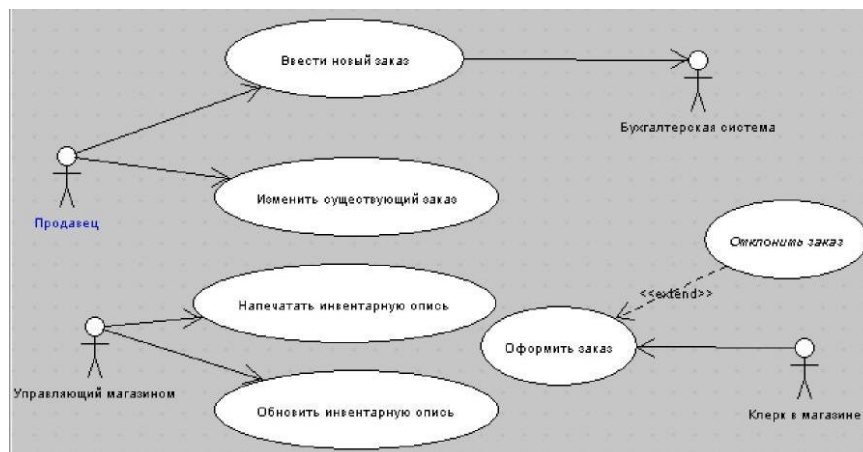


Рисунок 1.1 – Диаграмма вариантов использования для системы обработки заказов

Задание 5. Создать главную диаграмму классов.

1) Дважды щелкните на значке UntitledModel, отобразится содержимое пакета. Выберите Диаграмма классов. С помощью кнопки Новый пакет и панели инструментов разместите на диаграмме новый пакет. На вкладке Свойства, укажите имя пакета Entities (Сущность).

2) Создайте пакеты Boundaries (Границы) и Control (Управление) (рис. 1.2).

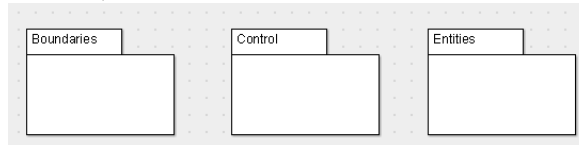


Рисунок 1.2 – Главная диаграмма классов системы обработки заказов

Задание 6. Создание диаграммы классов для сценария «Ввести новый заказ» со всеми классами

1) В меню Создать диаграмму, выберите Диаграмма классов.

2) Во вкладке Свойства введите имя для новой диаграммы класса Add New Order (Ввод нового заказа).

3) Щелкните в браузере на этой диаграмме, чтобы открыть ее.

4) Создайте в окне диаграммы классов классы OrderOptions (Выбор заказа), OrderDetail (Детали заказа), Order (Заказ), OrderMgr (Менеджер заказов) и TransactionMgr (Менеджер транзакций). Имя класса вводится во вкладке Свойства.

5) Добавьте операции, указанные на диаграмме классов (рис.1.3). Чтобы сделать это, выберите соответствующий класс, нажмите на кнопку Добавить операцию, здесь вы можете ввести видимость операции. Диаграмма классов должна выглядеть как на рисунке 3.

Задание 7. Добавление стереотипов в классы

1) Нажмите на класс OrderOptions.

2) Введите в поле имя класса стереотип <<Boundary>>.

3) В раскрывающемся списке стереотипов теперь будет стереотип Boundary. Укажите его.

4) Нажмите с помощью мыши на классе OrderDetail.

5) Из раскрывающегося списка Стереотип выберите стереотип Boundary.

6) Свяжите классы OrderMgr и TransactionMgr со стереотипом Control, а класс Order – со стереотипом Entity. Диаграмма классов должна выглядеть как на рисунке 1.3.

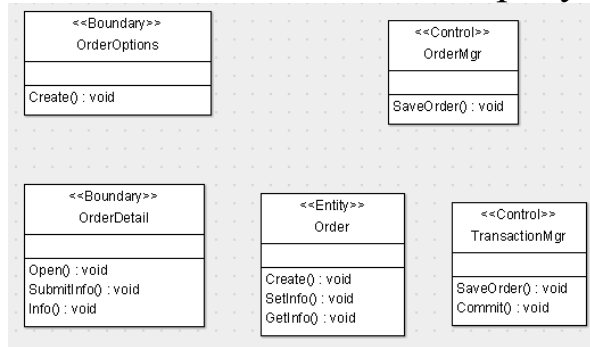


Рисунок 1.3 – Стереотипы классов для варианта использования «Ввести новый заказ»

Задание 8. Объединение классов в пакеты

- 1) Перетащите в браузер класс OrderDetail на пакет Boundaries.
- 2) Перетащите класс OrderOptions на пакет Boundaries.
- 3) Перетащить классы OrderMgr и TransactionMgr на пакет Control.
- 4) Перетащите класс Order на пакет Entities.

Задание 9. Добавить диаграмму классов к каждому пакету

- 1) Дважды щелкните левой кнопкой на пакете Boundaries в окне диаграмм.
- 2) В появившемся окне выберите Да.
- 3) Во вкладка Свойства, введите имя для новой диаграммы – MainB.
- 4) Щелкните на диаграмму, чтобы открыть его.
- 5) В браузере выделите класс OrderOptions (потом OrderDetail), щелкните ПКМ и выберите опцию «Добавить в диаграмму», а затем нажмите мышью в поле диаграммы.
- 6) Классы будут отображаться на диаграмме.
- 7) Выполните эти же шаги, чтобы поместить классы OrderMgr и TransactionMgr на главную диаграмму классов пакета Control (MainC).

8) Выполните эти же действия, чтобы поместить класс Order на главную диаграмму классов пакета Entities (MainE).


Задание 10. Для того чтобы создать диаграммы кооперации выполните следующие действия:

- 1) Нажмите на значок untitledModel в браузере.
- 2) Из меню Создать диаграмму, чтобы выберите Диаграмму коопераций.
- 3) Назовите эту диаграмму Ввод заказа.
- 4) Нажмите на нее, чтобы открыть.

Задание 11. Для добавления объектов к диаграмме коопераций, выполните следующие действия:

- 1) Перетащите с диаграммы вариантов использования объект Продавец на рабочую область.
- 2) На панели инструментов нажмите кнопку ClassifierRole.
- 3) Щелкните в любом месте диаграммы, чтобы разместить там новый объект.
- 4) Назовите объект «Order Options Form».
- 5) Помести на диаграмме остальные объекты: Order Detail Form, Order Manager, Transaction Manager, Order #1234.

Задание 12. Добавления сообщений на диаграмму коопераций:

- 1) На панели инструментов нажмите кнопку Новая ассоциация.
- 2) Свяжите Продавец с объектом Order Options Form.
- 3) Соедините остальные объекты.
- 4) Нажмите на связь между Продавцом и Order Options Form. На панели инструментов нажмите кнопку Добавить сообщение. Нажмите на связь между Продавцом и Order Options Form. Выбрав сообщение, введите его имя Create ().
- 5) Поместите на диаграмме оставшиеся сообщения: Open (), SubmitInfo(), Save (), SaveOrder(), SetInfo (), GetInfo ().
- 6) Нажмите на объекте для добавления к нему сообщения рефлексии. По бокам объекта будут показаны значки, среди которых необходимо выбрать .
- 7) Нажмите на связь рефлексии Transaction Manager, чтобы ввести сообщение. На панели инструментов нажмите кнопку

Добавить сообщение. Назовите новое сообщение Commit () (Сохранить информацию о заказе в базе данных) (рисунок 1.4).

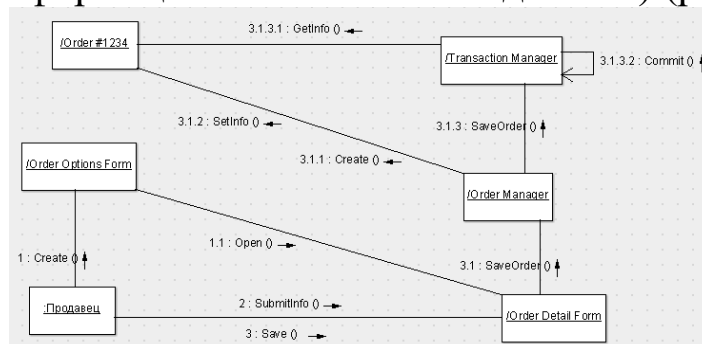


Рисунок 1.4 – Диаграмма коопераций

Задание 13. Для создания схемы последовательностей, выполните следующие действия: щелкните ПКМ логическое представление браузера. В открывшемся меню нажмите Create Diagram → Диаграмма последовательности. Назовите новую диаграмму «Ввод заказ». Дважды щелкните на нем, чтобы открыть.

Задание 14. Для добавления на диаграмму актера и объектов, выполните следующие действия:

1) Перетащите актера Продавец из браузера на диаграмму. Для панели панель инструментов, нажмите кнопку New Classifier Role. Имя объекта Order Options Form – Выбор варианта заказа.

2) Таким же образом добавьте все элементы, описанные в пункте 11.5 (рисунок 1.5).

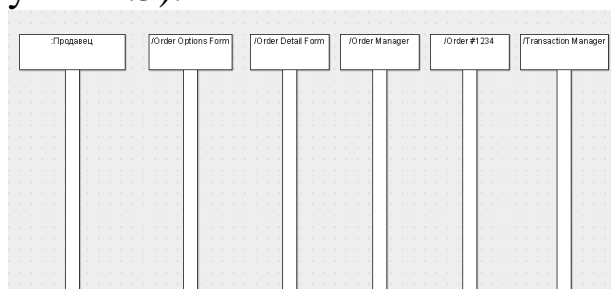


Рисунок 1.5 – Диаграмма последовательности

Задание 15. Для добавления сообщений на диаграмму, выполните следующие действия:

1) На панели инструментов нажмите кнопку Действие отправки, перетащите от линии жизни актера к линии жизни объекта Order Options Form. В ArgoUML в данном виде диаграмм сообщения автоматически не нумеруются, поэтому нумерация

переносится из диаграммы кооперации. Выбрав сообщение, введите имя 1: Create ().

2) Добавьте оставшиеся сообщения, описанные в пункте 12.5

Задание 16. Сопоставление объектов с классами

- 1) Нажмите на объекте Order Options Form.
- 2) В панели Свойств раскройте список База.
- 3) В открывшемся списке нажмите на кнопку с плюсом.
- 4) В открывшемся окне выберите класс OrderOptions и нажмите на кнопку со стрелкой для добавления в список.
- 5) Нажмите кнопку ОК.
- 6) Сопоставьте остальные объекты с классами: Order Detail Form - OrderDetail, Order Manager – OrderMgr, Transaction Manager - TransactionMgr, Order #1234 - Order.

Задание 17. Соотношение сообщений с операциями.

- 1) Нажмите на сообщение 1: Create () заказ.
- 2) В окне Свойств дважды щелкните в поле Действие. В открывшемся окне введите Create ().
- 3) Сопоставьте остальные сообщения.

Задание 18. Добавление нового класса

- 1) Найдите в дереве (браузере) диаграмму классов «Add New Order». Нажмите на нее дважды, чтобы открыть.
- 2) Поместите на этой диаграмме новый класс OrderItem (Позиция заказа). Назначьте этому классу стереотип Entity.
- 3) В дереве перетащите класса в пакет Entities.

Задание 19. Добавление атрибутов

- 1) Щелкните ПКМ на классе Order (Заказ).
- 2) В открывшемся меню выберите Добавить → Новый атрибут.
- 3) Введите новый атрибут OrderNumber: Integer и нажмите Enter. Тип атрибута можно выбрать из раскрывающегося списка в поле Тип или ввести вручную.
- 4) Введите следующий атрибут CustomerName: String.
- 5) Добавьте атрибуты OrderDate : Date и OrderFillDate : Date.

- 6) Щелкните ПКМ на классе OrderItem.
- 7) В открывшемся меню выберите Добавить → Новый атрибут.
- 8) Введите новый атрибут ITEMID: Integer.
- 9) Введите следующий атрибут ItemDescription: String.

Задание 20. Добавления операций к классу OrderItem

- 1) Щелкните ПКМ на классе OrderItem.
- 2) В открывшемся меню выберите Добавить → Новая операция.
- 3) Введите новую операцию Create.
- 4) Введите следующую операцию SetInfo.
- 5) Введите следующую операцию GetInfo.

Задание 21. Подробное описание операций с помощью диаграммы класса

- 1) Нажмите на класс Order, выбрав его таким образом.
- 2) Нажмите на этот класс в разделе операций.
- 3) Отредактируйте операцию Create() так, чтобы она выглядела таким образом: Create () : Boolean. Для этого дважды нажмите на операцию Create в разделе операций класса.
- 4) Отредактируйте операцию SetInfo(), чтобы она выглядела следующим образом: SetInfo(OrderNum : Integer, Customer : String, OrderDate : Date, FillDate : Date) : Boolean
- 5) Отредактируйте операцию GetInfo (), чтобы она выглядела следующим образом: GetInfo () : String

Задание 22. Подробное описание операций остальных классов

- 1) С помощью браузера или диаграмм, введите следующую сигнатуру операций класса OrderDetail: Open () : Boolean, SubmitInfo() : Boolean, Save() : Boolean.
- 2) С помощью браузера или диаграмм, введите следующую сигнатуру операций класса OrderOptions : Create () : Boolean .
- 3) С помощью браузера или диаграмм, следующую сигнатуру операций класса OrderMgr: SaveOrder(ORDERID: Integer): Boolean.

4) С помощью браузера или диаграмм, введите следующую сигнатуру операций класса TransactionMgr: SaveOrder (ORDERID: Integer): Boolean, Commit (): Integer.

Задание 23. Добавление отношения между классами.

Чтобы найти отношения, были изучены диаграммы последовательности. Все классы, которые там взаимодействуют, требовали определения соответствующих связей на диаграммах классов. После выявления связей их добавили в модель. Добавьте отношения, как показано на рисунке 6.

Множественность указывается выбором из контекстного меню связи опции Multiplicity. После добавления связей диаграмма классов должна выглядеть следующим образом.

Задание 24. Создание диаграммы состояний.

- 1) Найдите в браузере класс Order.
- 2) Выделите этот класс.
- 3) В меню Создать диаграмму выберите «Диаграмма состояний».

Задание 25. Добавление начального и конечного состояний.

- 1) На панели инструментов нажмите кнопку New Initial (Начальное состояние). Поместите это состояние на диаграмму.
- 2) На панели инструментов нажмите Конечное состояние. Поместите на диаграмму.

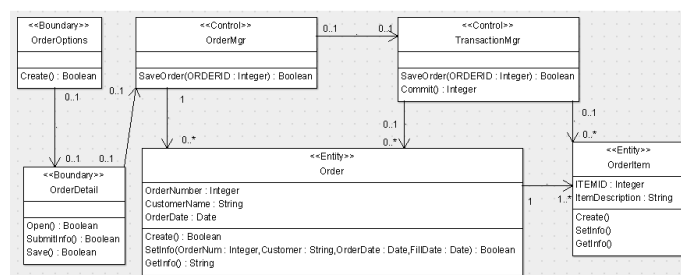


Рисунок 1.6 – Диаграмма классов Add New Order

Задание 26. Добавление суперсостояния.

- 1) На панели инструментов нажмите кнопку Композитное состояние.
- 2) Поместите это состояние на диаграмму.

Задание 27. Добавление оставшихся состояний.

1) На панели инструментов нажмите кнопку Простое состояние. Поместите его на диаграмму. Назовите состояние Отменен.

2) На панели инструментов нажмите кнопку Простое состояние. Поместите его на диаграмму. Назовите состояние Выполнен.

3) На панели инструментов нажмите кнопку Простое состояние. Поместите это состояние внутрь суперсостояния. Назовите состояние Инициализация.

4) На панели инструментов нажмите кнопку Простое состояние. Поместите это состояние внутрь суперсостояния. Назовите состояние Сборка заказа.

Задание 28. Подробное описание состояний

Для добавления в состояния действий на входе, выходе и деятельности исполнения на вкладке Свойства состояния выберите соответственно: действие при входе, действие при выходе, деятельность выполнения.

Задание 29. Добавление переходов

1) На панели инструментов нажмите кнопку New Transition (Переход).

2) Нажмите на начальной точке. Проведите линию перехода к состоянию Инициализация.

3) Повторить предыдущие шаги для добавления оставшихся переходов.

Задание 30. Подробное описание переходов (рис. 1.7)

1) Нажмите на переход от состояния Инициализация к состоянию Сборка заказа, открывая окно его свойств.

2) В поле Имя введите «Выполнить заказ».

3) Повторите этапы, добавив событие «Отменить заказ» для переключения между суперсостоянием и состоянием Отменен.

4) Нажмите на переход от состояния Сборка заказа к состоянию Заполнены (Выполнено), открывая окно его свойств.

5) В поле Имя введите фразу Добавить Порядок пункта (Добавить в заказ на новое место).

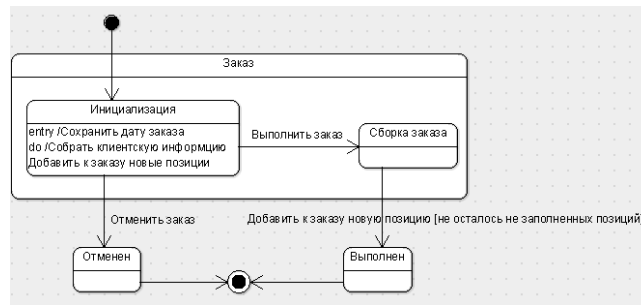


Рисунок 1.7 – Диаграмма состояний

Задание 31. Добавление компонентов и изображение зависимостей (рис. 1.8 – 1.10).

1) Выполните команду Создать диаграмму → Диаграмма развертывания.

2) Нажмите на название диаграммы в браузере.

3) Введите имя Entities.

4) На панели инструментов нажмите кнопку New Component (Компонент).

5) Щелкните в поле диаграммы. Введите имя компонента.

6) Добавить все компоненты.

7) На панели инструментов нажмите кнопку New Dependency (Зависимость).

8) Проведите необходимые зависимости. Аналогично создайте диаграммы Control и Boundaries.

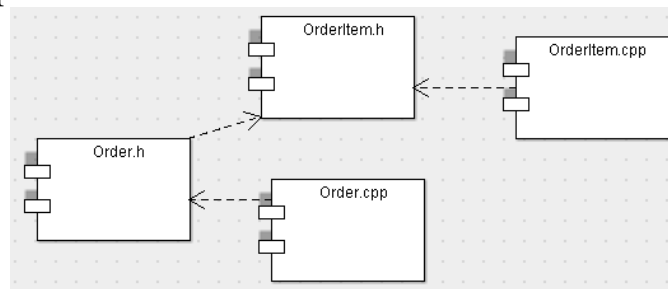


Рисунок 1.8 – Диаграмма компонентов Entities

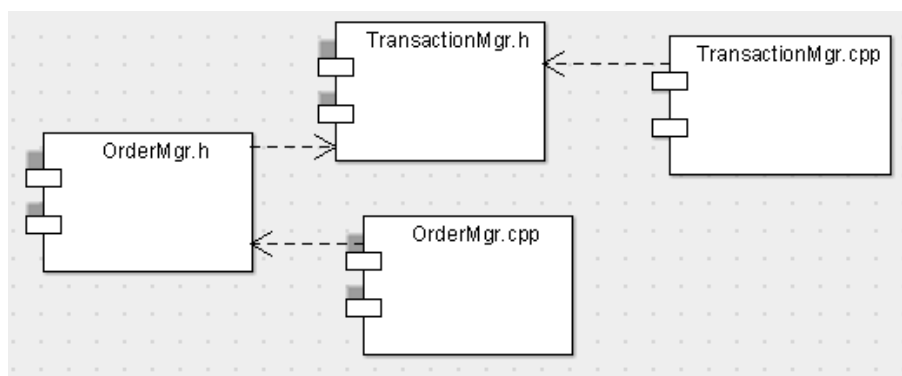


Рисунок 1.9 – Диаграмма компонентов Control

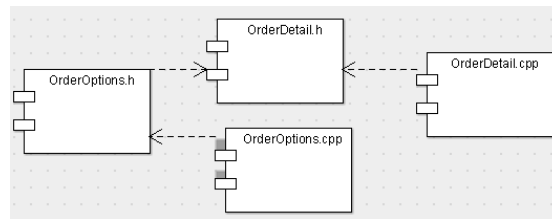


Рисунок 1.10 – Диаграмма компонентов Boundaries

Задание 32. Создание диаграммы компонентов системы

- 1) Выполните команду Создать диаграмму → Диаграмма развертывания.
- 2) Нажмите на название диаграммы Диаграмма компонентов системы.
- 3) Перетащите из дерева на рабочую область диаграммы компоненты OrderDetail.h, OrderOptions.s, OrderMgr.h, TransactionMgr.h, Order.h, OrderItem.h.
- 4) Создайте дополнительные компоненты OrderClient.exe и OrderServer.exe.
- 5) Создайте зависимости между компонентами, которые не были созданы автоматически.

Задание 33. Добавление узлов на диаграмму размещения (рис. 1.11)

- 1) Выберите пункт меню Создать диаграмму → Диаграмма развертывания.
- 2) На панели инструментов нажмите кнопку New Node.
- 3) Введите имя узла «Сервер базы данных».
- 4) Добавьте оставшиеся узлы.

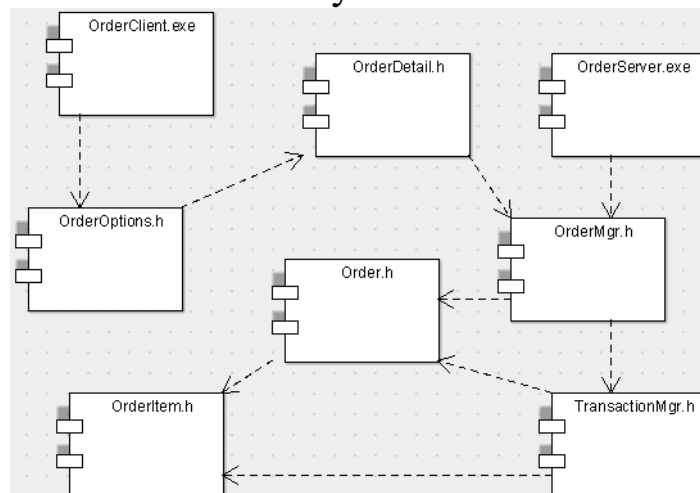


Рисунок 1.11 – Диаграмма компонентов системы

Задание 34. Добавление связей (рис. 1.12)

- 1) На панели инструментов нажмите кнопку New Link.
- 2) Нажмите на «Сервер базы данных».
- 3) Проведите линию до узла «Сервер приложений».
- 4) Добавьте остальные связи.

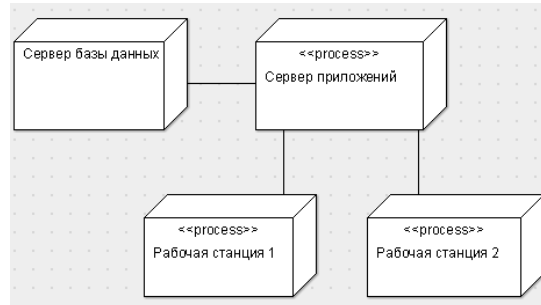


Рисунок 1.12 – Диаграмма размещений

Задание 35. Добавление процессов и операций

1) Щелкните ПКМ на «Сервер приложений». Из раскрывающегося списка выберите Apply Stereotypes → <<process>>.

2) Выполните аналогичные действия для объектов Рабочая станция 1 и Рабочая станция 2.

3) Выделите «Сервер приложений». На панели Свойств в поле Операции нажмите ПКМ и выберите Новая операция. В поле Имя введите OrderServerExe.

4) Добавьте остальные процессы: Рабочая станция 1 – OrderClientExe, Рабочая станция 2 – ATMClientExe.

Задание 36. Результаты выполнения практического задания запишите в отчет.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какова цель работы по составлению анкеты и сбору материала для конкретной задачи веб-приложения?
2. Какие компоненты включает в себя окно ArgoUML, и как они организованы?
3. Какую информацию можно найти на панели Explorer в ArgoUML?
4. Какие действия можно выполнить с помощью панели редактирования в ArgoUML?

5. Какие возможности предоставляет панель To-Do в ArgoUML?
6. Какие основные пункты включает меню "Файл" (File) в ArgoUML, и для чего они используются?
7. Что предоставляет возможность меню "Редактировать" (Edit) в ArgoUML?
8. Какие функции доступны через меню "Вид" (View) в ArgoUML?
9. Какие типы диаграмм можно создать с помощью меню "Создать диаграмму" (Create Diagram) в ArgoUML?
10. Что позволяет выполнить меню "Расставить" (Arrange) в ArgoUML?
11. Какую роль выполняет меню "Генерация кода" (Generation) в ArgoUML?
12. Что предлагает меню "Критика" (Critique) в ArgoUML?
13. Какие инструменты и разделы предоставляет меню "Помощь" (Help) в ArgoUML?

ПРАКТИЧЕСКАЯ РАБОТА №2. МОДЕЛИРОВАНИЕ ПРОЦЕССА РАЗРАБОТКИ ИНФОРМАЦИОННОГО РЕСУРСА СОВРЕМЕННЫМ ПО СРЕДСТВАМИ BPWIN ИЛИ ALLFUSION PROCESS MODELER

Цель работы: овладеть навыками моделирования процесса разработки информационного ресурса с применением выбранного инструмента для проектирования бизнес-процессов

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Создание современных информационных систем (ИС) представляет собой сложнейшую задачу, решение которой требует применения специальных методик и инструментов. Неудивительно, что в последнее время среди системных аналитиков и разработчиков значительно вырос интерес к CASE-технологиям и инструментальным CASE-средствам, позволяющим максимально систематизировать и автоматизировать все этапы разработки программного обеспечения.

Технология создания ИС предъявляет особые требования к методикам реализации и программным инструментальным средствам, а именно:

- наличие эффективных средств автоматизации на ранних этапах реализации проекта с целью исключения исправления ошибок, допущенных на предыдущих стадиях;
- наличие средств координации и управления коллективом разработчиков при реализации крупных проектов;
- наличие инструмента, позволяющего значительно (в несколько раз) уменьшить время разработки ИС;
- гибкость инструментальных средств к изменяющимся требованиям.

На современном рынке средств разработки ИС достаточно много систем, в той или иной степени удовлетворяющих перечисленным требованиям. При структурно-функциональном подходе к проектированию ИС CASE-средство AllFusion Modeling Suite 4.1 (рис. 2.1), разработанное фирмой Computer Associates входит в число лучших на сегодняшний день.

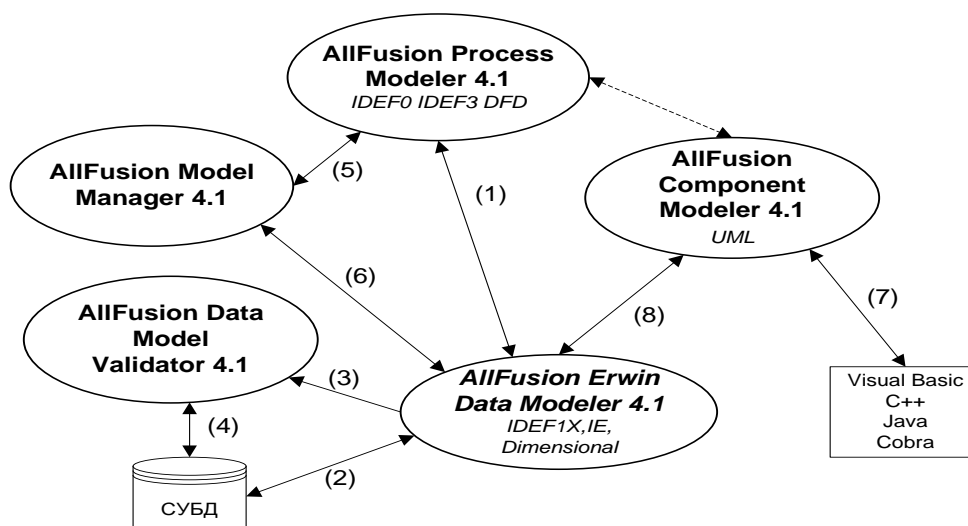


Рисунок 2.1 – Общая схема взаимодействия инструментальных средств AllFusion Modeling Suite 4.1

Для проведения анализа и реорганизации бизнес-процессов предназначено CASE-средство верхнего уровня **AllFusion Process Modeler (BPwin)**, поддерживающее методологии IDEF0 (функциональная модель), IDEF3 (WorkFlow Diagram) и DFD (DataFlow Diagram). Функциональная модель предназначена для описания существующих бизнес-процессов на предприятии (так называемая модель AS-IS) и идеального положения вещей – того, к чему нужно стремиться (модель TO-BE).

Для построения модели данных Computer Associates предлагает мощный и удобный инструмент – **AllFusion ERwin Data Modeler (ERwin)**. ERwin позволяет проводить процессы прямого и обратного проектирования баз данных: по модели данных можно сгенерировать схему БД или автоматически создать модель данных на основе информации системного каталога.

AllFusion Data Model Validator (ERwin Examiner) – основанный на базе знаний инструмент, который позволяет анализировать структуру баз данных с целью выявления и устранения недочетов и ошибок проектирования.

Создание современных ИС требует тесного взаимодействия всех участников проекта: менеджеров, бизнес-аналитиков и системных аналитиков, администраторов баз данных, разработчиков. Для этого использующиеся на разных этапах и разными специалистами средства моделирования и разработки должны быть объединены общей системой организации

совместной работы. Система **ModelMart** – это хранилище моделей, к которому открыт доступ для участников проекта создания ИС.

Автоматическая генерация кода приложения CASE-средствами на основе модели предметной области возможна с помощью **AllFusion Component Modeler (Paradigm Plus)**. Этот инструмент позволяет строить объектные модели и генерировать на основе полученной модели приложения на языках программирования C++, Visual Basic, Java и др.

AllFusion ERwin Data Modeler 4.1 (ERwin) позволяет проектировать, документировать и сопровождать БД и хранилища данных. Создав наглядную модель БД, вы сможете оптимизировать структуру БД и добиться её полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой БД, продуктивность и скорость её разработки.

Основным требованием, предъявляемым проектировщиками БД к решениям для моделирования данных является поддержка полного жизненного цикла разработки приложения. ERwin – это мощное и удобное в использовании средство моделирования данных и проектирования БД, которое эффективно работает на любой стадии жизненного цикла разработки, включая проектирование, генерацию кода и поддержку приложений БД.

Интерфейс ERwin выполнен в стиле Windows-приложений, достаточно прост и интуитивно понятен. Рассмотрим кратко основные функции ERwin по отображению модели и панели инструментов.

ERwin имеет 8 перемещаемых панелей инструментов:

- стандартную (Standard);
- палитру инструментов (ToolBox);
- выбора шрифтов и цветов (Font & Color);
- трансформации таблиц (Transforms);
- рисования (Drawing);
- выравнивания (Alignment);
- работы с сервером БД (Database);
- работы с хранилищем моделей ModelMart.

Каждому уровню отображения модели соответствует своя палитра инструментов.

Окно диаграммы содержит заголовок, рабочую область и соответствующие возможности (рис. 2.2).

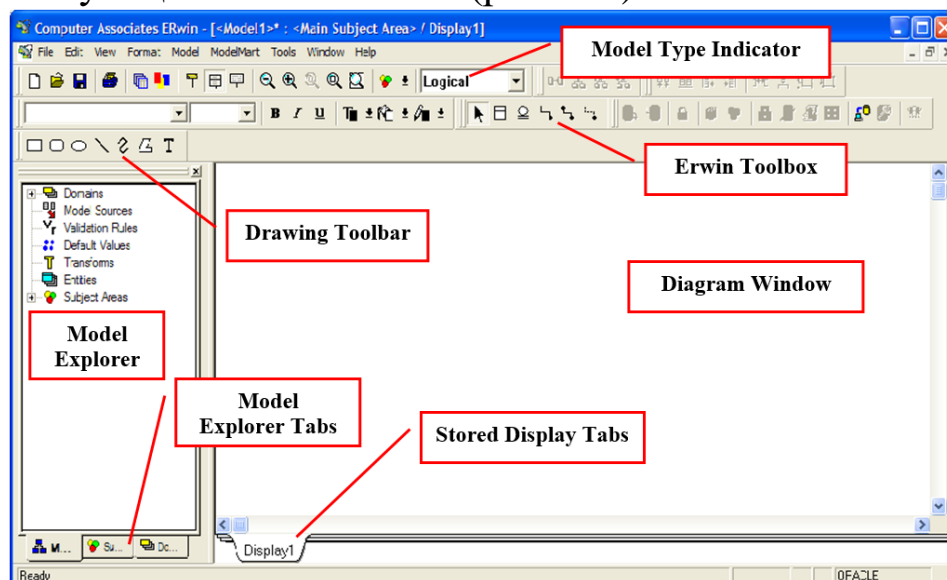


Рисунок 2.2 – Стандартное окно диаграммы

Diagram Window – окно диаграммы, отображающее графическое представление модели данных.

Model Explorer – обеспечивает иерархию базового текстового представления модели данных, которая отображена в окне: обеспечивает структурное представление модели данных и содержание. В Model Explorer можно увидеть другие виды моделей.

Erwin Toolbox – панель инструментов. Каждому уровню отображения модели соответствует своя панель инструментов.

Stored Display – графическое представление области, содержащее координаты сущностей и связей и графические режимы демонстрации изображения. Каждая модель в ERwin имеет загруженный дисплей, который называется Display1. Его можно переименовывать. Можно создавать другие дисплеи для модификации представления модели данных.

Model Type Indicator – указатель на тип модели, идентифицирующий текущий тип модели.

Модели подразделяются на:

- Logical – логическая модель, которая содержит сущности, атрибуты и ключевые группы;

- Physical – физическая модель (БД), которая содержит столбцы таблиц и типы данных;
- Logical/Physical – стандартная модель ERwin, включающая логическую и физическую модели.

Он может быть переключен между логической и физической моделью только для логико-физической модели.

Каждому уровню отображения модели соответствует своя палитра инструментов.

Создание новой модели

При создании новой модели возникает диалоговое окно Create Model – Select Template, в котором следует указать тип новой модели (рис. 2.3).

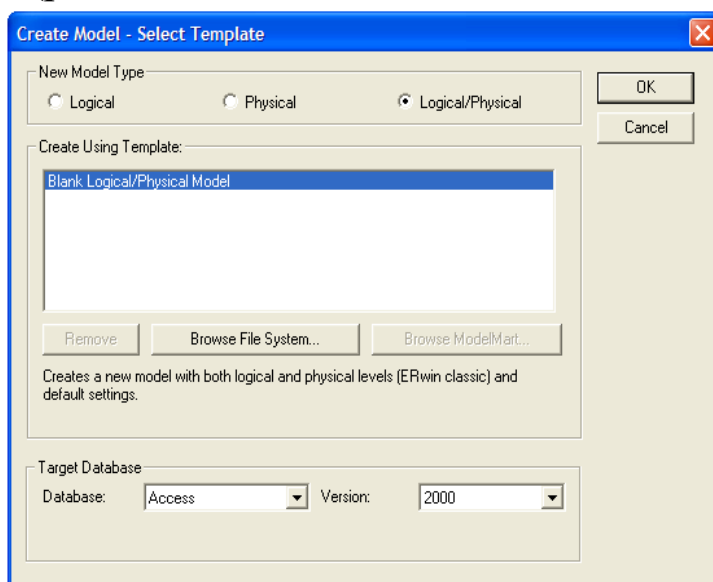


Рисунок 2.3 – Диалог создания новой модели

Как было указано выше, ERwin поддерживает три модели – Logical, Physical и Logical/Physical. В зависимости от типа выбранной модели будет меняться вид данного диалогового окна. Состав палитры инструментов также изменяется автоматически, когда происходит переключение с одной модели на другую.

Установка цвета и шрифта

Установить шрифт и цвет объектов в ERwin можно несколькими способами.

Во-первых, для установки можно использовать панель инструментов выбора шрифтов и цветов Font & Color.

Во-вторых, для редактирования шрифта и цвета конкретного объекта следует, щелкнув правой клавишей мыши по сущности или связи и выбрав из всплывающего меню пункт Object Font/Color, вызвать диалог Font/Color, в котором можно выбрать шрифт и установить его размер, стиль и цвет, установить цвет заливки.

Третьим способом изменения шрифта и цвета для всех объектов модели или для какой-либо отдельной категории объектов является диалог Default Font & Color (пункт меню Format/ Default Font&Color). Каждая вкладка на диалоге позволяет редактировать шрифт и цвет для определенной категории объектов (рис. 2.4):

General – все объекты модели – цвет фона диаграммы;

Entities (Tables) – наименования сущностей и таблиц; заливка сущностей и таблиц; линии, которыми прорисовываются сущности и таблицы;

Attributes (Columns) – атрибуты и колонки, включая отдельные настройки для атрибутов и колонок внешних ключей;

Relationships – связи, включая имя и обозначение мощности;

Subtypes – иерархия категорий, включая дискриминатор категории;

Drawing Object Text – текстовые блоки;

Drawing Object Colors – текстовые блоки.

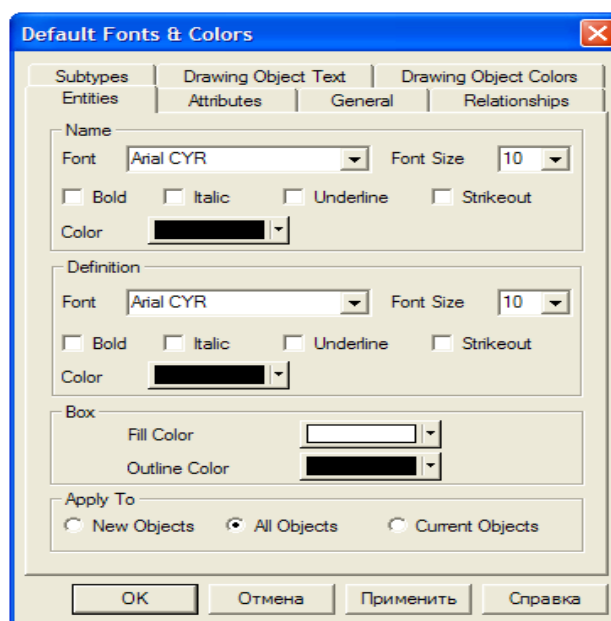


Рисунок 2.4 – Диалог Default Font & Color

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №2

1. Изучите основные функции и возможности выбранного средства моделирования бизнес-процессов.
2. Определите основные этапы и шаги процесса разработки информационного ресурса в рамках выбранного проекта.
3. Создайте модель бизнес-процесса, отражающей этапы разработки информационного ресурса с учетом его особенностей и требований.
4. Идентификация ключевых участников и ролей в процессе разработки информационного ресурса и их взаимосвязей.
5. Анализ проектируемого процесса на предмет выявления возможных улучшений, оптимизации и автоматизации этапов.
6. Документирование созданной модели бизнес-процесса в соответствии с требованиями выбранного средства моделирования.

РЕКОМЕНДАЦИИ К ВЫПОЛНЕНИЮ ЗАДАНИЯ

- При создании модели бизнес-процесса обратите внимание на четкость и понятность ее структуры.
- В процессе анализа модели уделите внимание выявлению возможных узких мест и проблемных зон для дальнейшего улучшения процесса.
- Не забывайте документировать все этапы работы и принятые решения для последующей передачи знаний и оценки результатов.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая цель работы по моделированию процесса разработки информационного ресурса с помощью BPwin или AllFusion Process Modeler?
2. Какие основные требования предъявляются к методикам и инструментальным средствам при создании информационных систем?
3. Какие основные функции предоставляют CASE-средства AllFusion Modeling Suite 4.1 и AllFusion Process Modeler (BPwin)?

4. Какие методологии поддерживает CASE-средство BPwin, и для чего они предназначены?
5. Какую роль выполняет функциональная модель в процессе моделирования бизнес-процессов с помощью BPwin или AllFusion Process Modeler?
6. Какие возможности предоставляет CASE-средство AllFusion ERwin Data Modeler (ERwin) для моделирования данных?
7. Какие этапы реализации проекта информационной системы могут быть автоматизированы с помощью выбранных инструментальных средств?
8. Какие преимущества предоставляет CASE-средство при моделировании бизнес-процессов и создании информационных ресурсов?
9. Каким образом выбранные инструменты удовлетворяют требованиям гибкости к изменяющимся требованиям информационных систем?
10. Какие основные этапы включает процесс моделирования процесса разработки информационного ресурса с использованием BPwin или AllFusion Process Modeler?

ПРАКТИЧЕСКАЯ РАБОТА №3. ПОСТРОЕНИЕ СЕТЕВОГО ГРАФИКА РАЗРАБОТКИ ВЕБ-ПРОЕКТА В MS PROJECT

Цель работы: овладеть навыками создания и управления сетевым графиком разработки веб-проекта с использованием программного обеспечения MS Project.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Структурное планирование включает в себя несколько этапов:

- 1) разбиение проекта на совокупность отдельных работ, выполнение которых необходимо для реализации проекта;
- 2) построение сетевого графика, описывающего последовательность выполнения работ;
- 3) оценка временных характеристик работ и анализ сетевого графика.

Основную роль на этапе структурного планирования играет сетевой график.

Сетевой график – это ориентированный граф, в котором вершинами обозначены работы проекта, а дугами – временные взаимосвязи работ.

Сетевой график должен удовлетворять следующим свойствам.

1. Каждой работе соответствует одна и только одна вершина. Ни одна работа не может быть представлена на сетевом графике дважды. Однако любую работу можно разбить на несколько отдельных работ, каждой из которых будет соответствовать отдельная вершина графика.
2. Ни одна работа не может быть начата до того, как закончатся все непосредственно предшествующие ей работы. То есть если в некоторую вершину входят дуги, то работа может начаться только после окончания всех работ, из которых выходят эти дуги.
3. Ни одна работа, которая непосредственно следует за некоторой работой, не может начаться до момента ее окончания. Другими словами, если из работы выходит

несколько дуг, то ни одна из работ, в которые входят эти дуги, не может начаться до окончания этой работы.

4. Начало и конец проекта обозначены работами с нулевой продолжительностью. Такие работы называются вехами и обозначают начало или конец наиболее важных этапов проекта.

Пример. В качестве примера рассмотрим проект "Разработка программного комплекса". Предположим, что проект состоит из работ, характеристики которых приведены в табл. 3.1.

Таблица 3.1. Работы проекта

| Номер работы | Название работы | Длительность |
|--------------|--------------------------------------|--------------|
| 1 | Начало реализации проекта | 0 |
| 2 | Постановка задачи | 10 |
| 3 | Разработка интерфейса | 5 |
| 4 | Разработка модулей обработки данных | 7 |
| 5 | Разработка структуры базы данных | 6 |
| 6 | Заполнение базы данных | 8 |
| 7 | Отладка программного комплекса | 5 |
| 8 | Тестирование и исправление ошибок | 10 |
| 9 | Составление программной документации | 5 |
| 10 | Завершение проекта | 0 |

Сетевой график для данного проекта изображен на рис. 3.1. На нем вершины, соответствующие обычным работам, обведены тонкой линией, а толстой линией обведены вехи проекта.

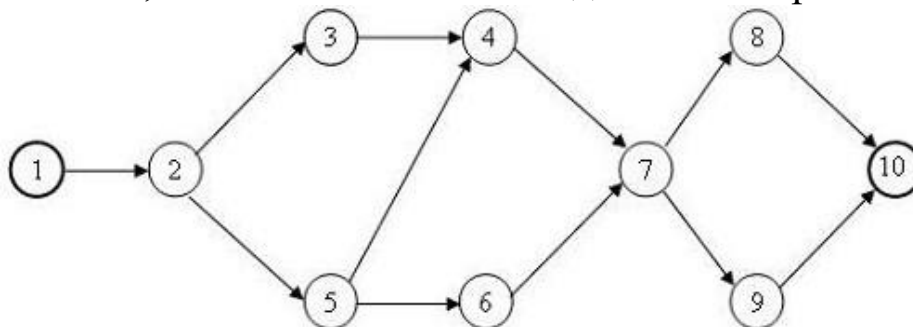


Рисунок 3.1. Сетевой график проекта

Сетевой график позволяет по заданным значениям длительностей работ найти критические работы проекта и его критический путь.

Критической называется такая работа, для которой задержка ее начала приведет к задержке срока окончания проекта в целом. Такие работы не имеют запаса времени. Некритические работы имеют некоторый запас времени, и в пределах этого запаса их начало может быть задержано.

Критический путь – это путь от начальной к конечной вершине сетевого графика, проходящий только через критические работы. Суммарная длительность работ критического пути определяет минимальное время реализации проекта.

Нахождение критического пути сводится к нахождению критических работ и выполняется в два этапа.

1. Вычисление **раннего времени начала** каждой работы проекта. Эта величина показывает время, раньше которого работа не может быть начата.
2. Вычисление **позднего времени начала** каждой работы проекта. Эта величина показывает время, позже которого работа не может быть начата без увеличения продолжительности всего проекта.

Схема сети — это графический способ просмотра задач, зависимостей и критического пути проекта. Поля (или узлы) представляют задачи, а зависимости отображаются в виде линий, соединяющих эти поля. После переключения представлений можно добавить условные обозначения, настроить способ отображения полей и распечатать схему сети.

Чтобы найти представление Схема сети, выберите **Пункт Просмотреть > сетевая схема** (рисунок 3.2).

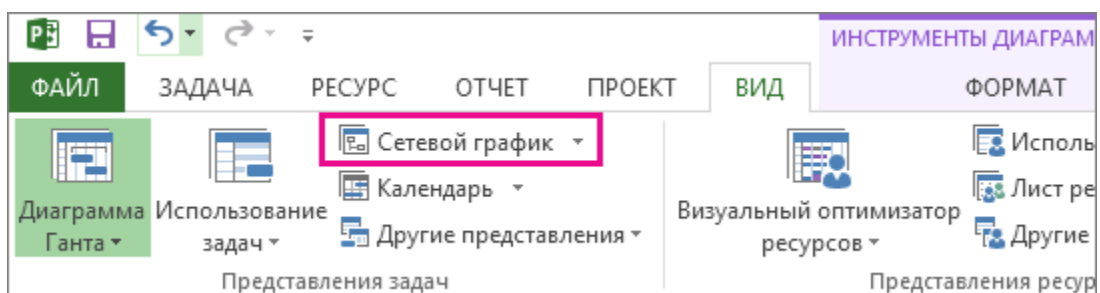


Рисунок 3.2 – Схема сети

Добавление условных обозначений

1. Выберите **Файл > Печать > настройка страницы**.

2. На вкладке **Условные обозначения** определите, как должна выглядеть легенда, на каких страницах она должна отображаться, а затем укажите нужные метки.
3. Нажмите кнопку **ОК**.

Автоматическое изменение способа выкладки полей

1. Выберите **Просмотреть > схему сети**.
2. Выберите **Формат > макет** (рис. 3.3).

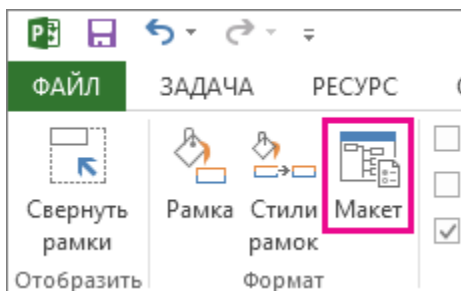


Рисунок 3.3 – Изменение способа выкладки полей

3. В разделе **Макет поля** выберите оптимальные варианты расположения, выравнивания, интервала, высоты и ширины. Чтобы выровнять поля, выберите **Фиксированный** в полях **Высота** и **Ширина**.

Помните, что сгруппированные задачи размещаются автоматически. Если вы хотите изменить группирование, вам потребуется отменить группирование.

Вручную измените способ выкладки полей

Если вам все еще не нравится расположение полей, нажмите кнопку **Формат > макет**, выберите **Разрешить ручное позиционирование** (рис. 3.4), нажмите кнопку **ОК**, а затем перетащите поля в нужное место.

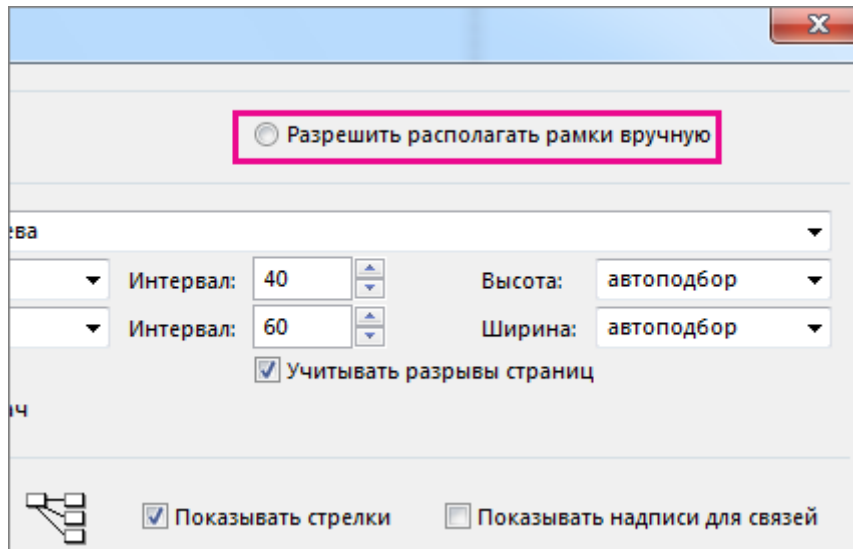




Рисунок 3.4 – Изменение способа выкладки полей

Если вы вручную изменяете положение задачи, можно изменить макет любых связанных задач или связанных с ней подзадач, щелкнув ее правой кнопкой мыши и выбрав пункт Макет связанные задачи сейчас.

Изменение стиля линии между полями

Если у вас много задач, связанных с задачами-предшественниками или последователями, то связи между полями может быть очень трудно отслеживать. Попробуйте изменить стиль линий, а затем упорядочить их таким образом, чтобы их было проще увидеть.

1. Выберите **Просмотреть > схему сети**.
2. Выберите **Формат > макет**.
3. В разделе **Стиль связи** выберите **Прямолинейный** или **Прямой**.

Прямолинейные ссылки выглядят так, как это , а прямые ссылки — .

4. Выберите **Показать стрелки**, чтобы добавить стрелки, указывающие на задачи предшественника и преемника. Выберите **Показать метки ссылок**, чтобы добавить зависимость и время потенциального клиента или задержки в линию ссылки.

Выберите тип сведений о задачах для отображения

Если все выглядит перегруженным (или вы начинаете сталкиваться с перегрузкой информации), попробуйте изменить сведения о задаче в каждом поле, чтобы вы видели только самое важное (рис. 3.5).

1. Выберите **Просмотреть > схему сети**.
2. Выберите **Формат > стили полей**.
3. В списке **Параметры стиля** выберите задачу, которую нужно изменить.

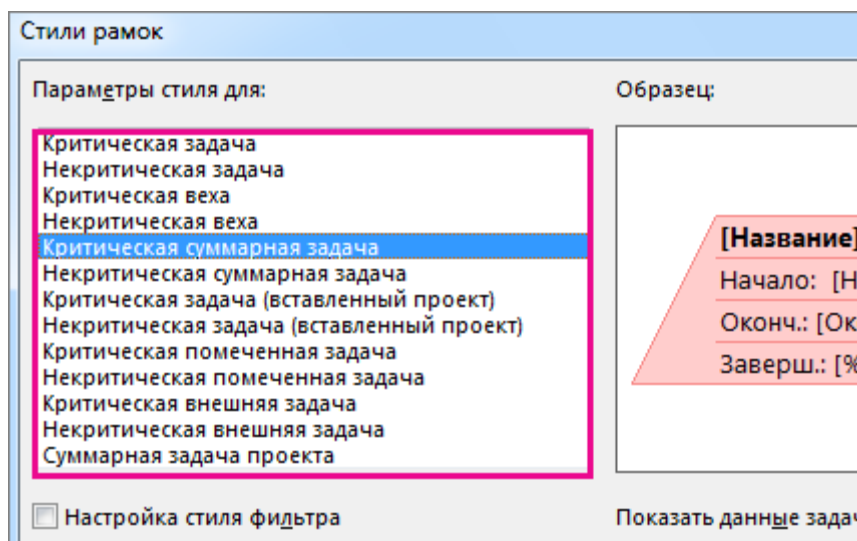


Рисунок 3.5 – Изменение сведений о задаче

4. В разделе **Граница** выберите фигуру, цвет, ширину и сетку, чтобы создать нужный вид.
5. Выберите имя в разделе **Шаблон данных**, чтобы применить изменения к существующему шаблону. Чтобы создать шаблон, который будет использовать внесенные изменения, выберите **Другие шаблоны**, а затем выберите **Создать** (чтобы создать новый шаблон), **Копировать** (чтобы создать новый шаблон на основе существующего), **Изменить** (чтобы изменить шаблон) или **Импорт** (чтобы импортировать шаблон из другого проекта).
6. Нажмите кнопку **ОК**.

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №3

1. Изучите основные функции MS Project, ознакомьтесь с его интерфейсом и возможностями по построению сетевых графиков.
2. Сформулируйте структуру разработки веб-проекта, определив основные этапы и задачи проекта (например, анализ требований, дизайн, разработка, тестирование, запуск).
3. Создайте новый проект в MS Project и введите все задачи, разбив их по этапам разработки.
4. Установите логические связи между задачами, определив зависимости (начало-конец, начало-начало, конец-конец) для правильного распределения времени выполнения задач.
5. Определите критический путь проекта и оцените его длительность для выявления наиболее важных задач.
6. Распределите ресурсы между задачами, учитывая их доступность и нагрузку.
7. Проанализируйте сетевой график на предмет возможных рисков и узких мест, а также определите стратегии управления ими.
8. Разработайте план управления рисками, включая меры по снижению влияния рисков на успешное завершение проекта.
9. Подготовьте отчет, содержащий созданный сетевой график, анализ критического пути, план управления рисками и выводы о результатах выполненной работы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какова цель работы по построению сетевого графика разработки веб-проекта с использованием MS Project?
2. Какие этапы включает в себя структурное планирование проекта?
3. Что представляет собой сетевой график в контексте проектного управления?
4. Какие основные свойства должен удовлетворять сетевой график?
5. Какие вершины и дуги соответствуют на сетевом графике?
6. Какие работы могут быть представлены на сетевом графике?

7. Какие условия должны выполняться для начала выполнения определенной работы согласно сетевому графику?
8. Что означает термин "веха" в контексте сетевого графика?
9. Каким образом определяются начало и конец проекта на сетевом графике?
10. Какие инструменты и программное обеспечение используются для построения и управления сетевым графиком, как в данной работе?

ПРАКТИЧЕСКАЯ РАБОТА №4. СОЗДАНИЕ ПРОСТЕЙШИХ ПРОГРАММ НА JS: ВЫВОД РЕЗУЛЬТАТОВ И ВВОД ДАННЫХ, ПЕРЕМЕННЫЕ, КОНСТАНТЫ, ОПЕРАТОРЫ JAVASCRIPT.

Цель работы: овладеть основами программирования на языке JavaScript, освоить базовые концепции, такие как работа с переменными, константами и операторами, а также научиться осуществлять ввод и вывод данных.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Основы JavaScript

Переменные и константы

Для хранения данных в программе используются переменные и константы. Переменные предназначены для хранения каких-нибудь временных данных или таких данных, которые в процессе работы могут менять свое значение. Константы, наоборот, предназначены для хранения неизменных данных, значение которых должно оставаться постоянным в течение всей программы.

Объявление переменных

Для создания переменных применяются операторы `var` и `let`. Например, объявим переменную `username`:

```
1 var username;
```

Объявление переменной представляет отдельную инструкцию, поэтому завершается точкой с запятой.

Аналогичное определение переменной с помощью оператора `let`:

```
1 let username;
```

Каждая переменная имеет имя. Имя представляет собой произвольный набор алфавитно-цифровых символов, знака подчеркивания (`_`) или знака доллара (`$`), причем названия не должны начинаться с цифровых символов. То есть мы можем использовать в названии буквы, цифры, подчеркивание и символ `$`. Однако все остальные символы запрещены.

Например, правильные названия переменных:

```
1 $commision
2 someVariable
3 product_Store
4 income2
5 myIncome_from_deposit
```

Следующие названия являются некорректными и не могут использоваться:

```
1 222lol
2 @someVariable
3 my%percent
```

Также нельзя давать переменным такие имена, которые совпадают с зарезервированными ключевыми словами. В JavaScript не так много ключевых слов, поэтому данное правило не сложно соблюдать. Например, следующее название будет некорректным, так как `for` - ключевое слово в JavaScript.

Также не рекомендуется объявлять переменные с именем, которые аналогичны уже имеющимся глобальным переменным. Например, для вывода на консоль используется метод `console.log()`. Здесь `console` — это имя глобального объекта, и мы можем определить переменную с этим именем.

```
1 let console; // определяем переменную console
2 console.log("Hello"); // ! Ошибка
```

Определение переменной `console` приведет к тому, что она переопределит одноименный глобальный объект `console`, и при вызове метода `console.log()` мы столкнемся с ошибкой.

При названии переменных надо учитывать, что JavaScript является регистрозависимым языком, то есть в следующем коде объявлены две разные переменные:

```
1 let username;
2 let userName;
```

Через запятую можно определить сразу несколько переменных:

```
1 var username, age, height;
2 let a, b, c;
```

Присвоение переменной значения

После определения переменной ей можно присвоить какое-либо значение. Для этого применяется оператор присваивания (=):

```
1  var username;
2  username = "Tom";
```

То есть в данном случае переменная username будет хранить строку "Tom". После присвоения переменной значения мы можем что-то сделать с этим значением, например, выведем его на консоль:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="utf-8" />
5    <title>METANIT.COM</title>
6  </head>
7  <body>
8    <script>
9      let username;      // Определение переменной username
10     username = "Tom";   // Присвоение переменной значения
11     console.log(username); // Вывод значения переменной username на консоль
12   </script>
13 </body>
14 </html>
```

Можно сразу присвоить переменной значение при ее определении:

```
1  var username = "Tom";
2  let usage = 37;
```

Процесс присвоения переменной начального значения называется инициализацией. Можно обратить внимание, что одна переменная определена с помощью let, а другая - с помощью var. Конкретно в данном случае это не имеет значения, и мы могли бы определить обе переменных либо с помощью let, либо с помощью var.

Можно сразу инициализировать несколько переменных:

```
1  let name1 = "Tom", name2 = "Bob", name3 = "Sam";
2  console.log(name1); // Tom
3  console.log(name2); // Bob
4  console.log(name3); // Sam
```

JavaScript позволяет некоторые вольности при определении переменной. Так, мы можем НЕ использовать ключевые слова let или var

```
1  <!DOCTYPE html>
```

```

2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <title>METANIT.COM</title>
6 </head>
7 <body>
8   <script>
9     username = "Tom"; // ошибки нет, норм
10    console.log(username); // Tom
11  </script>
12 </body>
13 </html>

```

Хотя мы можем так делать, но это не самый рекомендуемый подход. В одной из следующих статей мы увидим, с какими проблемами мы можем столкнуться при подобном подходе.

Изменение переменных

Отличительной чертой переменных является то, что мы можем изменить их значение:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <title>METANIT.COM</title>
6 </head>
7 <body>
8   <script>
9     let username = "Tom";
10    console.log("username до изменения:", username);
11    username = "Bob";
12    console.log("username после изменения:", username);
13  </script>
14 </body>
15 </html>

```

Константы

С помощью ключевого слова `const` можно определить константу, которая, как и переменная, хранит значение, однако это значение не может быть изменено.

```
1 const username = "Tom";
```

Если мы попробуем изменить ее значение, то мы столкнемся с ошибкой:

```
1 const username = "Tom";
```

```
2 username = "Bob"; // ошибка, username - константа, поэтому мы не можем изменить ее значение
```

Также стоит отметить, что поскольку мы не можем изменить значение константы, то она должна быть инициализирована, то есть при ее определении мы должны предоставить ей начальное значение. Если мы этого не сделаем, то опять же мы столкнемся с ошибкой:

```
1 const username; // ошибка, username не инициализирована
```

Если вы уверены, что значение в процессе программы не будет изменяться, тогда это значение определяется в виде константы. Если неизвестно, будет ли значение меняться или нет, то рекомендуется определить значение как константу. В случае если далее потребуется его изменить, то можно изменить определение значения с `const` на `var/let`.

Типы данных

Все используемые данные в javascript имеют определенный тип. В JavaScript имеется восемь типов данных:

- **String**: представляет строку
- **Number**: представляет числовое значение
- **BigInt**: предназначен для представления очень больших целых чисел
- **Boolean**: представляет логическое значение **true** или **false**
- **Undefined**: представляет одно специальное значение - **undefined** и указывает, что значение не установлено
- **Null**: представляет одно специальное значение - **null** и указывает на отсутствие значения
- **Symbol**: представляет уникальное значение, которое часто применяется для обращения к свойствам сложных объектов
- **Object**: представляет комплексный объект

Ввод и вывод данных в JavaScript

Вывод текста, данных на экран в JavaScript осуществляется при помощи двух операторов: `Alert` и `Write`.

Оператор Alert в JavaScript

Этот способ вывода текста (информации) в JavaScript характерен тем, что на экране браузера появляется небольшое

окно с сообщением – его еще называют диалоговой панелью. Характерные черты панели – наличие кнопки Ok и текстовой информации.

Давайте рассмотрим пример вывода сообщения в JavaScript: `Alert('Привет! Как дела?')` // в диалоговой панели появится соответствующая надпись

Вот так просто с помощью оператора `Alert` в JavaScript мы можем производить вывод данных на экран (текста, картинки, сообщения).

Данный способ вывода сообщений удобен в том случае, когда текстовое сообщение небольшое и не нуждающееся в форматировании. В противном случае стоит использовать оператор `write`.

Оператор Write в JavaScript

Данный метод предоставляется объектом `document`. Поэтому оператор вывода сообщений будет выглядеть так: **`document.write('Текстовая информация')`**. Текст будет отображаться не в диалоговой панели, а в окне браузера.

В отличие от первого способа мы можем форматировать наш документ при помощи тегов HTML языка. Строка с сообщением будет выглядеть в этом случае так, как если бы она являлась частью HTML странички.

Пример форматирования: `document.write('<h1><i>Вот так форматируется текст</i></h1>')`

Если сообщение большое, то его можно разделять на несколько подстрок при помощи символа `+`. `document.write('Разделяем сообщение ' + ' символом +')`

Ввод данных в JavaScript

Для ввода данных в JavaScript можно использовать также два способа: вызов `confirm` или `prompt`. Оба метода, как и `alert`, работают с диалоговой панелью, но имеют различные задачи. Первый требует от пользователя лишь выбора одного из двух вариантов, а второй – в заполнении формы.

Оператор confirm в JavaScript

При использовании данного оператора пользователь увидит на экране диалоговую панель, содержащую какое-то сообщение,

а чуть ниже – две кнопки – Ok и Cancel. Такой способ необходим в том случае, если программа нуждается в действии от пользователя – подтверждения или опровержения какой-то информации.

Давайте рассмотрим пример использования оператора confirm:

```
if (confirm('Вы готовы перейти по ссылке?'))
document.write('Переходим...')
else
document.write('Переход по ссылке отменен')
```

Скрипт работает так:

- Выводит пользователю сообщение «Вы готовы перейти по ссылке?» и ждет его действий.
- Если пользователь нажимает да (Ok), то программа выводит на экран сообщение «Переходим...».
- Если пользователь нажимает отмена (cancel), то на экран выводится «Переход по ссылке отменен».

ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ № 4

1. Создайте программу, которая выводит в консоль приветственное сообщение.
2. Напишите программу, которая запрашивает у пользователя его имя и выводит сообщение с приветствием, используя введенное имя.
3. Реализуйте программу для вычисления площади прямоугольника. Пользователь должен ввести длину и ширину прямоугольника, а программа должна вывести результат.
4. Напишите программу для конвертации температуры из градусов Цельсия в градусы Фаренгейта. Пользователь вводит температуру в градусах Цельсия, а программа выводит результат в градусах Фаренгейта.
5. Создайте программу для вычисления среднего арифметического двух чисел, введенных пользователем.
6. Напишите программу, которая определяет, является ли введенное пользователем число четным или нечетным, и выводит соответствующее сообщение.

7. Реализуйте программу для вычисления площади круга. Пользователь вводит радиус круга, а программа выводит площадь.
8. Создайте программу для проверки является ли введенное пользователем число положительным, отрицательным или нулем, и выводите соответствующее сообщение.
9. Напишите программу, которая проверяет, является ли введенный пользователем год високосным, и выводит сообщение с результатом.
10. Реализуйте программу для вычисления суммы первых N натуральных чисел, где N вводится пользователем.

КОНТРОЛЬНАЯ РАБОТА

1. Какие базовые концепции программирования изучаются в данной работе?
2. Что такое переменные в контексте JavaScript, и для чего они используются?
3. В чем разница между переменными и константами в JavaScript?
4. Как объявляются переменные и константы в JavaScript?
5. Какие операторы доступны в JavaScript, и для чего они предназначены?
6. Как осуществляется ввод данных в программу на JavaScript?
7. Какие методы используются для вывода данных из программы на JavaScript?
8. Какие инструменты можно использовать для тестирования JavaScript-программ, разработанных в рамках данной работы?
9. Какие практические примеры можно реализовать, чтобы продемонстрировать освоенные концепции переменных, констант и операторов в JavaScript?

ПРАКТИЧЕСКАЯ РАБОТА №5. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ УСЛОВНЫХ ОПЕРАТОРОВ И ЦИКЛОВ.

Цель работы: овладеть основными конструкциями управления потоком выполнения программы в JavaScript, такими как условные операторы (if-else) и циклы (for, while), а также научиться применять их для решения различных задач.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Условные операторы позволяют пропустить или выполнить другие операторы в зависимости от значения указанного выражения. Эти операторы являются точками принятия решений в программе, и иногда их также называют операторами «ветвления».

Оператор if/else

Оператор if — это базовый управляющий оператор, позволяющий интерпретатору JavaScript принимать решения или, точнее, выполнять операторы в зависимости от условий. Оператор if имеет две формы.

Первая:

```
if (выражение)
    оператор
```

В этой форме сначала вычисляется выражение. Если полученный результат является истинным, то оператор выполняется. Если выражение возвращает ложное значение, то оператор не выполняется. Например:

```
if (username == null) // Если переменная username равна null или undefined
    username = "Alex"; // определить ее
```

Обратите внимание, что скобки вокруг условного выражения являются обязательной частью синтаксиса оператора if.

Вторая форма оператора if вводит конструкцию else, выполняемую в тех случаях, когда выражение возвращает ложное значение. Ее синтаксис:

```
if (выражение)
    оператор1
else
    оператор2
```

Эта форма выполняет оператор1, если выражение возвращает истинное значение, и оператор2, если выражение возвращает ложное значение. Например:

```
if (n == 1)
    console.log("Получено 1 новое сообщение.");
else
    console.log("Получено " + n + " новых сообщений.");
```

Оператор else if

Оператор **if/else** вычисляет значение выражения и выполняет тот или иной фрагмент программного кода, в зависимости от результата.

Возможный способ сделать это состоит в применении оператора **else if**. Формально он не является самостоятельным оператором JavaScript; это лишь распространенный стиль программирования, заключающийся в применении повторяющегося оператора **if/else**:

```
if (n == 1) {
    // Выполнить блок 1
}
else if (n == 2) {
    // Выполнить блок 2
}
else if (n == 3) {
    // Выполнить блок 3
}
else {
    // Если ни один из предыдущих операторов else не был выполнен, выполнить
    блок 4
}
```

Циклы

Циклы позволяют в зависимости от определенных условий выполнять некоторое действие множество раз. В JavaScript имеются следующие виды циклов:

- **for**
- **for..in**
- **for..of**
- **while**
- **do..while**

Цикл for

Цикл **for** имеет следующее формальное определение:

```
1 for ([инициализация счетчика]; [условие]; [изменение счетчика]){
```

```

2
3    // действия
4 }

```

Например, используем цикл for для перебора чисел от 0 до 4:

```

1  for(let i = 0; i<5; i++){
2
3      console.log(i);
4  }
5  console.log("Конец работы");

```

Первая часть объявления цикла - `let i = 0` - создает и инициализирует счетчик - переменную `i`. И перед выполнением цикла ее значение будет равно 0. Это то же самое, что и объявление переменной.

Вторая часть - условие, при котором будет выполняться цикл: `i<5`. В данном случае цикл будет выполняться, пока значение `i` не достигнет 5.

Третья часть - `i++` - приращение счетчика на единицу.

То есть при запуске переменная `i` равна 0. Это значение отвечает условию `i<5`, поэтому будет выполняться блок цикла, а именно строка кода

```
1  console.log(i);
```

После выполнения блока цикла выполняется третья часть объявления цикла - приращение счетчика. То есть переменная `i` становится равной 1. Это значение также отвечает условию, поэтому блок цикла снова выполняется. Таким образом, блок цикла сработает 5 раз, пока значение `i` не станет равным 5. Это значение НЕ отвечает условию, поэтому произойдет выход из цикла. И управление программой перейдет к инструкциям, которые идут после блока цикла. Консольный вывод программы:

```

0
1
2
3
4
Конец работы

```

Каждое отдельное повторение цикла называется итерацией. Таким образом, в данном случае сработают 5 итераций.

При этом необязательно увеличивать счетчик на единицу, можно производить с ним другие действия, например, уменьшать на единицу:

```

1  for (let i = 10; i > 5; i--) {
2
3      console.log(i);
4  }

```

В данном случае на консоль выводится числа от 10 до 6.

Или увеличим счетчик на 2:

```

1  for (let i = 0; i < 10; i+=2) {
2
3      console.log(i);
4  }

```

Здесь выводятся на консоль все четные числа от 0 до 8

При этом можно опускать различные части объявления цикла:

```

1  let i = 0;
2  for (; i < 60;) {
3
4      console.log(i);
5      i = i + 10;
6  }

```

В данном случае переменная *i* определена вне цикла. В самом объявлении цикла есть только условие, остальные две части отсутствуют. Изменение переменной происходит в самом блоке цикла: оно увеличивается на 10. В итоге на консоль будут выведены числа 0, 10, 20, 30, 40, 50.

Счетчик удобно использовать как индекс элементов массива и таким образом перебирать массив:

```

1  const people = ["Tom", "Sam", "Bob"];
2  for (let i=0; i < 3; i++) {
3
4      console.log(people[i]);
5  }

```

Консольный вывод браузера:

```

Tom
Sam
Bob

```

Применение нескольких счетчиков в цикле

При необходимости можно использовать несколько счетчиков:

```

1  for (let i = 1, j=1; i < 5, j < 4; i++, j++) {
2
3      console.log (i + j);
4  }
5  // 1 итерация: i=1, j=1; i + j = 2
6  // 2 итерация: i=2, j=2; i + j = 4

```

7 // 3 итерация: i=3, j=3; i + j = 6

Здесь теперь используются два счетчика и два условия. Рассмотрим пошагово, что здесь происходит:

- 1) Первая итерация. Начальные значения переменных i и j:
1 i=1, j=1;
- 2) Для каждой переменной установлены свои условия. И вначале начальные значения переменных соответствуют этим условиям:
1 i < 5, j < 4;
- 3) В блоке цикла выводится сумма этих переменных. И дальше значения обоих переменных увеличиваются на единицу. Они становятся равны
1 i=2, j=2;
- 4) Эти значения также соответствуют условиям, поэтому выполняется вторая итерация
- 5) Вторая итерация. Значения переменных i и j:
1 i=2, j=2;
- 6) После выполнения блока цикла значения обоих переменных увеличиваются на единицу. Они становятся равны
1 i=3, j=3;
- 7) Эти значения также соответствуют условиям, поэтому выполняется третья итерация
- 8) Третья итерация. Значения переменных i и j:
1 i=3, j=3;
- 9) После выполнения блока цикла значения обоих переменных увеличиваются на единицу. Они становятся равны
1 i=4, j=4;
- 10) Значение переменной i соответствует условию i < 5, однако значение переменной j (4) НЕ соответствует условию j < 4. Поэтому происходит выход из цикла. Его работа завершена.

Выполнение действий в объявлении цикла

Стоит отметить, что третья часть цикла, где обычно происходит изменение счетчика в реальности представляет произвольное действие, которое выполняется после завершения цикла. Так, мы можем написать следующим образом:

```
1 for (let i = 0; i < 5; console.log(i++));
2 console.log ("Конец работы");
```

Здесь не определено блока цикла, а сами действия цикла определены в третьей части заголовка цикла - console.log(i++)

Аналогично в первой части определения цикла - инициализации мы можем выполнять некоторые действия, а не обязательно только объявление счетчика:

```
1 let i=0;
2 for(console.log("Init"); i < 5; i++) {
3
4     console.log(i);
5 }
```

Здесь определение счетчика вынесено вне цикла, а в инициализационной части цикла на консоль выводится строка. Вывод браузера:

```
Init
0
1
2
3
4
```

Вложенные циклы

Одни циклы могут внутри себя содержать другие:

```
1 for (let i=1; i <= 5; i++) {
2
3     for (let j = 1; j <=5; j++) {
4         console.log (i * j);
5     }
6 }
```

Здесь один цикл включается в себя другой. Во внешнем цикле определяется переменная *i*. Вначале она равна 1 и это значение соответствует условию цикла (*i* <=5), поэтому будет выполняться блок цикла, который содержит внутренний цикл.

Во внутреннем цикле определяется переменная-счетчик *j*, которая изначально равна 1, и потом внутренний цикл выполняет 5 итераций, пока переменная *j* не станет равна 5.

После того, как блок внешнего цикла завершен, переменная *i* увеличивается на 1 и становится равной 2, что опять же соответствует условию. И снова выполняется блок внешнего цикла. В этом блоке снова выполняются пять итераций внутреннего цикла. И так далее. В итоге внутренний цикл будет выполняться 25 раз.

Используя вложенные циклы и несколько счетчиков, можно перебирать многомерные массивы:

```
1 const people = [["Tom", 39], ["Sam", 28], ["Bob", 42]];
2 for(let i=0; i < 3; i++){ // перебираем двухмерный массив
```

```

3
4   for(let j=0; j < 2; j++){ // перебираем вложенные массивы
5
6       console.log(people[i][j]);
7   }
8   console.log("====="); // для разделения элементов
9 }

```

Здесь массив `people` представляет двухмерный массив из 3-х элементов, где каждый элемент представляет, в свою очередь, подмассив из 2-х элементов - условно имени и возраста пользователя. Во внешнем цикле определяем счетчик `i` для прохода по всем подмассивам в двухмерном массиве `people`, а во внутреннем цикле определяем счетчик `j` для прохода по всем элементам каждого подмассива. Консольный вывод:

```

Tom
39
=====
Sam
28
=====
Bob
42
=====

```

Цикл **while**

Цикл `while` выполняется до тех пор, пока некоторое условие истинно. Его формальное определение:

```

1   while(условие){
2
3       // действия
4   }

```

Опять же выведем с помощью `while` числа от 1 до 5:

```

1   let i = 1;
2   while (i <=5) {
3
4       console.log(i);
5       i++;
6   }

```

Цикл `while` здесь будет выполняться, пока значение `i` не станет равным 6.

Цикл **do...while**

В цикле `do` сначала выполняется код цикла, а потом происходит проверка условия в инструкции `while`. И пока это условие истинно, цикл повторяется. Например:

```

1   let i = 1;

```

```

2  do {
3    console.log(i);
4    i++;
5  } while (i <= 5)

```

Здесь код цикла сработает 5 раз, пока *i* не станет равным 5. При этом цикл `do` гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции `while` не будет истинно.

Операторы `continue` и `break`

Иногда бывает необходимо выйти из цикла до его завершения. В этом случае мы можем воспользоваться оператором `break`:

```

1  for (let i=1; i <= 6; i++) {
2
3    if(i===4) break;
4    console.log(i);
5  }
6  console.log ("Конец работы");

```

Данный цикл увеличивает переменную *i* с 1 до 6 включая, то есть согласно условию цикла блок цикла должен выполняться 6 раз, то есть произвести 6 итераций. Однако поскольку в блоке цикла происходит проверка `if(i===4) break;` то, когда значение переменной *i* достигнет 4, то данное условие прервет выполнение цикла с помощью оператора `break`. И цикл завершит работу.

```

1
2
3
Конец работы

```

Если нам надо просто пропустить итерацию, но не выходить из цикла, мы можем применять оператор `continue`. Например, изменим предыдущий пример, только вместо `break` используем оператор `continue`:

```

1  for (let i=1; i <= 6; i++) {
2
3    if(i===4) continue;
4    console.log(i);
5  }
6  console.log ("Конец работы");

```

В этом случае, когда значение переменная *i* станет равной 4, то выражение `i===4` возвратит `true`, поэтому будет выполняться конструкция `if(i===4) continue;` С помощью оператора `continue`

она завершит текущую итерацию, далее идущие инструкции цикла не будут выполняться, а произойдет переход к следующей итерации:

```
1
2
3
5
6
Конец работы
```

Цикл **for...in**

Цикл **for...in** предназначен главным образом для перебора объектов. Его формальное определение:

```
1  for (свойство in объект) {
2      // действия
3  }
```

Этот цикл перебирает все свойства объекта. Например:

```
1  const person = {name: "Tom", age: 37};
2  for (prop in person) {
3
4      console.log(prop);
5  }
```

Здесь перебирается объект `person`, который имеет два свойства - `name` и `age`. Соответственно на консоли мы увидим:

```
name
age
```

Получив свойства и используя специальный синтаксис `объект[свойство]`, мы можем получить значение каждого свойства:

```
1  const person = {name: "Tom", age: 37};
2  for (prop in person) {
3
4      console.log (prop, person[prop]);
5  }
```

Консольный вывод:

```
name Tom
age 37
```

Цикл **for...of**

Цикл **for...of** предназначен для перебора наборов данных. Например, строка представляет фактически набор символов. И мы можем перебрать ее с помощью данного цикла:

```
1  const text = "Hello";
2  for (char of text) {
3
```

```

4     console.log(char);
5 }

```

В итоге цикл перебирает все символы строки `text` и помещает каждый текущий символ в переменную `ch`, значение которой затем выводится на консоль.

```

Н
е
1
1
о

```

Другим примером может быть перебор массива:

```

1 const people = ["Tom", "Sam", "Bob"];
2 for (const person of people) {
3     console.log(person);
4 }

```

В данном случае цикл перебирает элементы массива `people`. Каждый элемент последовательно помещается в константу `person`. И далее мы можем вывести ее значение на консоль:

```

Tom
Sam
Bob

```

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №5

1. Напишите программу, которая проверяет, является ли введенное пользователем число положительным, отрицательным или нулем, и выводит соответствующее сообщение.
2. Разработайте программу, которая проверяет, является ли введенное пользователем число четным или нечетным, и выводит соответствующее сообщение.
3. Создайте программу для определения наибольшего из двух введенных пользователем чисел.
4. Напишите программу, которая находит сумму всех чисел от 1 до N , где N - число, введенное пользователем.
5. Реализуйте программу для вычисления факториала числа, введенного пользователем.
6. Создайте программу, которая проверяет, является ли введенное пользователем число простым, и выводит соответствующее сообщение.
7. Напишите программу, которая выводит все числа от 1 до 100, кратные 3.

8. Разработайте программу для вывода таблицы умножения на заданное пользователем число.
9. Создайте программу, которая выводит все простые числа от 1 до N, где N - число, введенное пользователем.
10. Напишите программу, которая находит наименьший делитель числа, введенного пользователем.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая цель работы по созданию программ с использованием условных операторов и циклов в JavaScript?
2. Какие конструкции управления потоком выполнения программы изучаются в данной работе?
3. Что такое условные операторы, и какова их роль в программировании на JavaScript?
4. Какие формы имеет оператор if в JavaScript, и как они используются?
5. В чем отличие между оператором if и оператором else if в JavaScript?
6. Какие виды циклов доступны в JavaScript, и для чего они используются?
7. Как работает цикл for в JavaScript, и каков его синтаксис?
8. Какие примеры задач можно решить с помощью циклов и условных операторов в JavaScript?
9. Как можно использовать циклы для обхода массивов или объектов в JavaScript?
10. Какие особенности у оператора do..while по сравнению с оператором while в JavaScript?

ПРАКТИЧЕСКАЯ РАБОТА №6. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ

Цель работы: овладеть навыками создания и использования функций в JavaScript для упрощения и структурирования кода, а также для повторного использования частей программы.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Функции представляют собой набор инструкций, которые можно повторно вызывать в различных частях программы по имени функции. В общем случае синтаксис определения функции выглядит следующим образом:

```
1  function имя функции(параметры) {  
2  
3      // Инструкции  
4  }
```

Определение функции начинается с ключевого слова `function`, после которого следует имя функции. Наименование функции подчиняется тем же правилам, что и наименование переменной: оно может содержать только цифры, буквы, символы подчеркивания и доллара (\$) и должно начинаться с буквы, символа подчеркивания или доллара.

После имени функции в скобках идет перечисление параметров. Даже если параметров у функции нет, то просто идут пустые скобки. Затем в фигурных скобках идет тело функции, содержащее набор инструкций.

Определим простейшую функцию:

```
1  function hello () {  
2  
3      console.log("Hello");  
4  }
```

Данная функция называется `hello ()`. Она не принимает никаких параметров и все, что она делает, это выводит на консоль браузера строку "Hello".

Чтобы функция выполнила свою работу, нам надо ее вызвать. Общий синтаксис вызова функции:

```
1  имя_функции(параметры)
```

При вызове после имени вызываемой функции в скобках указывается список параметров. Если функция не имеет параметров, то указываются пустые скобки.

Например, определим и вызовем простейшую функцию:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="utf-8" />
5    <title>Hello</title>
6  </head>
7  <body>
8  <script>
9    // определение функции
10   function hello () {
11
12     console.log("Hello");
13   }
14   // вызов функции
15   hello ();
16 </script>
17 </body>
18 </html>

```

В данном случае функция hello не принимает параметров, поэтому при ее вызове указываются пустые скобки:

```

1  hello ();

```

Отличительной чертой функций является то, что их можно многократно вызывать в различных местах программы:

```

1  // определение функции
2  function hello () {
3    console.log("Hello");
4  }
5  // вызов функции
6  hello ();
7  hello ();
8  hello ();

```

Переменные и константы в качестве функций

Подобно тому, как константам и переменным присваиваются простейшие значения (числа, строки и т.д.), также им можно присваивать функции. Затем через такую переменную или константу можно вызвать присвоенную ей функцию:

```

1  <!DOCTYPE html>
2  <html>

```



```

3   <head>
4     <meta charset="utf-8" />
5     <title>Hello</title>
6   </head>
7   <body>
8     <script>
9     // определение функции
10    function hello () {
11      console.log("Hello");
12    }
13    // передача константе message ссылки на функцию hello
14    const message = hello;
15    message (); // вызываем функцию, ссылка на которую хранится в константе message
16  </script>
17 </body>
18 </html>

```

Присвоив константе или переменной функцию:

```

1  const message = hello;
    затем мы можем по имени константы/переменной вызывать
эту функцию:

```

```

1  message ();

```

Также мы можем динамически менять функции, которые хранятся в переменной:

```

1  function goodMorning () {
2    console.log ("Доброе утро");
3  }
4  function goodEvening () {
5    console.log ("Добрый вечер");
6  }
7  let message = goodMorning; // присваиваем переменной message функцию
8  goodMorning
9  message (); // Доброе утро
10 message = goodEvening; // меняем функцию в переменной message
10 message (); // Добрый вечер

```

Функции-выражения и анонимные функции

Необязательно давать функциям определенное имя. Можно использовать анонимные функции. Такие функции при определении присваиваются константе или переменной. Эти функции еще называют функции-выражения (function expression):

```

1  const message = function () {
2
3    console.log ("Hello JavaScript");
4  }
5  message ();

```

Используя имя константы или переменной, которой присвоена функция, можно вызывать эту функцию.

Локальные функции

JavaScript позволяет определять локальные функции - функции внутри других функций. Локальные функции видно только в рамках внешней функции, в которой они определены. Например:

```

1  function print () {
2
3      printHello ();
4      printHello ();
5      printHello ();
6
7      function printHello () {
8          console.log("Hello");
9      }
10 }
11 print ();
12 printHello (); // Uncaught ReferenceError: printHello is not defined - локальную
    функцию можно вызвать только из ее окружающей функции

```

Здесь внутри функции `print` определена локальная функция `printHello`, которая просто выводит строку "Hello". И внутри функции `print` мы можем вызвать локальную функцию `printHello`, однако вне окружающей функции локальную функцию вызвать нельзя.

Данный пример довольно простой и не имеет большого смысла. Однако, как правило, локальные функции определяются для таких действий, которые применяются многократно только в рамках какой-то одной функции и больше нигде. К минусам локальных функции можно отнести то, что они создаются всякий раз, когда происходит вызов внешней функции.

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №6

1. Напишите программу, которая вычисляет площадь круга по заданному радиусу. Для этого используйте функцию `calculateCircleArea(radius)`, которая принимает радиус круга и возвращает его площадь.
2. Разработайте программу для нахождения суммы чисел в заданном диапазоне. Создайте функцию `calculateSum (start,`

- end), которая принимает начальное и конечное значения диапазона и возвращает сумму всех чисел в этом диапазоне.
3. Создайте программу для проверки является ли число простым. Напишите функцию `isPrime(number)`, которая принимает число и возвращает `true`, если оно простое, и `false` в противном случае.
 4. Напишите программу для перевода градусов Цельсия в градусы Фаренгейта. Создайте функцию `celsiusToFahrenheit(celsius)`, которая принимает температуру в градусах Цельсия и возвращает температуру в градусах Фаренгейта.
 5. Разработайте программу для определения наибольшего общего делителя (НОД) двух чисел. Напишите функцию `gcd(a, b)`, которая принимает два числа и возвращает их НОД.
 6. Создайте программу для нахождения факториала числа. Напишите функцию `factorial(n)`, которая принимает число и возвращает его факториал.
 7. Напишите программу для определения количества простых чисел в заданном диапазоне. Создайте функцию `countPrimes(start, end)`, которая принимает начальное и конечное значения диапазона и возвращает количество простых чисел в этом диапазоне.
 8. Разработайте программу для нахождения среднего значения списка чисел. Напишите функцию `calculateAverage(numbers)`, которая принимает массив чисел и возвращает их среднее значение.
 9. Создайте программу для проверки является ли строка палиндромом. Напишите функцию `isPalindrome(str)`, которая принимает строку и возвращает `true`, если она является палиндромом, и `false` в противном случае.
 10. Напишите программу для нахождения наибольшего элемента в массиве чисел. Создайте функцию `findMax(arr)`, которая принимает массив чисел и возвращает его наибольший элемент.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какова цель работы по созданию программ с использованием функций в JavaScript?

2. Как определяются функции в JavaScript, и какие основные элементы входят в их синтаксис?
3. Какова роль параметров в определении функций, и как они используются в теле функции?
4. Как вызывается функция в JavaScript, и какие правила следует при этом учитывать?
5. Какие преимущества предоставляют функции в программировании, особенно в контексте структурирования и повторного использования кода?
6. Можно ли использовать переменные и константы для хранения функций в JavaScript, и как это реализуется?
7. В чем заключается концепция функций-выражений или анонимных функций в JavaScript, и как они используются?
8. Какие виды функций в JavaScript вы знаете, и как они отличаются друг от друга?
9. Как можно изменять функции, хранящиеся в переменных, и как это может быть полезно в практическом программировании?
10. Как можно использовать локальные функции в JavaScript, и в каких случаях они предпочтительны по сравнению с глобальными функциями?

ПРАКТИЧЕСКАЯ РАБОТА №7. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТНОЙ МОДЕЛИ ДОКУМЕНТА

Цель работы: овладеть навыками работы с объектной моделью документа (DOM) в JavaScript для управления содержимым и структурой веб-страницы, а также для создания интерактивных веб-приложений.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Объектная модель документа, или «DOM», определяет логическую структуру документов HTML и выступает в качестве интерфейса для веб-страниц. Благодаря использованию языков программирования, таких как JavaScript, мы можем получить доступ к DOM, чтобы управлять веб-сайтами и делать их интерактивными.

Браузер также создает представление этого документа, известное как объектная модель документа. Благодаря использованию DOM JavaScript может получать доступ и изменять содержимое и элементы веб-сайта.

Чтобы просмотреть DOM с помощью веб-браузера, кликните правой кнопкой мыши в любом месте страницы и выберите «Просмотреть код». Будут открыты Инструменты разработчика (рисунок 7.1):

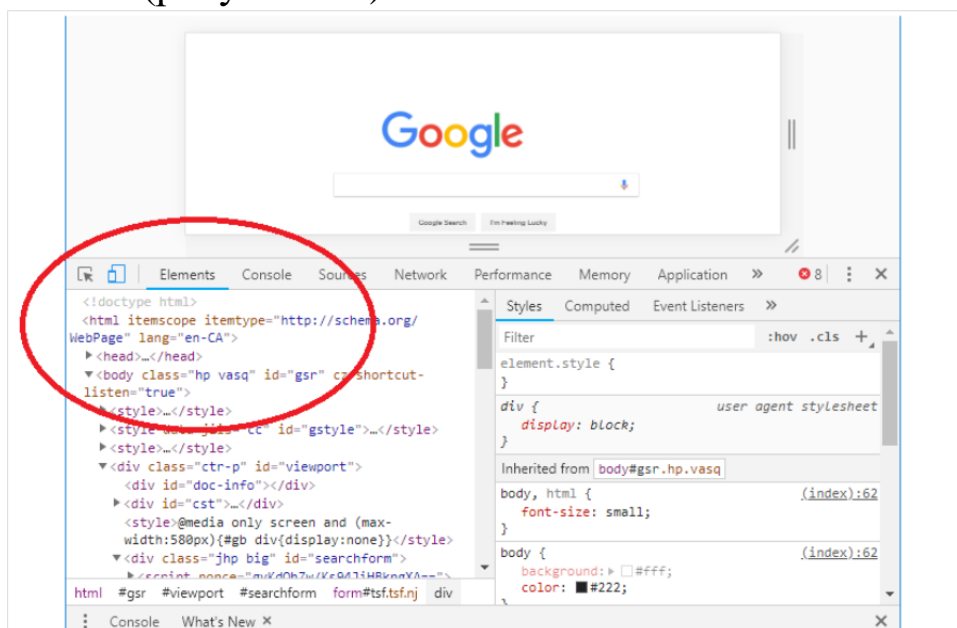


Рисунок 7.1 – Инструменты разработчика

DOM отображается на вкладке Элементы. Вы также можете просмотреть его, выбрав вкладку «Консоль» и введя «document».

Объект document

Объект document является встроенным объектом, содержащим множество свойств и методов. Мы обращаемся к этому объекту и манипулируем им с помощью JavaScript. А манипулируя DOM, мы можем сделать веб-страницы интерактивными! Поскольку мы больше не ограничены только созданием статических сайтов со стилизованным HTML-контентом.

Теперь мы можем создавать приложения, которые обновляют данные страницы без необходимости обновления страницы, мы можем дать пользователям возможность настраивать макет страницы, мы можем создавать элементы перетаскивания, браузерные игры, часы, таймеры и сложную анимацию. Работа с DOM открывает множество возможностей.

Итак, давайте выполним нашу первую манипуляцию с DOM... Перейдите на сайт www.google.com и откройте Инструменты разработчика. Затем выберите вкладку Консоль и введите следующее:

```
1 document.body.style.backgroundColor = 'orange';
```

Нажмите Enter, и вы увидите, что цвет фона теперь изменился на оранжевый. Конечно, вы не редактировали исходный код Google (!), но вы изменили способ отображения содержимого локально в вашем браузере, манипулируя объектом document.

document является объектом, свойство body которого мы выбрали для редактирования путем доступа к атрибуту style и изменения значения свойства backgroundColor на orange.

Обратите внимание, что в JavaScript мы используем способ написания имен backgroundColor, а не background-color, как в CSS. Любое свойство CSS через дефис будет записано в JavaScript в camelCase. Вы можете увидеть настройки DOM в разделе элемента body на вкладке Elements или набрав document.body в консоли.

Поскольку мы работаем в браузере напрямую с DOM, мы фактически не меняем исходный код. Если вы обновите браузер, все вернется в исходное состояние.

Дерево DOM и узлы

Во многом из-за структуры DOM, его часто называют Дерево DOM (рис. 7.2).

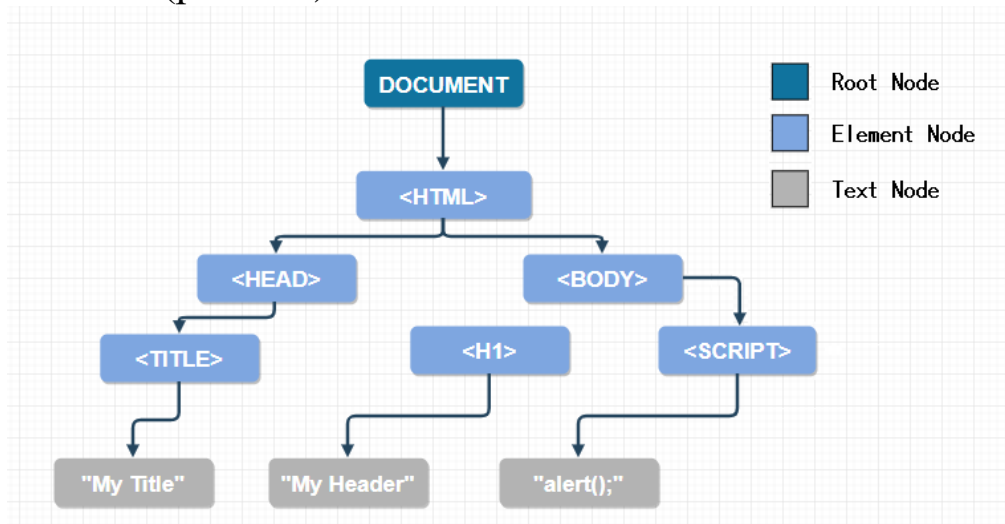


Рисунок 7.2 – Дерево DOM

Дерево состоит из объектов, называемых узлами. Существует много разных типов узлов, но чаще всего вы будете работать с узлами элементов (элементы HTML), текстовыми узлами (любое текстовое содержимое) и узлами комментариев (закомментированный код). Объект document является его собственным узлом, который находится в корневом каталоге.

```

1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title>Nodes</title>
6 </head>
7
8 <body>
9   <h1>This is an element node</h1>
10  <!-- This is a comment node -->
11  This is a text node.
12 </body>
13
14 </html>
  
```

При работе с узлами DOM их также называют родителями, потомками и элементами одного уровня, в зависимости от их связи с другими узлами.

В приведенном выше примере кода узел элемента `html` является родительским узлом, а элементы `head` и `body` являются его потомками. `body` содержит три дочерних узла (которые являются элементами одного уровня по отношению друг к другу — во многом похоже на семейное древо).

Как определить тип узла

Таким образом, у каждого узла в документе есть тип, мы можем получить доступ к типу, используя свойство `nodeType`.

Давайте рассмотрим несколько примеров типов, которые существуют в нашем предыдущем примере. Наши `html`, `title`, `body` и `h1` относятся к типу `ELEMENT_NODE` со значением 1.

Наш текст `This is a text node.`, расположенный внутри `body`, который не находится внутри прочих элементов — это `TEXT_NODE` со значением 3. И наш комментарий `<!-- This is a comment node -->` — это тип `COMMENT_NODE` со значением 8.

Доступ к элементам DOM

В этом разделе мы рассмотрим методы, которые можем использовать, чтобы получить доступ к элементам DOM: `getElementById()`, `getElementsByClassName()`, `getElementsByTagName()`, `querySelector()` и `querySelectorAll()`.

Мы будем работать с HTML-файлом, который состоит из множества элементов. HTML выглядит следующим образом:

```

1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <title>Master the DOM! </title>
6 </head>
7
8 <body>
9
10 <h1>Master the DOM! </h1>
11
12 <div id="test">I'm an ID</div>
13
14 <div class="test">I'm a class</div>
```



```

15 <div class="test">I'm another class</div>
16
17 <section>I'm a tag</section>
18 <section>I'm another tag</section>
19
20 <div id="test-query">Use a query selector</div>
21
22 <div class="test-query-all">Use query selector ALL</div>
23 <div class="test-query-all">Use query selector ALL</div>
24
25 </body>
26 </html>

```

Результат можно увидеть на рисунке 7.3.

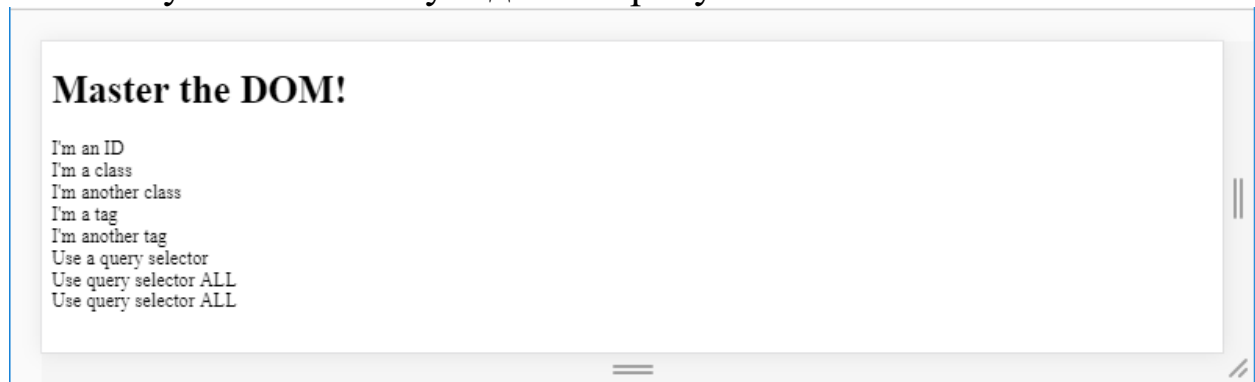


Рисунок 7.3 – Результат

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №7

1. Напишите программу, которая изменяет текст элемента заголовка страницы. Для этого используйте объектную модель документа для доступа к элементу и измените его текст.
2. Создайте программу для изменения стиля элемента. Напишите функцию, которая меняет цвет фона элемента на странице по его идентификатору.
3. Разработайте программу для скрытия и отображения элементов на странице. Напишите функции для скрытия и отображения элемента по его классу.
4. Напишите программу для добавления нового элемента на страницу. Создайте функцию, которая добавляет новый элемент (например, абзац) в конец содержимого элемента с определенным идентификатором.
5. Создайте программу для удаления элемента из страницы. Напишите функцию для удаления элемента по его классу.

6. Напишите программу для обработки событий на странице. Создайте функцию, которая выводит сообщение при клике на кнопку.
7. Разработайте программу для изменения содержимого элемента списка. Напишите функцию, которая изменяет текст элемента списка по его индексу.
8. Создайте программу для создания таблицы на странице. Напишите функцию, которая генерирует таблицу с заданным количеством строк и столбцов и добавляет ее на страницу.
9. Напишите программу для валидации формы на странице. Создайте функцию, которая проверяет правильность заполнения полей формы и выводит сообщение об ошибке, если что-то было введено неправильно.
10. Разработайте программу для изменения размера изображения на странице. Напишите функцию, которая изменяет размер изображения по его идентификатору.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет собой DOM, и какую роль он играет в веб-разработке?
2. Как можно просмотреть DOM с помощью инструментов разработчика веб-браузера?
3. Что представляет собой объект document, и какие возможности он предоставляет для взаимодействия с содержимым веб-страницы?
4. Как можно изменить стиль элемента на веб-странице с использованием JavaScript и DOM?
5. Что такое дерево DOM, и какова его структура?
6. Какие типы узлов существуют в DOM, и как можно определить тип узла?
7. Как можно получить доступ к элементам DOM с помощью JavaScript, и какие методы для этого существуют?
8. Чем отличаются методы getElementById(), getElementsByName(), getElementsByTagName() от методов querySelector() и querySelectorAll()?

9. Какие методы и свойства можно использовать для манипуляции содержимым и структурой веб-страницы с помощью DOM?

ПРАКТИЧЕСКАЯ РАБОТА №8. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ПОЛЬЗОВАТЕЛЬСКИХ ОБЪЕКТОВ

Цель работы: овладеть навыками создания пользовательских объектов в JavaScript для организации структуры данных и управления информацией в приложениях.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Объект является фундаментальным типом данных в языке JavaScript. **Объект** — это составное значение: он объединяет в себе набор значений (простых значений или других объектов) и позволяет сохранять и извлекать эти значения по именам.

Объект является неупорядоченной коллекцией свойств, каждое из которых имеет имя и значение. Имена свойств являются строками, поэтому можно сказать, что объекты отображают строки в значения. Такое отображение строк в значения может называться по-разному: возможно, вы уже знакомы с такой фундаментальной структурой данных, как «хеш», «словарь» или «ассоциативный массив». Однако объект представляет собой нечто большее, чем простое отображение строк в значения.

Помимо собственных свойств объекты в языке JavaScript могут также наследовать свойства от других объектов, известных под названием «прототипы». Методы объекта — это типичные представители унаследованных свойств, а «наследование через прототипы» является ключевой особенностью языка JavaScript.

Объекты в языке JavaScript являются динамическими - обычно они позволяют добавлять и удалять свойства - но они могут использоваться также для имитации статических объектов и «структур», которые имеются в языках программирования со статической системой типов. Кроме того, они могут использоваться (если не учитывать, что объекты отображают строки в значения) для представления множеств строк.

Любое значение в языке JavaScript, не являющееся строкой, числом, true, false, null или undefined, является объектом. И даже строки, числа и логические значения, не являющиеся объектами, могут вести себя как неизменяемые объекты (имеют объекты-обертки String, Number и т.п.).

Объекты являются изменяемыми значениями и операции с ними выполняются по ссылке, а не по значению. Если переменная *x* ссылается на объект, и выполняется инструкция `var y = x;`, в переменную *y* будет записана ссылка на тот же самый объект, а не его копия. Любые изменения, выполняемые в объекте с помощью переменной *y*, будут также отражаться на переменной *x*.

Свойство имеет имя и значение. Именем свойства может быть любая строка, включая и пустую строку, но объект не может иметь два свойства с одинаковыми именами. Значением свойства может быть любое значение, допустимое в языке JavaScript, или (в ECMAScript 5) функция чтения или записи (или обе).

В дополнение к именам и значениям каждое свойство имеет ряд ассоциированных с ним значений, которые называют атрибутами свойства:

- Атрибут **writable** определяет доступность значения свойства для записи.
- Атрибут **enumerable** определяет доступность имени свойства для перечисления в цикле `for/in`.

Атрибут **configurable** определяет возможность настройки, т.е. удаления свойства и изменения его атрибутов.

До появления стандарта ECMAScript 5 все свойства в объектах, создаваемые программой, доступны для записи, перечисления и настройки. В ECMAScript 5 предусматривается возможность настройки атрибутов ваших свойств.

В дополнение к свойствам каждый объект имеет три атрибута объекта:

- Атрибут `prototype` содержит ссылку на другой объект, от которого наследуются свойства.
- Атрибут `class` содержит строку с именем класса объекта и определяет тип объекта.
- Флаг `extensible` (в ECMAScript 5) указывает на возможность добавления новых свойств в объект.

Наконец, ниже приводится описание некоторых терминов, которые помогут нам различать три обширные категории объектов в языке JavaScript и два типа свойств:

Объект базового языка

Это объект или класс объектов, определяемый спецификацией ECMAScript. Массивы, функции, даты и регулярные выражения (например) являются объектами базового языка.

Объект среды выполнения

Это объект, определяемый средой выполнения (такой как веб-браузер), куда встроен интерпретатор JavaScript. Объекты HTMLElement, представляющие структуру веб-страницы в клиентском JavaScript, являются объектами среды выполнения. Объекты среды выполнения могут также быть объектами базового языка, например, когда среда выполнения определяет методы, которые являются обычными объектами Function базового языка JavaScript.

Пользовательский объект

Любой объект, созданный в результате выполнения программного кода JavaScript.

Создание объектов

Объекты можно создавать с помощью литералов объектов, ключевого слова new и (в ECMAScript 5) функции Object.create().

Литералы объектов

Самый простой способ создать объект заключается во включении в программу литерала объекта. Литерал объекта — это заключенный в фигурные скобки список свойств (пар имя/значение), разделенных запятыми. Именем свойства может быть идентификатор или строковый литерал (допускается использовать пустую строку). Значением свойства может быть любое выражение, допустимое в JavaScript - значение выражения (это может быть простое значение или объект) станет значением свойства.

Ниже приводятся несколько примеров создания объектов:

```
var empty = {}; // Объект без свойств
var point = {x:0, y:0}; // Два свойства
var point2 = {x: point.x, y: point.y+1}; // Более сложные значения
var site = {
  "url site": "www.professorweb.ru", // Имена свойств с пробелами
```

```
'desc-text': "Платформа .NET Framework",           // и дефисами, поэтому исп.
кавычки
```

```
author: {                                           // Значением этого свойства является
    firstname: "Alexandr",                          // объект. Обратите внимание, что
    surname: "Frolov"                               // имена этих свойств без кавычек.
};
```

В ECMAScript 5 (и в некоторых реализациях ECMAScript 3) допускается использовать зарезервированные слова в качестве имен свойств без кавычек. Однако в целом имена свойств, совпадающие с зарезервированными словами, в ECMA-Script 3 должны заключаться в кавычки. В ECMAScript 5 последняя запятая, следующая за последним свойством в литерале объекта, игнорируется. В большинстве реализаций ECMAScript 3 завершающие запятые также игнорируются, но IE интерпретирует их наличие как ошибку.

Литерал объекта — это выражение, которое создает и инициализирует новый объект всякий раз, когда производится вычисление этого выражения. Значение каждого свойства вычисляется заново, когда вычисляется значение литерала. Это означает, что с помощью единственного литерала объекта можно создать множество новых объектов, если этот литерал поместить в тело цикла или функции, которая будет вызываться многократно, и что значения свойств этих объектов могут отличаться друг от друга.

Создание объектов с помощью оператора new

Оператор new создает и инициализирует новый объект. За этим оператором должно следовать имя функции. Функция, используемая таким способом, называется конструктором и служит для инициализации вновь созданного объекта. Базовый JavaScript включает множество встроенных конструкторов для создания объектов базового языка. Например:

```
var o = new Object ();           // Создать новый пустой объект: то же, что и {}
var a = new Array ();           // Создать пустой массив: то же, что и []
var d = new Date ();            // Создать объект Date, представляющий текущее время
var r = new RegExp("js");       // Создать объект RegExp для операций сопоставления с
шаблоном
```

Помимо этих встроенных конструкторов имеется возможность определять свои собственные функции-

конструкторы для инициализации вновь создаваемых объектов. О том, как это делается, рассказывается в следующей статье.

Object.create()

Стандарт ECMAScript 5 определяет метод `Object.create()`, который создает новый объект и использует свой первый аргумент в качестве прототипа этого объекта. Дополнительно `Object.create()` может принимать второй необязательный аргумент, описывающий свойства нового объекта.

`Object.create()` является статической функцией, а не методом, вызываемым относительно некоторого конкретного объекта. Чтобы вызвать эту функцию, достаточно передать ей желаемый объект-прототип:

```
// obj наследует свойства x и y
var obj = Object.create({x:1, y:2});
```

Чтобы создать объект, не имеющий прототипа, можно передать значение `null`, но в этом случае вновь созданный объект не унаследует ни каких-либо свойств, ни базовых методов, таких как `toString()` (а это означает, что этот объект нельзя будет использовать в выражениях с оператором `+`):

```
// obj2 не наследует ни свойств, ни методов
var obj2 = Object.create(null);
```

Если в программе потребуется создать обычный пустой объект (который, например, возвращается литералом `{}` или выражением `new Object()`), передайте в первом аргументе `Object.prototype`:

```
// obj3 подобен объекту, созданному
// с помощью {} или new Object()
var obj3 = Object.create(Object.prototype);
```

Возможность создавать новые объекты с произвольными прототипами (скажем иначе: возможность создавать «наследников» от любых объектов) является мощным инструментом, действие которого можно имитировать в ECMAScript 3 с помощью функции, представленной в примере ниже:

```
// inherit() возвращает вновь созданный объект, наследующий свойства
// объекта-прототипа p. Использует функцию Object.create() из ECMAScript 5,
// если она определена, иначе используется более старый прием.
function inherit(p) {
  if (p == null) throw TypeError(); // p не может быть значением null
```



```

if (Object.create)                // Если Object.create() определена...
    return Object.create(p);      // использовать ее.

var t = typeof p;                // Иначе выяснить тип и проверить его
if (t !== "object" && t !== "function")
    throw TypeError ();

function f () {};                // Определить пустой конструктор.
f. prototype = p;                // Записать в его свойство prototype
                                // ссылку на объект p.

return new f ();                 // Использовать f () для создания
                                // "наследника" объекта p.
}

```

Реализация функции `inherit ()` приобретет больше смысла, как только мы познакомимся с конструкторами в следующей статье. А пока просто считайте, что она возвращает новый объект, наследующий свойства объекта в аргументе. Обратите внимание, что функция `inherit ()` не является полноценной заменой для `Object.create()`: она не позволяет создавать объекты без прототипа и не принимает второй необязательный аргумент, как `Object.create()`.

Получение и изменение свойств

Получить значение свойства можно с помощью операторов точки (.) и квадратных скобок ([]). Слева от оператора должно находиться выражение, возвращающее объект. При использовании оператора точки справа должен находиться простой идентификатор, соответствующий имени свойства. При использовании квадратных скобок в квадратных скобках должно указываться выражение, возвращающее строку, содержащую имя требуемого свойства:

```

// Простой объект
var user = {login:'kot86', name:'Alexandr', age:26};

var login = user.login;          // Получить свойство "login" объекта user
var name = user.name;            // Получить свойство "name" объекта user
var age = user['age'];            // Получить свойство "age" объекта user

```

Чтобы создать новое свойство или изменить значение существующего свойства, также используются операторы точки и квадратные скобки, как в операциях чтения значений свойств, но само выражение помещается уже слева от оператора присваивания:

```

user.age = 28; // Изменить значение свойства 'age'
user['login'] = 'kot84'; // Изменить значение свойства 'login'
user['surname'] = 'Frolov'; // Создать новое свойство 'surname'

```

В ECMAScript 3 идентификатор, следующий за точкой, не может быть зарезервированным словом: нельзя записать обращение к свойству `o.for` или `o.class`, потому что `for` является ключевым словом, а `class` - словом, зарезервированным для использования в будущем.

Если объект имеет свойства, имена которых совпадают с зарезервированными словами, для доступа к ним необходимо использовать форму записи с квадратными скобками: `o["for"]` и `o["class"]`. Стандарт ECMAScript 5 ослабляет это требование (как это уже сделано в некоторых реализациях ECMAScript 3) и допускает возможность использования зарезервированных слов после оператора точки.

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №8

1. Создайте программу для представления информации о студенте с помощью пользовательского объекта. Каждый объект должен содержать поля с именем, возрастом, факультетом и курсом студента.
2. Напишите программу для создания пользовательского объекта "Книга". Объект должен содержать поля с названием, автором и годом издания книги.
3. Разработайте программу для хранения информации о сотруднике компании. Создайте объект "Сотрудник", содержащий поля с именем, должностью и зарплатой сотрудника.
4. Напишите программу для учета товаров на складе с помощью объектов. Создайте пользовательский объект "Товар", содержащий поля с названием, количеством и ценой товара.
5. Создайте программу для организации списка задач с использованием объектов. Каждый объект "Задача" должен содержать поля с описанием, статусом выполнения и сроком выполнения задачи.
6. Разработайте программу для хранения информации о покупках с помощью объектов. Создайте объект "Покупка",

содержащий поля с названием товара, его стоимостью и количеством.

7. Напишите программу для представления информации о футбольной команде с использованием пользовательских объектов. Каждый объект "Игрок" должен содержать поля с именем, позицией и возрастом игрока.
8. Создайте программу для учета расходов с помощью пользовательских объектов. Объект "Расход" должен содержать поля с описанием расхода, его суммой и датой.
9. Разработайте программу для организации списка контактов с использованием объектов. Каждый объект "Контакт" должен содержать поля с именем, номером телефона и адресом электронной почты.
10. Напишите программу для учета поездок с помощью пользовательских объектов. Создайте объект "Поездка", содержащий поля с местом назначения, датой и расходами.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляют собой объекты в языке JavaScript, и как они организованы?
2. Какие атрибуты свойств объекта могут быть определены в JavaScript?
3. Какие атрибуты объекта существуют в JavaScript, и что они определяют?
4. Чем отличается объект базового языка от объекта среды выполнения и пользовательского объекта?
5. Как можно создать объект в JavaScript с помощью литералов объектов?
6. Как можно создать объект в JavaScript с использованием ключевого слова new?
7. Как можно создать объект в JavaScript с помощью функции Object.create()?
8. Каким образом происходит доступ к свойствам объекта в JavaScript?
9. Каким образом происходит изменение и удаление свойств объекта в JavaScript?

ПРАКТИЧЕСКАЯ РАБОТА №9. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ЭЛЕМЕНТОВ ФОРМЫ

Цель работы: овладеть навыками работы с элементами формы в HTML и их управлением с помощью JavaScript для создания интерактивных пользовательских интерфейсов.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Один из способов взаимодействия с пользователями представляют html-формы. Например, если нам надо получить от пользователя некоторую информацию, мы можем определить на веб-странице формы, которая будет содержать текстовые поля для ввода информации и кнопку для отправки. И после ввода данных мы можем обработать введенную информацию.

Для создания формы используется элемент `<form>`:

```
1 <form id="search" name="search">
2 </form>
```

В JavaScript форма представлена объектом `HtmlFormElement`. И после создания формы мы можем к ней обратиться различными способами.

Получение формы

Первый способ заключается в прямом обращении по имени формы:

```
1 const searchForm = document. search;
```

Второй способ состоит в обращении к коллекции форм документа - коллекция **forms** и поиске в ней нужной формы:

```
1 const searchForm1 = document. forms["search"]; // по имени
2 const searchForm2 = document. forms [0]; // по индексу
```

Третий способ представляет получение форм стандартными методами для поиска элемента по `id`, по тегу или по селектору. Например:

```
1 const formById = document.getElementById("search");
2 const formByName = document.getElementsByName("search») [0];
3 const formBySelector = document.querySelector("form");
```

Свойства и методы форм

Форма имеет ряд свойств, из которых перечислю основные:

- **name:** имя формы
- **elements:** коллекция элементов формы
- **length:** количество элементов формы

- **action:** значение атрибута action - адрес отправки формы
- **method:** значение атрибута method - метод HTTP, применяемый для отправки

Например, получим свойства формы:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
</head>
<body>
  <form id="search" name="search" action="https://google.com/search" method="get">
    <input type="text" id="key" name="q" />
    <input type="submit" id="send" name="send" />
  </form>
  <script>
    const form = document.getElementById("search");
    console.log (form. elements); // HTMLFormControlsCollection (2) [input, input, q: input, send:
input]
    console.log (form. length); // 2
    console.log(form.name); // search
    console.log (form. action); // https://google.com/search
    console.log(form.method);    // get
  </script>
</body>
</html>
```

Среди методов формы надо отметить метод submit (), который отправляет данные формы на сервер, и метод reset (), который очищает поля формы:

```
1  const form = document.forms["search"];
2  form.submit();
3  form.reset();
```

Элементы форм

Форма может содержать различные элементы ввода html: input, textarea, button, select и т.д. Для каждого из элементов существует свой тип JavaScript:

| html-элемент | Тип JavaScript |
|-------------------------------|---------------------|
| <input> | HTMLInputElement |
| <textarea> | HTMLTextAreaElement |
| <select> | HTMLSelectElement |
| <option> (в списках <select>) | HTMLOptionElement |

Для получения элементов форм можно использовать два способа:

Применение стандартных методов `getElementById()`, `getElementsByClassName()`, `getElementsByTagName()`, `getElementsByName()`, `querySelector()` и `querySelectorAll()` для поиска элементов соответственно по `id`, классу, тегу, имени или селектору. Например, возьмем ранее определенную форму и получим ее поле ввода:

```
1 // получаем элемент по id="key"
2 const keyField = document.getElementById("key");
3 console.log(keyField);
```

Использование свойства `elements` соответствующей формы. Например:

```
1 const form = document.getElementById("search");
2 // получение поля по индексу
3 const keyField = form.elements[0];
4 console.log(keyField);
5 // получение этого же поля, но через имя
6 const keyField2 = form.elements["q"];
7 console.log(keyField2);
```

Использование имени формы и элемента. Например:

```
1 // поле q на форме search
2 const keyField = document.search.q;
3 console.log(keyField);
```

Свойства элементов форм

Все они имеют ряд общих свойств и методов. Также, как и форма, элементы форм имеют свойство `name`, с помощью которого можно получить значение атрибута `name`. Другим важным свойством является свойство `value`, которое позволяет получить или изменить значение поля:

```
1 <form name="search">
2   <input type="text" name="key" value="hello world"></input>
3   <input type="submit" name="send"></input>
4 </form>
5 <script>
6 const form = document.getElementById("search");
7 // получение поля формы по имени
8 const keyField = form.elements["key"];
9 // получение значения поля
10 console.log(keyField.value);
11 // установка значения поля
12 keyField.value = "Enter a string";
13 </script>
```

Свойство `type` позволяет получить тип поля ввода. Это либо название тега элемента (например, `textarea`), либо значение атрибута `type` у элементов `input`.

```

1  const form = document.getElementById("search");
2  // получение поля формы по имени
3  const keyField = form.elements["key"];
4  // получение значения поля
5  console.log(keyField.type); // text

```

Из методов можно выделить методы `focus()` (устанавливает фокус на элемент) и `blur()` (убирает фокус с элемента):

```

1  const searchForm = document.forms["search"];
2  const keyField = searchForm.elements["key"];
3  keyField.focus();

```

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №9

1. Создайте программу для расчета площади прямоугольника с использованием формы. На странице должны быть поля для ввода длины и ширины прямоугольника, а также кнопка для расчета площади. После нажатия кнопки программа должна выводить результат на страницу.
2. Напишите программу для проверки является ли введенное число четным или нечетным с использованием формы. Создайте поле для ввода числа и кнопку для проверки. После нажатия кнопки программа должна выводить сообщение о том, является ли число четным или нечетным.
3. Разработайте программу для конвертации температуры из градусов Цельсия в градусы Фаренгейта с использованием формы. На странице должно быть поле для ввода температуры в градусах Цельсия и кнопка для конвертации. После нажатия кнопки программа должна выводить результат на страницу.
4. Напишите программу для расчета суммы двух чисел с использованием формы. Создайте два поля для ввода чисел и кнопку для расчета суммы. После нажатия кнопки программа должна выводить результат на страницу.
5. Создайте программу для проверки пароля с использованием формы. На странице должно быть поле для ввода пароля и кнопка для проверки. После нажатия кнопки программа должна выводить сообщение о правильности введенного пароля.
6. Разработайте программу для создания списка задач с использованием формы. На странице должно быть поле для

ввода задачи и кнопка для добавления в список. После добавления задачи программа должна выводить ее в списке задач.

7. Напишите программу для валидации email адреса с использованием формы. На странице должно быть поле для ввода email и кнопка для проверки. После нажатия кнопки программа должна выводить сообщение о правильности введенного адреса.
8. Создайте программу для расчета факториала числа с использованием формы. На странице должно быть поле для ввода числа и кнопка для расчета факториала. После нажатия кнопки программа должна выводить результат на страницу.
9. Разработайте программу для конвертации валюты с использованием формы. На странице должны быть поля для ввода суммы в исходной валюте, выбор валюты для конвертации и кнопка для расчета. После нажатия кнопки программа должна выводить результат на страницу.
10. Напишите программу для выбора любимого цвета с использованием формы. На странице должен быть элемент для выбора цвета и кнопка для подтверждения выбора. После нажатия кнопки программа должна выводить выбранный цвет на страницу.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие элементы HTML используются для создания формы, и какова структура элемента `<form>`?
2. Как можно получить ссылку на форму в JavaScript?
3. Какие свойства доступны у объекта формы в JavaScript, и как они могут быть получены?
4. Какие методы доступны у объекта формы в JavaScript, и для чего они используются?
5. Какие элементы HTML могут находиться внутри формы, и какие типы JavaScript соответствуют каждому из них?
6. Как можно получить ссылку на элемент формы в JavaScript с использованием методов поиска?
7. Какие свойства доступны у объекта элемента формы в JavaScript, и для чего они используются?

8. Как можно получить или изменить значение поля формы с помощью JavaScript?
9. Какие методы можно использовать для отправки данных формы на сервер или для сброса данных полей формы?

ПРАКТИЧЕСКАЯ РАБОТА №10. СОЗДАНИЕ ПРОГРАММ НА СЕРВЕРНОМ ЯЗЫКЕ ПРОГРАММИРОВАНИЯ, СОДЕРЖАЩИХ КОНСТАНТЫ, ПЕРЕМЕННЫЕ, ОПЕРАТОРЫ

Цель работы: овладеть основами программирования на сервере с использованием языка PHP, включая работу с константами, переменными и операторами, а также научиться решать различные задачи с помощью PHP.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Программа или скрипт на PHP, как правило, находится в файле расширением **.php**. Хотя разработчики могут также вставлять код php и в файлы с расширениями **.html/.htm**.

Документ PHP может содержать как разметку html, так и код на языке php. Для перехода от разметки html к коду php используются теги **<?php** и **?>**, между которыми идет код php. Данные теги служат указанием интерпретатору PHP, что их содержимое надо интерпретировать как код php, а не разметку html.

Например, определим папке, где хранятся файлы веб-сайта (исходя из прошлых тем это должна быть папка c:\localhost), файл **hello.php**. Определим в этом файле следующий код:

```
1  <?php
2  echo "Привет мир!";
3  ?>
```

Собственно код PHP состоит из набора инструкций. Здесь использована одна инструкция `echo "Привет мир!";`. Эта инструкция представляет встроенную команду `echo`, которая выводит на веб-страницу некоторое значение. Выводимое значение указывается после команды `echo` - в данном случае это строка "Привет мир!". Каждая отдельная инструкция в PHP завершается точкой с запятой.

Поскольку в данном случае файл "hello.php" находится в корневой папке веб-сервера, то для обращения к этому скрипту в адресной строке браузера надо ввести адрес `http://localhost/hello.php`. В итоге при обращении к этому скрипту мы увидим в веб-браузере следующую страницу:

Для оформления кода РНР также можно использовать краткую версию тегов: `<? и ?>`. Для этого в файле `php.ini` надо изменить строку:

```
1 short_open_tag = Off
    на
1 short_open_tag = On
```

Когда пользователь обращается к скрипту в адресной строке браузера, набирая, например, `http://localhost/hello.php`, то веб-сервер передает его интерпретатору РНР. Затем интерпретатор обрабатывает код и генерирует на его основе html-разметку. И затем сгенерированный html-код отправляется пользователю.

В случае выше определенного скрипта `hello.php` сгенерированная разметка будет выглядеть следующим образом:

```
1 Привет мир!
```

Но, естественно, мы можем также добавить некоторый код html. Например, изменим скрипт `hello.php` следующим образом:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>METANIT.COM</title>
5 <meta charset="utf-8" />
6 </head>
7 <body>
8 <h1> Сайт на РНР</h1>
9 <?php
10 echo "Привет мир!";
11 ?>
12 </body>
13 </html>
```

Как уже указывалось выше, интерпретатор с помощью тегов `<?php ... ?>` сможет понять, что весь текст между этими тегами следует рассматривать как код РНР. А весь код вне этих тегов рассматривается как код html.

В этом случае интерпретатор сформирует следующий html-код:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>METANIT.COM</title>
5 <meta charset="utf-8" />
```

```

6    </head>
7    <body>
8    <h1> Сайт на PHP</h1>
9    Привет мир!
10   </body>
11   </html>

```

При этом естественно мы можем использовать больше инструкций, а также встраивать код php в различных частях веб-страницы. Например:

```

1    <!DOCTYPE html>
2    <html>
3    <head>
4    <title>METANIT.COM</title>
5    <meta charset="utf-8" />
6    </head>
7    <body>
8    <h1>
9    <?php
10   echo "Первый сайт на PHP";
11   ?>
12 </h1>
13 <div>
14 <?php
15 echo "<h2>Заголовок параграфа</h2>";
16 echo "Текст параграфа";
17 ?>
18 </div>
19 </body>
20 </html>

```

В данном случае код PHP встраивается в двух местах. В первом случае - внутри элемента `<h1>`. Во втором случае внутри элемента `<div>`

При чем при использовании функции `echo` мы можем включать в выводимый текст html-код, как в случае с выражением:

```

1    echo "<h2>Заголовок параграфа</h2>";

```

Хотя выше код php определялся в файле с расширением **.php**, но равным образом мы также можем определять код в файлах с расширением **.html**, и они также будут обрабатываться интерпретатором PHP.

Сокращенная версия тегов php

Если нам надо вывести на веб-страницу одно какое-нибудь значение, то мы можем использовать специальную форму тегов php - `<? = ...?>` - после знака `=` ("равно") ставится выводимое выражение. Например:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <h1>
9  <? = "Первый сайт на PHP";?>
10 </h1>
11 <div>
12 2 + 2 = <?= 2+2 ?>
13 </div>
14 </body>
15 </html>

```

В первом случае выводится строка `<?= "Первый сайт на PHP"; ?>`.

Во втором случае выводится результат выражения `2 + 2`: `<?= 2+2 ?>`.

Переменные

Переменные хранят отдельные значения, которые можно использовать в выражениях PHP. Для определения переменных применяется знак доллара `$`. Например:

```
1 $num;
```

Здесь определена переменная `$num`. Поскольку определение переменной — это отдельная инструкция, она завершается точкой с запятой.

Как правило, названия переменных начинаются с маленькой буквы или символа подчеркивания. Стоит учитывать, что PHP является регистрозависимым языком, а значит, переменные `$num` и `$Num` будут представлять две разные переменные.

Также при наименовании переменных нам надо учитывать следующие правила:

- Имена переменных должны начинаться с алфавитного символа или с подчеркивания

- Имена переменных могут содержать только символы: a–z, A–Z, 0–9, и знак подчеркивания
- Имена переменных не должны включать в себя пробелы

С помощью операции присвоения (=) переменной присваивается определенное значение:

```
1 $num = 10;
```

Здесь определена переменная \$num, которая хранит число 10.

После определения переменной и присвоения ей значения мы можем использовать ее в выражениях PHP. Например, вывести ее значение на веб-страницу:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>METANIT.COM</title>
5 <meta charset="utf-8" />
6 </head>
7 <body>
8
9 <?php
10 // определение переменной $num
11 $num = 10;
12 // вывод значения переменной $num на веб-страницу
13 echo $num;
14 ?>
15
16 </body>
17 </html>
```

С помощью функции echo значение переменной \$num выводится на веб-страницу. И при обращении к скрипту мы увидим значение переменной \$num:

Отличительной особенностью переменных является то, что мы можем изменять их значение:

```
1 <?php
2 // определение переменной $num
3 $num = 10;
4 // вывод значения переменной $num на веб-страницу
5 echo $num;
6
7 // меняем значение переменной
8 $num = 22;
9 echo $num;
10 ?>
```

Также можно присваивать значение другой переменной:

```
1 $a = 15;
2 $b = $a;
3 echo $b;
```

Если переменная объявлена, но ей изначально не присвоено никакого значения (не инициализирована), то нам будет проблематично ее использовать. Например:

```
1 <?php
2 $num;
3 echo $num;
4 ?>
```

При попытке вывести значение переменной мы получим диагностическое сообщение о том, что переменная не определена:

```
Warning: Undefined variable $num in C:\localhost\hello.php on line 13
22
```

Поэтому перед использованием переменной ей следует присвоить начальное значение.

Вывод значения переменной

В предыдущих примерах для вывода значения переменной применялась команда **echo**, после которой указывалось выводимое значение. Однако есть и другой способ вывести значение переменной. Например, мы хотим одновременно вывести значения двух переменных:

```
1 <?php
2 $num_1 = 11;
3 $num_2 = 35;
4
5 echo "num_1 = $num_1 num_2=$num_2";
6 ?>
```

Здесь функции echo передается строка. Чтобы встроить в строку значение переменной, в этой строке указываем имя переменной вместе со знаком \$. И когда в строке PHP встретит выражение \$num_1, он заменит это выражение значением переменной \$num_1. То же самое касается и переменной \$num_2. В итоге при выполнении этого скрипта браузер отобразит значения обеих переменных:

```
num_1 = 11 num_2=35
```

Типы данных

PHP является языком с динамической типизацией. Это значит, что тип данных переменной выводится во время выполнения, и в отличие от ряда других языков программирования в PHP не надо указывать перед переменной тип данных.

В PHP есть десять базовых типов данных:

- bool (логический тип)
- int (целые числа)
- float (дробные числа)
- string (строки)
- array (массивы)
- object (объекты)
- callable (функции)
- mixed (любой тип)
- resource (ресурсы)
- null (отсутствие значения)

Из этих типов данных первые четыре являются скалярными: bool, int, float, string.

Операции

Арифметические операции

1. + (операция сложения)

Например, \$a + 5

2. - (операция вычитания)

Например, \$a - 5

3. * (умножение)

Например, \$a * 5

4. / (деление)

Например, \$a / 5

5. % (получение остатка от деления)

Например:

- 1 \$a = 12;
- 2 echo \$a % 5; // равно 2

6. ** (возведение в степень)

Например, \$a ** 2

Примеры операций:

- 1 \$a = 8 + 2; // 10, сложение


```

2  $a = 8 - 2; // 6, вычитание
3  $a = 8 * 2; // 16, умножение
4  $a = 8 / 2; // 4, деление
5  $a = 8 % 2; // 0, деление по модулю
6  $a = 8 ** 2; // 64, возведение в степень

```

Инкремент и декремент

Отдельно следует сказать операции **инкремента** и **декремента**, которые также являются арифметическими операциями, но производятся над одним операндом.

Инкремент - операция ++ увеличивает число на единицу. Например, ++\$a

Есть два типа инкремента: префиксный инкремент (++\$a) и постфиксный (\$a++). Важно понимать разницу между этими операциями. Рассмотрим сначала префиксный инкремент:

```

1  $a = 12;
2  $b = ++$a; // $b равно 13
3  echo "a = $a b = $b";

```

Результат работы:

a = 13 b = 13

Здесь сначала к значению переменной \$a прибавляется единица, а затем ее значение приравнивается переменной \$b.

Теперь посмотрим, что будет в случае с постфиксным инкрементом:

```

1  $a = 12;
2  $b = $a++; // $b равно 12
3  echo "a = $a b = $b";

```

Результат работы:

a = 13 b = 12

Здесь сначала значение переменной \$a передается переменной \$b, а затем происходило увеличение значения переменной \$a.

Декремент - операция -- представляет уменьшение значения на единицу. Она аналогично инкременту бывает префиксной и постфиксной и работает аналогично. Например, префиксный декремент:

```

1  $a = 12;
2  $b = --$a; // $b равно 11
3  echo "a = $a b = $b";

```

Результат работы:

```
a = 11 b = 11
```

Сначала значение переменной \$a уменьшается на единицу, а затем ее значение приравнивается переменной \$b.

Постфиксный декрементом:

```
1 $a = 12;
2 $b = $a--; // $b равно 12
3 echo "a = $a b = $b";
```

Результат работы:

```
a = 11 b = 12
```

Здесь же сначала значение переменной \$a передается переменной \$b и только после этого уменьшается на единицу.

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №10

1. Напишите программу для вычисления площади прямоугольника на основе заданных значений длины и ширины, используя переменные для хранения значений и операторы для вычисления площади.
2. Разработайте программу для проверки является ли введенное пользователем число четным или нечетным, используя операторы для проверки остатка от деления и условные операторы для вывода результата.
3. Создайте программу для определения наибольшего из двух чисел, введенных пользователем, используя переменные для хранения значений и операторы для сравнения чисел.
4. Напишите программу для вывода всех чисел от 1 до 100, кратных 3, используя операторы для проверки остатка от деления и циклы для перебора чисел.
5. Разработайте программу для проверки является ли введенное пользователем число простым, используя операторы для проверки делителей числа и условные операторы для вывода результата.
6. Создайте программу для вычисления факториала числа, введенного пользователем, используя циклы и операторы для вычисления факториала.

7. Напишите программу для вывода таблицы умножения на заданное пользователем число, используя циклы и операторы для умножения чисел.
8. Разработайте программу для нахождения среднего значения списка чисел, введенного пользователем, используя переменные для хранения суммы и количества чисел, и операторы для вычисления среднего.
9. Создайте программу для проверки является ли введенная пользователем строка палиндромом, используя операторы для переворачивания строки и условные операторы для проверки совпадения.
10. Напишите программу для нахождения наименьшего делителя числа, введенного пользователем, используя операторы для проверки делителей числа и циклы для перебора возможных делителей.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие языки программирования могут использоваться для создания программ на сервере?
2. В чем заключается основная разница между кодом PHP и HTML?
3. Какие теги используются для встраивания кода PHP в HTML-документ?
4. Какой синтаксис используется для определения переменных в PHP?
5. Какие символы могут содержать имена переменных в PHP?
6. Как можно присвоить значение переменной в PHP?
7. Как можно вывести значение переменной на веб-страницу с помощью PHP?
8. Каким образом можно изменить значение переменной в PHP?
9. Какие операторы используются в PHP для выполнения различных операций?
10. Каким образом можно использовать краткую форму тегов PHP для вывода значений на веб-страницу?

ПРАКТИЧЕСКАЯ РАБОТА №11. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ВЕТВЛЕНИЙ, ЦИКЛОВ И МАССИВОВ

Цель работы: овладеть навыками программирования на РНР с использованием ветвлений (условных операторов), циклов и массивов для решения различных задач.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Условные конструкции позволяют направлять работу программы в зависимости от условия по одному из возможных путей. И одной из таких конструкций в языке РНР является конструкция `if..else`

Конструкция `if..else`

Конструкция `if` (условие) проверяет истинность некоторого условия, и если оно окажется истинным, то выполняется блок выражений, стоящих после `if`. Если же условие ложно, то есть равно `false`, тогда блок `if` не выполняется. Например:

```
1  <?php
2  $a = 4;
3  if($a>0) {
4      echo "Переменная а больше нуля";
5  }
6  echo "<br>конец выполнения программы";
7  ?>
```

Блок выражений ограничивается фигурными скобками. И так как в данном случае условие истинно (то есть равно `true`): значение переменной `$a` больше 0, то блок инструкций в фигурных скобках также будет выполняться. Если бы значение `$a` было бы меньше 0, то блок `if` не выполнялся.

Если блок `if` содержит всего одну инструкцию, то можно опустить фигурные скобки:

```
1  <?php
2  $a = 4;
3  if($a>0)
4      echo "Переменная а больше нуля";
5  echo "<br>конец выполнения программы";
6  ?>
```

Можно в одной строке поместить всю конструкцию:

```
1  if($a>0) echo "Переменная а больше нуля";
```

В данном случае к блоку `if` относится только инструкция `echo "Переменная а больше нуля";`

else

Блок `else` содержит инструкции, которые выполняются, если условие после `if` ложно, то есть равно `false`:

```
1  <?php
2  $a = 4;
3  if ($a > 0) {
4      echo "Переменная а больше нуля";
5  }
6  else {
7      echo "Переменная а меньше нуля";
8  }
9  echo "<br>конец выполнения программы";
10 ?>
```

Если `$a` больше 0, то выполняется блок `if`, если нет, то выполняется блок `else`.

Поскольку здесь в обоих блоках по одной инструкции, также можно было не использовать фигурные скобки для определения блоков:

```
1  if ($a > 0)
2      echo "Переменная а больше нуля";
3  else
4      echo "Переменная а меньше нуля";
```

elseif

Конструкция `elseif` вводит дополнительные условия в программу:

```
1  $a = 5;
2  if($a>0) {
3      echo "Переменная а больше нуля";
4  }
5  elseif ($a < 0) {
6      echo "Переменная а меньше нуля";
7  }
8  else {
9      echo "Переменная а равна нулю";
10 }
```

Можно добавить множество блоков `elseif`. И если ни одно из условий в `if` или `elseif` не выполняется, тогда срабатывает блок `else`.

Определение условия

Выше в качестве условия применялись операции сравнения. Однако в реальности в качестве условия может применяться любое выражение, а не только такое, которое возвращает true или false. Если передаваемое выражение равно 0, то оно интерпретируется как значение false. Другие значения рассматриваются как true:

```
1  if (0) {} // false
2  if (-0.0) {} // false
3  if (-1) {} // true
4  if ("") {} // false (пустая строка)
5  if ("a") {} // true (непустая строка)
6  if (null) {} // false (значение отсутствует)
```

Альтернативный синтаксис if

PHP также поддерживает альтернативный синтаксис для конструкции if..else, при которой вместо открывающей фигурной скобки ставится двоеточие, а в конце всей конструкции ставится ключевое слово endif.

```
1  $a = 5;
2  if ($a > 0):
3      echo "Переменная а больше нуля";
4  elseif ($a < 0):
5      echo "Переменная а меньше нуля";
6  else:
7      echo "Переменная а равна нулю";
8  endif;
```

Комбинированный режим HTML и PHP

Также мы можем написать конструкцию if...else иным образом, переключаясь внутри конструкции на код HTML:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <?php
9  $a = 5;
10 ?>
11
12 <?php if ($a > 0) { ?>
13 <h2>Переменная а больше нуля</h2>
14 <?php } ?>
15
```

```
16 </body>
17 </html>
```

В данном случае само условие указывается в отдельном блоке php: `<?php if ($a > 0) { ?>`. Важно, что при этом этот блок содержит только открывающую фигурную скобку `"{"`.

Завершается конструкция `if` другим блоком php, который содержит закрывающую фигурную скобку: `<?php } ?>`

Между этими двумя блоками php располагается код, который отображается на html-странице, если условие в `if` истинно. Причем этот код представляет именно код html, поэтому здесь можно разместить различные элементы html, как в данном случае элемент `<h2>`

При необходимости можно добавить выражения `else` и `elseif`:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>METANIT.COM</title>
5 <meta charset="utf-8" />
6 </head>
7 <body>
8 <?php
9 $a = -5;
10 ?>
11
12 <?php if ($a > 0) { ?>
13 <h2> Переменная а больше нуля</h2>
14 <?php} elseif ($a < 0) { ?>
15 <h2> Переменная а меньше нуля</h2>
16 <?php} else { ?>
17 <h2> Переменная а равна нулю</h2>
18 <?php} ?>
19 </body>
20 </html>
```

Также можно применять альтернативный синтаксис:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>METANIT.COM</title>
5 <meta charset="utf-8" />
6 </head>
7 <body>
8 <?php
9 $a = 0;
10 ?>
11
12 <?php if ($a > 0) :>
```

```

13 <h2> Переменная а больше нуля</h2>
14 <?php elseif ($a <0):?>
15 <h2> Переменная а меньше нуля</h2>
16 <?php else:?>
17 <h2>Переменная а равна нулю</h2>
18 <?php endif; ?>
19 </body>
20 </html>

```

Тернарная операция

Тернарная операция состоит из трех операндов и имеет следующее определение: [первый операнд - условие] ? [второй операнд] : [третий операнд]. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно true, то возвращается второй операнд; если условие равно false, то третий. Например:

```

1 $a = 1;
2 $b = 2;
3 $z = $a < $b ? $a + $b : $a - $b;
4 echo $z;

```

Если значение переменной \$a меньше \$b и условие истинно, то переменная \$z будет равняться \$a + \$b. Иначе значение \$z будет равняться \$a - \$b

Циклы

Для совершения повторяемых действий в PHP, как и в других языках программирования, используются циклы. В PHP имеются следующие виды циклов:

- for
- while
- do..while

Цикл for

Цикл for имеет следующее формальное определение:

```

1 for ([инициализация счетчика]; [условие]; [изменение счетчика])
2 {
3     // действия
4 }

```

Рассмотрим стандартный цикл for:

```

1 <?php
2 for ($i = 1; $i < 10; $i++)
3 {

```



```

4     echo "Квадрат числа $i равен " . $i * $i . "<br/>";
5 }
6 ?>

```

Первая часть объявления цикла - $i = 1$ - создает и инициализирует счетчик - переменную i . И перед выполнением цикла его значение будет равно 1. По сути, это то же самое, что и объявление переменной.

Вторая часть - $i < 10$; - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока i не достигнет 10.

Третья часть - $i++$ - изменение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: $i--$. Или, например, увеличить не на 1, на 3: $i += 3$.

В итоге блок цикла сработает 9 раз, пока значение i не станет равным 10. И каждый раз это значение будет увеличиваться на 1. Каждое отдельное повторение цикла называется итерацией. Таким образом, в данном случае произойдет 9 итераций.

В итоге браузер отобразит нам следующий результат:

```

Квадрат числа 1 равен 1
Квадрат числа 2 равен 4
Квадрат числа 3 равен 9
Квадрат числа 4 равен 16
Квадрат числа 5 равен 25
Квадрат числа 6 равен 36
Квадрат числа 7 равен 49
Квадрат числа 8 равен 64
Квадрат числа 9 равен 81

```

Объявление цикла `for` может опускать отдельные части. Например, опустить определение счетчика (он может быть определен вне цикла):

```

1  $i = 5;
2  for (; $i < 10; $i++)
3  {
4      echo $i;
5  }

```

Можно опустить изменение значения счетчика и изменять его внутри цикла:

```

1  $i = 0;
2  for (; $i < 10;)
3  {

```

```

4     echo $i;
5     $i += 2;
6 }

```

В данном случае в цикле на каждой итерации переменная `$i` увеличивает значение на 2. Соответственно мы получим следующий результат:

```
02468
```

Также можно в объявлении цикла определять и использовать сразу несколько переменных:

```

1  for ($i = 1, $j = 1; $i + $j < 10; $i++, $j += 2)
2  {
3      echo "$i + $j = " . $i + $j . "<br>";
4  }

```

В данном случае в объявлении цикла определяются две переменных: `$i` и `$j`. При каждой итерации переменная `$i` увеличивается на 1, а `$j` - на 2. При этом цикл продолжается, пока сумма двух переменных не достигнет 10:

```

1 + 1 = 2
2 + 3 = 5
3 + 5 = 8

```

Также можно применять альтернативный синтаксис, при котором вместо открывающей фигурной скобки ставится двоеточие, а вместо закрывающей фигурной скобки - ключевое слово `endfor`:

```

1  for ($i = 1; $i < 10; $i++):
2      echo "Квадрат числа $i равен " . $i * $i . "<br/>";
3  endfor;

```

Цикл while

Цикл `while` проверяет истинность некоторого условия, и если условие истинно, то выполняются блок выражений цикла:

```

1  <?php
2  $counter = 1;
3  while($counter < 10)
4  {
5      echo $counter * $counter . "<br />";
6      $counter++;
7  }
8  ?>

```

Если в блоке `while` всего одна инструкция, то фигурные скобки блока можно опустить:

```

1  <?php
2  $counter = 0;
3  while(++$counter < 10)
4      echo $counter * $counter . "<br />";

```

5 ?>

Также можно применять альтернативный синтаксис, при котором вместо открывающей фигурной скобки ставится двоеточие, а вместо закрывающей фигурной скобки - ключевое слово **endwhile**:

```
1 $counter = 1;
2 while($counter<10):
3     echo $counter * $counter . "<br />";
4     $counter++;
5 endwhile;
```

Цикл **do...while**

Цикл **do...while** похож на цикл **while**, только теперь выполняется блок цикла, и только потом выполняется проверка условия. То есть даже если условие ложно, то блок цикла выполнится как минимум один раз:

```
1 <?php
2 $counter = 1;
3 do
4 {
5     echo $counter * $counter . "<br />";
6     $counter++;
7 }
8 while($counter<10)
9 ?>
```

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №11

1. Напишите программу, которая определяет, является ли введенное пользователем число четным или нечетным, используя ветвление (**if-else**).
2. Разработайте программу для определения наибольшего числа из трех введенных пользователем чисел, используя ветвление (**if-else**).
3. Создайте программу для вывода всех чисел от 1 до 100, которые делятся на 7, используя цикл (**for** или **while**).
4. Напишите программу, которая проверяет, является ли введенная пользователем строка палиндромом, используя ветвление (**if-else**) и циклы.
5. Разработайте программу для вычисления факториала числа, введенного пользователем, используя цикл (**for** или **while**).

6. Создайте программу для суммирования всех чисел введенного пользователем массива, используя цикл (for или foreach).
7. Напишите программу для нахождения среднего значения массива чисел, введенного пользователем, используя цикл (for или foreach).
8. Разработайте программу для вывода таблицы умножения от 1 до 10, используя вложенные циклы (for или while).
9. Создайте программу для проверки является ли введенное пользователем число простым, используя ветвление (if-else) и циклы.
10. Напишите программу, которая находит сумму всех четных чисел в заданном пользователем диапазоне, используя ветвление (if-else) и циклы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие конструкции используются в PHP для направления выполнения программы в зависимости от условий?
2. Каков синтаксис конструкции if..else в PHP?
3. Какие альтернативные формы синтаксиса конструкции if..else поддерживает PHP?
4. В чем разница между конструкциями if..else и switch в PHP?
5. Какие значения рассматриваются как true и false в условиях в PHP?
6. Как можно использовать конструкцию if..else в комбинации с HTML в PHP?
7. Какие операторы сравнения используются в условиях в PHP?
8. Что такое конструкция elseif, и как она используется?
9. Какие операции можно выполнять внутри условий в PHP?
10. В каких случаях предпочтительнее использовать альтернативный синтаксис для конструкции if..else в PHP?

ПРАКТИЧЕСКАЯ РАБОТА №12. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ И ФУНКЦИЙ

Цель работы: овладеть навыками работы с регулярными выражениями и функциями в PHP для обработки и анализа текстовой информации.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Функции представляют собой блок инструкций, которые многократно можно вызывать в различных частях программы. Функции позволяют разделять программу на меньшие функциональные части.

Общий синтаксис определения функции выглядит следующим образом:

```
1  function имя_функции ([параметр [, ...]])
2  {
3      // Инструкции
4  }
```

Определение функции начинается с ключевого слова `function`, за которым следует имя функции. Имя функции должно начинаться с алфавитного символа или подчеркивания, за которыми может следовать любое количество алфавитно-цифровых символов или символов подчеркивания.

После имени функции в скобках идет перечисление параметров. Даже если параметров у функции нет, то просто идут пустые скобки. Затем в фигурных скобках идет тело функции, содержащее набор инструкций.

Определим простейшую функцию:

```
1  <?php
2  function hello ()
3  {
4      echo "Hello PHP";
5  }
6  ?>
```

Данная функция называется `hello`. Она не имеет параметров, поэтому после названия функции идут пустые скобки. Блок функции содержит только одну инструкцию, которая выводит сообщение "Hello PHP".

Чтобы функция сработала, ее надо вызвать. Для вызова функции указывается ее имя, после которого в скобках идут значения для ее параметров (если, конечно, она имеет параметры):

1 название_функции (значения_для_параметров_функции);

Например, вызовем вышеопределенную функцию hello:

```
1  <?php
2  function hello ()
3  {
4      echo "Hello PHP";
5  }
6
7  hello (); // вызов функции
8  ?>
```

Поскольку для функции hello мы не определили никаких параметров, то при ее вызове указываем название функции и после нее пустые круглые скобки. Заканчивается вызов функции точкой с запятой.

В итоге браузер выведет сообщение:

Hello PHP

Преимуществом функций является то, что, определив однажды, мы можем многократно их вызывать в различных частях программы:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <?php
9  function hello ()
10 {
11     echo "<h2>Hello PHP</h2>";
12 }
13 hello ();
14 hello ();
15 hello ();
16 ?>
17 </body>
18 </html>
```

Стоит отметить, что в данном случае сначала определяется функция, а затем она вызывается. Но мы также можно сделать наоборот:

```
1  <?php
2  hello (); // вызов функции
3
4  function hello ()
5  {
6      echo "Hello PHP";
7  }
8  ?>
```

Тем не менее есть исключения. В частности, при определении функции при определенном условии.

```
1  // hello (); // здесь будет ошибка
2
3  if(true) {
4      function hello ()
5      {
6          echo "Hello PHP";
7      }
8
9      hello ();
10 }
```

Параметры функции

С помощью параметров мы можем передавать в функцию некоторые данные. Параметры определяются в скобках после названия функции как обычные переменные, отделенные друг от друга запятой.

Например, создадим и вызовем функцию с одним параметром:

```
1  <?php
2  function hello($name)
3  {
4      echo "<h2>Hello $name</h2>";
5  }
6
7  hello("Tom");
8  hello("Bob");
9  hello("Sam");
10 ?>
```

Здесь функция hello определяет один параметр - \$name. При наименовании параметров применяются те же правила, что и для переменных. Также название параметров начинается со знака

доллара \$. Единственное, что не нужно указывать значение для параметра.

```
1 function hello($name)
```

Внутри самой функции мы можем использовать параметр так же, как обычные переменные. Например, в данном случае его значение выводится на веб-страницу:

```
1 echo "<h2>Hello $name</h2>";
```

В дальнейшем при вызове функции нам надо передать для параметра некоторое значение. Значения при вызове функции передаются в скобках, сколько функция определяет параметров, столько необходимо передать значений.

Так, в данном случае функция определяет один параметр, соответственно при вызове функции передается только одно значение. Однако при каждом вызове это может быть разное значение:

```
1 hello("Tom");
2 hello("Bob");
3 hello("Sam");
```

Результат работы программы (рис. 12.1):

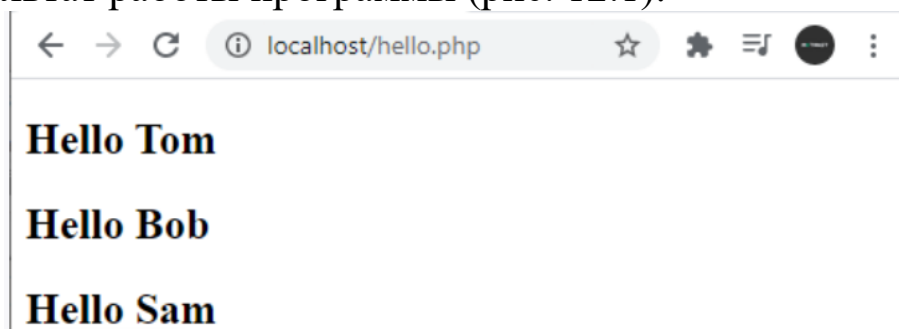


Рисунок 12.1 – Результат работы программы

Если мы не передадим значение для параметра: `hello ()`; то мы столкнемся с ошибкой.

В качестве значения в функцию может передаваться и значение переменной:

```
1 $userName = "Tom";
2 hello($userName);
```

Подобным образом можно определять функции и с большим количеством параметров:

```
1 <?php
2 function displayInfo($name, $age)
3 {
```



```

4     echo "<div>Имя: $name <br />Возраст: $age</div><hr>";
5 }
6
7     displayInfo("Tom", 36);
8     displayInfo("Bob", 39);
9     displayInfo("Sam", 28);
10  ?>

```

Здесь функция `displayInfo` определяет два параметра, соответственно при вызове функции нам надо передать в функцию два значения. Значения отделяются запятой и передаются параметрам по позиции. Так, первое значение передается первому параметру, второе значение передается второму параметру и так далее. В итоге мы получим следующий результат (рис. 12.2):

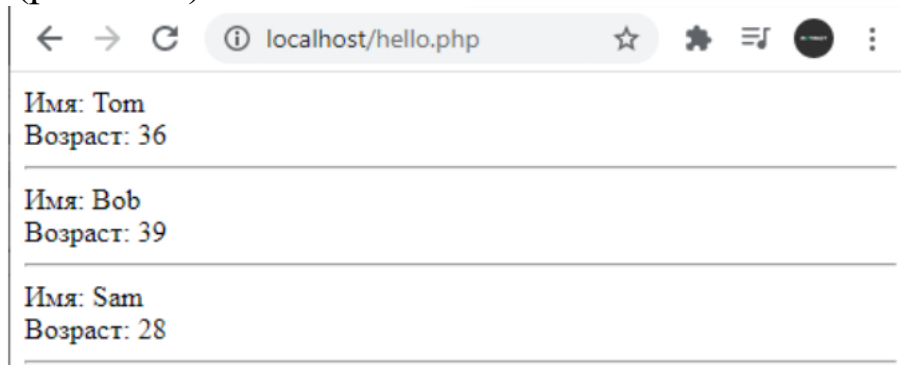


Рисунок 12.2 – Результат работы программы

Необязательные параметры

Выше при определении функции мы были обязаны передать для всех параметров функции значения. Например, если функция определяет два параметра, соответственно нам надо передать в ее вызов два значения. Однако, РНР позволяет сделать параметры необязательными. Такие параметры имеют значение по умолчанию, которое применяется, если при вызове функции не передано никакого значения.

```

1  <?php
2  function displayInfo($name, $age = 18)
3  {
4      echo "<div>Имя: $name <br />Возраст: $age</div><hr>";
5  }
6
7  displayInfo("Tom", 36);
8  displayInfo("Sam");
9  ?>

```

Здесь параметр \$age определяется как необязательный. Для этого ему присваивается начальное значение - число 18. Причем если функция содержит обязательные и необязательные параметры, то необязательные параметры должны определяться в конце (как в данном случае параметр \$age).

При первом вызове в функцию передаются два значения: `displayInfo("Tom", 36)`, поэтому параметр \$age получит второе значение - число 36.

Во втором вызове в функцию передается одно значение: `displayInfo("Sam")`, поэтому параметр \$age будет использовать значение по умолчанию - число 18.

Имя: Tom
Возраст: 36

Имя: Sam
Возраст: 18

Именованные параметры

Начиная с версии 8.0 в PHP была добавлена поддержка именованных параметров. Так, до PHP 8.0 при вызове функции значения можно было передать параметрам только по позиции. Именованные параметры позволяют передавать значения параметрам по имени:

```

1  <?php
2  function displayInfo($name, $age = 18)
3  {
4      echo "<div>Имя: $name <br />Возраст: $age</div><hr>";
5  }
6
7  displayInfo(age: 23, name: "Bob");
8  displayInfo(name: "Tom", age: 36);
9  displayInfo(name: "Alice");
10 ?>
```

При вызове функции сначала указывается название параметра (без символа \$) и через двоеточие указывается значение параметра: `age: 23, name: "Bob"`. И в этом случае нам необязательно соблюдать позицию параметров.

Можно сочетать передачу значений параметрам по имени и по позиции. При этом любые именованные необязательные

параметры должны располагаться после Неименованных параметров:

```
1 displayInfo("Tom", age: 36);
```

Переменное количество параметров

В PHP функция может принимать переменное количество параметров. Для этого у функции определяется один параметр, перед которым указывается оператор ... (три точки). Такой параметр рассматривается как массив:

```
1 <?php
2 function sum(...$numbers)
3 {
4     $result = 0;
5     foreach($numbers as $number) {
6         $result += $number;
7     }
8     echo "<p>Сумма: $result</p>";
9 }
10 sum(1, 2, 3);
11 sum(2, 3);
12 sum(4, 5, 8, 10);
13 ?>
```

При обращении к подобной функции мы можем передавать в нее различное количество значений. Результат:

Сумма: 6

Сумма: 5

Сумма: 27

Но, допустим, готовый массив значений, которые мы хотим передать в функцию. Чтобы его передать в функцию, опять же применяется оператор ..., который указывается перед переменной массива:

```
1 <?php
2 function sum (...$numbers)
3 {
4     $result = 0;
5     foreach ($numbers as $number) {
6         $result += $number;
7     }
8     echo "<p>Сумма: $result</p>";
9 }
10 $numbers = [3, 5, 7, 8];
11 sum (...$numbers); // 23
12 ?>
```

Если функция должна принимать и другие параметры, то параметр, который представляет переменное количество значений, указывается в конце после остальных параметров.

Например, определим функцию, которая принимает имя студента и неопределенное количество его баллов успеваемости и выводит средний балл:

```

1  function getAverageScore($name, ...$scores)
2  {
3      $result = 0.0;
4      foreach($scores as $score) {
5          $result += $score;
6      }
7      $result = $result / count($scores);
8      echo "<p>$name: $result</p>";
9  }
10 getAverageScore("Tom", 5, 5, 4, 5);
11 getAverageScore("Bob", 4, 3, 4, 4, 4);

```

Баллы успеваемости передаются через параметр `$scores`, который указывается в конце списка параметров. В самой функции для вычисления среднего балла все баллы складываются и делятся на их количество. Количество элементов массива можно подсчитать с помощью встроенной в PHP функции `count()`, в которую передается массив.

Анонимные функции

Анонимные функции позволяют передавать в качестве параметров функции другие функции или присваивать их переменным.

Анонимная функция определяется как обычная функция за тем исключением, что она не имеет имени. Например:

```

1  $hello = function($name)
2  {
3      echo "<h2>Hello $name</h2>";
4  };

```

Здесь переменной `$hello` присваивается анонимная функция. Эта функция также определяется с помощью ключевого слова `function`. Она также принимает параметры - в данном случае параметр `$name`. И также она имеет некоторый блок операторов.

Для вызова подобной функции применяется имя представляющей ее переменной:

```

1  $hello("Tom");

```

Фактически подобная переменная применяется как стандартная функция.

Полный код:

```
1 <?php
2 $hello = function($name)
3 {
4     echo "<h2>Hello $name</h2>";
5 };
6 $hello("Tom");
7 $hello("Bob");
8 ?>
```

Также анонимные функции могут возвращать некоторое значение:

```
1 <?php
2 $sum = function($a, $b)
3 {
4     return $a + $b;
5 };
6 $number = $sum(5, 11); //16
7 echo $number;
8 ?>
```

Распространенным случаем применения анонимных функций является передача их параметрам других функций. Таким анонимные функции еще называют функциями обратного вызова или коллбеками (callback function). Рассмотрим простейший пример:

```
1 <?php
2 function welcome($message)
3 {
4     $message();
5 }
6
7 welcome(function()
8 {
9     echo "Hello!";
10 });
11 ?>
```

В данном случае функция `welcome()` имеет параметр `$message`, который внутри функции вызывается подобно функции `$message()`.

При вызове функции `welcome()` параметру `$message` передается анонимная функция, которая выводит строку "Hello!".

В итоге при вызове

```
1 $message();
```

Фактически будет выполняться функция

```
1 function()
2 {
3     echo "Hello!";
4 }
```

Подобным образом мы можем передавать одну функцию различные анонимные функции:

```
1 <?php
2 function welcome($message)
3 {
4     $message();
5 }
6 $goodMorning = function() { echo "<h3>Доброе утро</h3>"; };
7 $goodEvening = function() { echo "<h3>Добрый вечер</h3>"; };
8
9 welcome($goodMorning);      // Доброе утро
10 welcome($goodEvening);     // Добрый вечер
11 welcome(function(){ echo "<h3>Привет</h3>"; }); // Привет
12 ?>
```

В чем польза подобной передачи анонимных функций в качестве параметров? А польза заключается в том, что мы можем определить функцию, но на момент определения функции точно не зная, какие действия она будет выполнять при вызове. Либо просто сделать логику функции более гибкой. Например, нам надо определить функцию, которая выводит сумму элементов массива:

```
1 function sum($numbers)
2 {
3     $result = 0;
4     foreach($numbers as $number){
5
6         $result += $number;
7     }
8     return $result;
9 }
10
11 $myNumbers = [-2, -1, 0, 1, 2, 3, 4, 5];
12 $numbersSum = sum($myNumbers);
13 echo $numbersSum;      // 12
```

Но теперь пойдем дальше: допустим, мы хотим, чтобы функция подсчитывала сумму только тех чисел, которые удовлетворяют определенному условию, например, только положительных чисел или только четных чисел. Но на момент

определения функции мы не знаем, какое условие мы захотим использовать в будущем. И в этом случае мы можем использовать анонимные функции:

```

1  <?php
2  function sum($numbers, $condition)
3  {
4      $result = 0;
5      foreach($numbers as $number){
6          if($condition($number))
7          {
8              $result += $number;
9          }
10     }
11     return $result;
12 }
13
14 // для четных чисел
15 $isEvenNumber = function($n){ return $n % 2 === 0;};
16 // для положительных чисел
17 $isPositiveNumber = function($n){ return $n > 0;};
18
19 $myNumbers = [-2, -1, 0, 1, 2, 3, 4, 5];
20 $positiveSum = sum($myNumbers, $isPositiveNumber);
21 $sevenSum = sum($myNumbers, $isEvenNumber);
22 echo "Сумма положительных чисел: $positiveSum <br /> Сумма четных чисел: $sevenSum";
23 ?>

```

Результат скрипта:

Сумма положительных чисел: 15

Сумма четных чисел: 4

Здесь функция `sum()` в качестве второго параметра - `$condition` принимает другую функцию. В цикле при переборе массива с помощью этой функции мы проверяем, удовлетворяет ли элемент массива условию:

```
1  if($condition($number))
```

Причем сейчас мы не знаем, что это будет за условие, как именно функция `$condition` будет работать. Мы только знаем, что она получает число и возвращает `true`, если число удовлетворяет условию, либо `false` - если не удовлетворяет.

И если только число соответствует условию, тогда прибавляем число к общему результату:

```
1  $result += $number;
```

При вызове функции `sum()` в качестве параметра `$condition` передаются конкретные анонимные функции. Например, функция, которая определяет, является ли число четным:

```
1 $isEvenNumber = function($n){ return $n % 2 === 0;};
```

Логика функции проста: если остаток от деления числа на 2 равен 0, то число четное. и функция возвращает true.

Далее мы передаем эту функцию:

```
1 $sevenSum = sum($myNumbers, $isEvenNumber);
```

В итоге при вызове `if($condition($number))` фактически будет выполняться функция `$isEvenNumber()`.

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №12

1. Напишите программу, которая проверяет, содержит ли введенная пользователем строка только буквы латинского алфавита, используя регулярное выражение и соответствующую функцию PHP.
2. Разработайте программу для поиска всех слов, начинающихся с определенной буквы в заданном пользователем тексте, с использованием регулярных выражений и функций PHP.
3. Создайте программу для проверки правильности введенного email адреса, используя регулярное выражение и функцию PHP для проверки формата email.
4. Напишите программу для удаления всех цифр из введенной пользователем строки, используя регулярное выражение и функцию PHP для замены символов.
5. Разработайте программу для определения количества слов во введенной пользователем строке, используя регулярные выражения и функции PHP для разделения строки на слова.
6. Создайте программу для проверки является ли введенная пользователем строка палиндромом, используя регулярные выражения и функции PHP для обработки строки.
7. Напишите программу для проверки правильности введенного URL адреса, используя регулярные выражения и функции PHP для проверки формата URL.
8. Разработайте программу для извлечения всех чисел из введенной пользователем строки, используя регулярные выражения и функции PHP для поиска чисел.
9. Создайте программу для замены всех гласных букв во введенной пользователем строке на символ "*", используя

регулярные выражения и функции РНР для замены символов.

10. Напишите программу для проверки корректности введенного пароля, используя регулярные выражения и функции РНР для проверки соответствия пароля определенным правилам (например, минимальная длина, наличие цифр и специальных символов).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие основные цели применения функций в РНР?
2. Как выглядит общий синтаксис определения функции в РНР?
3. Какие элементы включает в себя определение функции в РНР?
4. Как вызвать определенную функцию в РНР?
5. Что такое параметры функции, и как они определяются?
6. Может ли функция в РНР иметь несколько параметров? Если да, то как их разделять?
7. Можно ли вызывать функцию до ее определения в РНР? Почему?
8. Какие преимущества предоставляют функции в РНР?
9. Как можно использовать функции для повторного использования кода?
10. Как передать значения параметров в функцию при ее вызове?

ПРАКТИЧЕСКАЯ РАБОТА №13. ОБРАБОТКА ДАННЫХ ФОРМЫ

Цель работы: овладеть навыками обработки данных, отправленных через HTML-форму, с использованием PHP для дальнейшей обработки, валидации и сохранения информации.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Отправка форм

Одним из основных способов передачи данных веб-сайту является обработка форм. Формы представляют специальные элементы разметки HTML, которые содержат в себе различные элементы ввода - текстовые поля, кнопки и т.д. И с помощью данных форм мы можем ввести некоторые данные и отправить их на сервер. А сервер уже обрабатывает эти данные.

Создание форм состоит из следующих аспектов:

- Создание элемента `<form></form>` в разметке HTML
- Добавление в этот элемент одно или несколько поле ввода
- Установка метода передачи данных. Чаще всего используются методы GET или POST
- Установка адреса, на который будут отправляться введенные данные

POST-запросы

Создадим новую форму. Для этого определим новый файл `form.php`, в которое поместим следующее содержимое:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <h3>Форма ввода данных</h3>
9  <form action="user.php" method="POST">
10   <p>Имя: <input type="text" name="name" /></p>
11   <p>Возраст: <input type="number" name="age" /></p>
12   <input type="submit" value="Отправить">
13 </form>
14 </body>
15 </html>
```

Атрибут `action="user.php"` элемента `form` указывает, что данные формы будет обрабатывать скрипт `user.php`, который будет находиться с файлом `form.php` в одной папке. А атрибут `method="POST"` указывает, что в качестве метода передачи данных будет применяться метод `POST`.

Теперь определим файл `user.php`, который будет иметь следующее содержание:

```

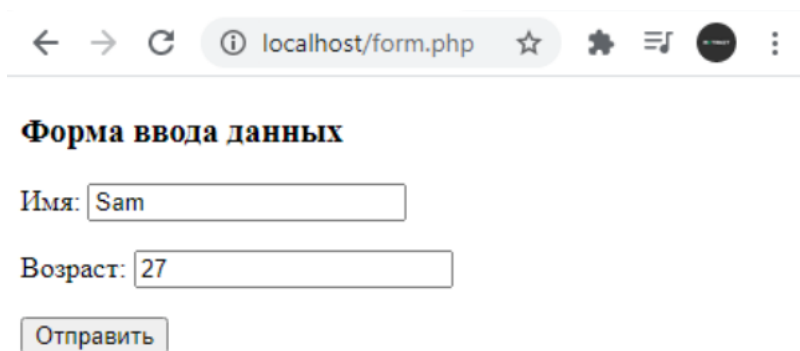
1  <?php
2  $name = "не определено";
3  $age = "не определен";
4  if(isset($_POST["name"])){
5
6      $name = $_POST["name"];
7  }
8  if(isset($_POST["age"])){
9
10     $age = $_POST["age"];
11 }
12 echo "Имя: $name <br> Возраст: $age";
13 ?>
```

Для обработки запросов типа `POST` в `PHP` используется встроенная глобальная переменная `$_POST`. Она представляет ассоциативный массив данных, переданных с помощью метода `POST`. Используя ключи, мы можем получить отправленные значения. Ключами в этом массиве являются значения атрибутов `name` у полей ввода формы.

Например, так как атрибут `name` поля ввода возраста имеет значение `age` (`<input type="number" name="age" />`), то в массиве `$_POST` значение этого поля будет представлять ключ `"age"`: `$_POST["age"]`

И поскольку возможны ситуации, когда поле ввода будет не установлено, то в этом случае желательно перед обработкой данных проверять их наличие с помощью функции `isset()`. И если переменная установлена, то функция `isset()` возвратит значение `true`.

Теперь мы можем обратиться к скрипту `form.php` и ввести в форму какие-нибудь данные (рис.13.1):



← → ↻ ⓘ localhost/form.php ☆ ⚙ ⌵ ⌵

Форма ввода данных

Имя:

Возраст:

Рисунок 13.1 – Форма ввода

И по нажатию кнопки введенные данные методом POST будут отправлены скрипту user.php.

Необязательно отправлять данные формы другому скрипту, можно данные формы обработать в том же файле формы. Для этого изменим файл form.php следующим образом:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <?php
9  $name = "не определено";
10 $age = "не определен";
11 if(isset($_POST["name"])){
12
13     $name = $_POST["name"];
14 }
15 if(isset($_POST["age"])){
16
17     $age = $_POST["age"];
18 }
19 echo "Имя: $name <br> Возраст: $age";
20 ?>
21 <h3>Форма ввода данных</h3>
22 <form method="POST">
23     <p>Имя: <input type="text" name="name" /></p>
24     <p>Возраст: <input type="number" name="age" /></p>
25     <input type="submit" value="Отправить">
26 </form>
27 </body>
28 </html>

```

Поскольку в данном случае мы отправляем данные этому же скрипту - то есть по тому же адресу, то у элемента форма можно не устанавливать атрибут action (рис. 13.2).

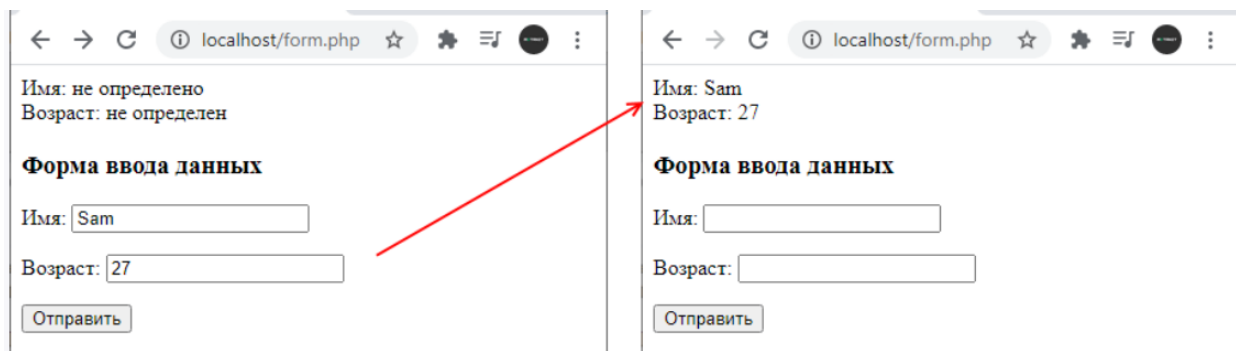


Рисунок 13.2 – Форма ввода данных

Стоит отметить, что в принципе мы можем отправлять формы и запросом GET, в этом случае для получения тех же значений формы применяется массив \$_GET:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <?php
9  $name = "не определено";
10 $age = "не определен";
11 if(isset($_GET["name"])){
12
13     $name = $_GET["name"];
14 }
15 if(isset($_GET["age"])){
16
17     $age = $_GET["age"];
18 }
19 echo "Имя: $name <br> Возраст: $age";
20 ?>
21 <h3>Форма ввода данных</h3>
22 <form method="GET">
23     <p>Имя: <input type="text" name="name" /></p>
24     <p>Возраст: <input type="number" name="age" /></p>
25     <input type="submit" value="Отправить">
26 </form>
27 </body>
28 </html>

```

Работа с полями ввода форм

Формы могут содержать различные элементы - текстовые поля, флажки, переключатели и т.д., обработка которых имеет свои особенности.

Обработка флажков

Флажки или чекбоксы (html-элемент `<input type="checkbox"/>`) могут находиться в двух состояниях: отмеченном (checked) и неотмеченном. Например:

```
1  Запомнить: <input type="checkbox" name="remember" checked="checked" />
```

Если флажок находится в неотмеченном состоянии, например:

```
1  Запомнить: <input type="checkbox" name="remember" />
```

то при отправке формы значение данного флажка не передается на сервер.

Если флажок отмечен, то при отправке на сервер для поля remember будет передано значение on:

```
1  $remember = $_POST["remember"];
```

Если нас не устраивает значение on, то с помощью атрибута value мы можем установить нужное нам значение:

```
1  Запомнить: <input type="checkbox" name="remember" value="1" />
```

Иногда необходимо создать набор чекбоксов, где можно выбрать несколько значений. Например:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <?php
9  if(isset($_POST["technologies"])){
10
11      $technologies = $_POST["technologies"];
12      foreach($technologies as $item) echo "$item<br />";
13  }
14  ?>
15  <h3>Форма ввода данных</h3>
16  <form method="POST">
17      <p>ASP.NET: <input type="checkbox" name="technologies[]" value="ASP.NET" /></p>
18      <p>PHP: <input type="checkbox" name="technologies[]" value="PHP" /></p>
19      <p>Node.js: <input type="checkbox" name="technologies[]" value="Node.js" /></p>
20      <input type="submit" value="Отправить">
21  </form>
22  </body>
23  </html>
```

В этом случае значение атрибута name должно иметь квадратные скобки. И тогда после отправки сервер будет получать массив отмеченных значений:

```
1 $technologies = $_POST["technologies"];
2 foreach($technologies as $item) echo "$item<br />";
```

В данном случае переменная \$technologies будет представлять массив, который можно перебрать и выполнять все другие операции с массивами (рис. 13.3).

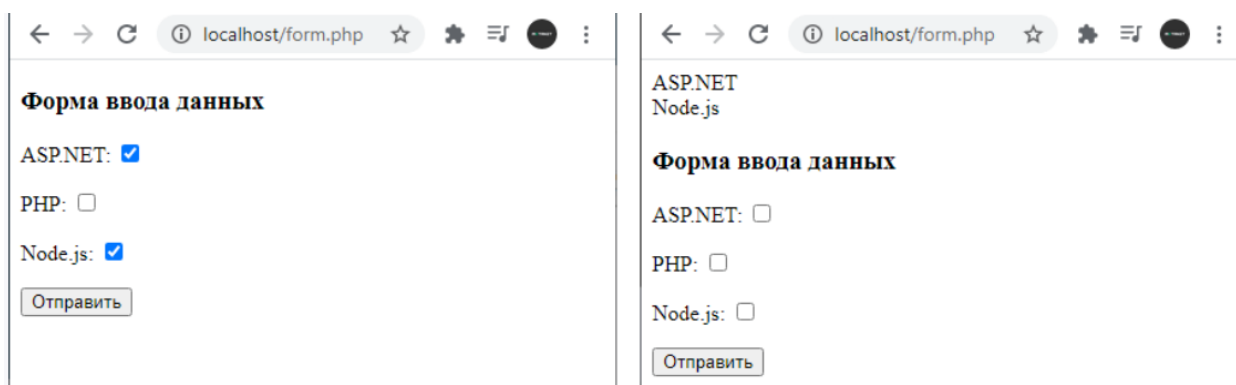


Рисунок 13.3 – Обработка флажков

Переключатели

Переключатели или радиокнопки позволяют сделать выбор между несколькими взаимоисключающими вариантами (рис. 13.4):

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>METANIT.COM</title>
5 <meta charset="utf-8" />
6 </head>
7 <body>
8 <?php
9 if(isset($_POST["course"]))
10 {
11     $course = $_POST["course"];
12     echo $course;
13 }
14 ?>
15 <h3>Форма ввода данных</h3>
16 <form method="POST">
17     <input type="radio" name="course" value="ASP.NET" />ASP.NET <br>
18     <input type="radio" name="course" value="PHP" />PHP <br>
19     <input type="radio" name="course" value="Node.js" />Node.js <br>
20     <input type="submit" value="Отправить">
21 </form>
```

```

22 </body>
23 </html>

```

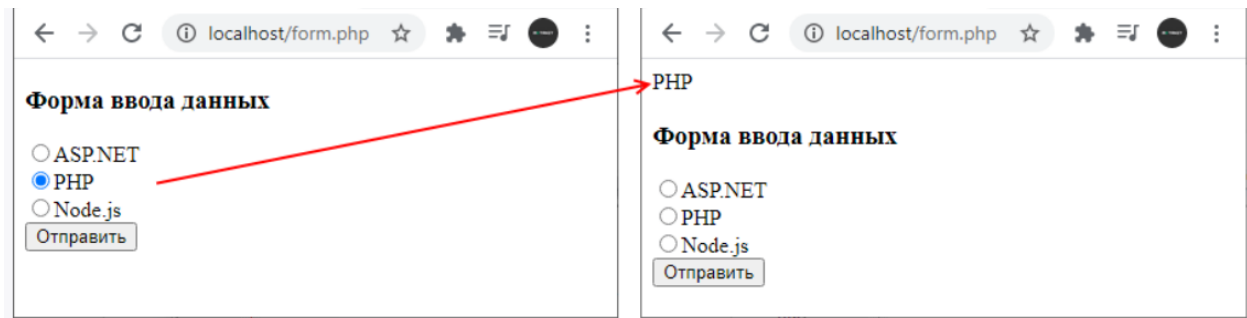


Рисунок 13.4 – Форма с переключателями

На сервер передается значение атрибута value у выбранного переключателя. Получение переданного значения:

```

1  if(isset($_POST["course"]))
2  {
3      $course = $_POST["course"];
4      echo $course;
5  }

```

Список

Список представляет элемент select, который предоставляет выбор одного или нескольких элементов (рис. 13.5):

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <?php
9  if(isset($_POST["course"]))
10 {
11     $course = $_POST["course"];
12     echo $course;
13 }
14 ?>
15 <h3>Форма ввода данных</h3>
16 <form method="POST">
17     <select name="course" size="1">
18         <option value="ASP.NET">ASP.NET</option>
19         <option value="PHP">PHP</option>
20         <option value="Ruby">RUBY</option>
21         <option value="Python">Python</option>
22     </select>
23     <input type="submit" value="Отправить">
24 </form>
25 </body>

```


26 </html>

Элемент `<select>` содержит ряд вариантов выбора в виде элементов `<option>`:

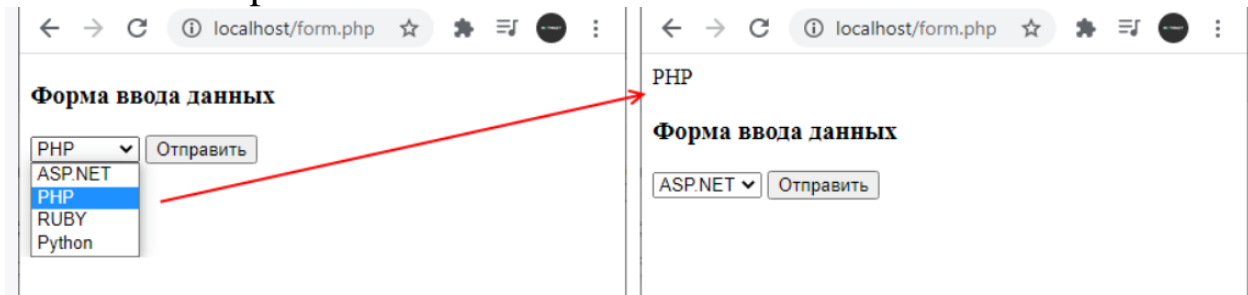


Рисунок 13.5 – Форма со списком

Получить выбранный элемент в коде PHP, как и обычное одиночное значение:

```
1  if(isset($_POST["course"]))
2  {
3      $course = $_POST["course"];
4      echo $course;
5  }
```

Но элемент `<select>` также позволяет сделать множественный выбор. И в этом случае обработка выбранных значений изменяется, так как сервер получает массив значений:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>METANIT.COM</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <?php
9  if(isset($_POST["courses"]))
10 {
11     $courses = $_POST["courses"];
12     foreach ($courses as $item) echo "$item<br>";
13 }
14 ?>
15 <h3> Форма ввода данных</h3>
16 <form method="POST">
17     <select name="courses[]" size="4" multiple="multiple">
18         <option value="ASP.NET">ASP.NET</option>
19         <option value="PHP">PHP</option>
20         <option value="Ruby">RUBY</option>
21         <option value="Python">Python</option>
22     </select><br>
23     <input type="submit" value="Отправить">
```

```

24 </form>
25 </body>
26 </html>

```

Такие списки имеют атрибут `multiple="multiple"`. Для передачи массива также указываются в атрибуте `name` квадратные скобки: `name="courses []"` (рис 13.6).

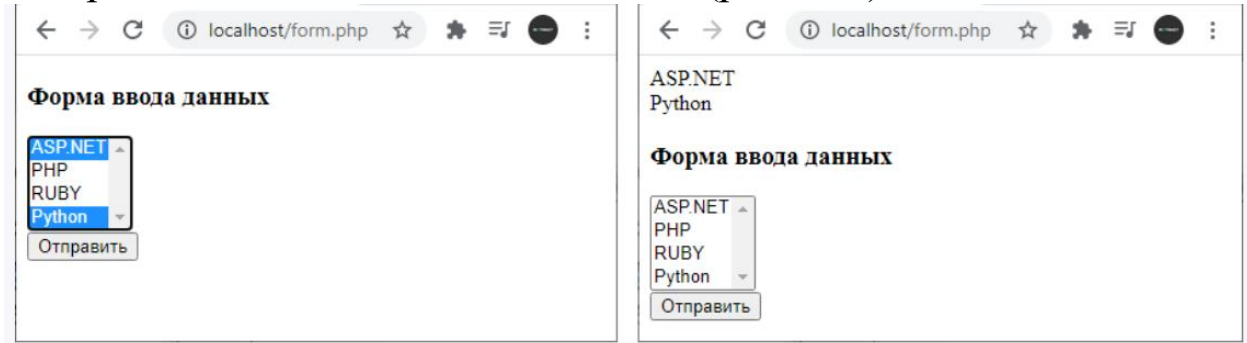


Рисунок 13.6 – Форма с массивом значений

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №13

1. Создайте HTML-форму для ввода имени и email адреса. Форма должна отправлять данные методом POST.
2. Напишите программу на PHP для обработки данных, отправленных из формы. Программа должна проверять наличие и корректность введенных данных (непустое имя, валидный email адрес).
3. Разработайте программу для вывода введенных пользователем данных на экран с помощью PHP после их успешной обработки и валидации.
4. Создайте HTML-форму для ввода пароля. Пароль должен быть скрытым при вводе. Форма должна отправлять данные методом POST.
5. Напишите программу на PHP для обработки введенного пароля. Программа должна проверять длину пароля и его сложность (наличие цифр, букв в разных регистрах, специальных символов).
6. Разработайте программу для вывода сообщения об успешном вводе пароля или сообщения об ошибке при некорректном вводе.
7. Создайте HTML-форму для загрузки изображения на сервер. Форма должна отправлять файл методом POST.

8. Напишите программу на PHP для обработки загруженного изображения. Программа должна проверять тип файла (должен быть изображением), его размер и сохранять на сервере при успешной загрузке.
9. Разработайте программу для вывода предварительного просмотра загруженного изображения на экране с помощью PHP после его успешной обработки.
10. Создайте HTML-форму для ввода текстового сообщения. Форма должна отправлять данные методом POST.
11. Напишите программу на PHP для обработки введенного текстового сообщения. Программа должна удалять все HTML-теги из текста сообщения для предотвращения XSS-атак.
12. Разработайте программу для вывода текстового сообщения без HTML-тегов на экране с помощью PHP после его успешной обработки.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие элементы включает в себя создание формы в HTML?
2. Какие атрибуты используются для определения метода передачи данных формы?
3. Как установить адрес, на который будут отправляться данные из формы?
4. Какие методы передачи данных формы вы знаете? В каких случаях лучше использовать каждый из них?
5. Как обрабатывать данные, отправленные через форму с помощью метода POST в PHP? Как использовать массив \$_POST для получения этих данных?
6. Какие функции PHP можно использовать для проверки наличия отправленных данных из формы?
7. Как можно обработать данные формы в том же файле, в котором определена форма? Как это реализовать?
8. Каким образом можно передавать данные формы с помощью метода GET в PHP? Как получить эти данные с использованием массива \$_GET?
9. Какие ситуации могут потребовать использования метода GET вместо POST для передачи данных формы?

10. Какие атрибуты формы можно опустить, если данные формы отправляются на тот же адрес, откуда форма была вызвана?

ПРАКТИЧЕСКАЯ РАБОТА №14. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ СПЕЦИАЛЬНЫХ ФУНКЦИЙ

Цель работы: овладеть навыками использования специальных функций в РНР для выполнения различных задач, таких как работа с датами, строками, массивами и другими специфическими операциями.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Стрелочные функции

Стрелочные функции (arrow function) позволяют упростить запись анонимных функций, которые возвращают некоторое значение. И при этом стрелочные функции автоматически имеют доступ к переменным из внешнего окружения.

Стрелочная функция определяется с помощью оператора `fn`:

```
1 fn(параметры) => действия;
```

После оператора `fn` в скобках идет список параметров. Затем указывается оператор `=>`. А после него располагаются действия функции, которые возвращают некоторый результат.

Например:

```
1 $a = 8;
2 $b = 10;
3
4 $closure = fn($c) => $a + $b + $c;
5
6 $result = $closure(22); // 40
```

В данном случае определение стрелочной функции

```
1 $closure = fn($c) => $a + $b + $c;
```

Фактически будет аналогично:

```
1 $closure = function($c) use($a, $b)
2 {
3     return $a + $b + $c;
4 };
```

Только в отличие от стандартных анонимных функций стрелочные функции предоставляют более лаконичный синтаксис.

Также стрелочные функции могут применяться в качестве параметров функции:

```

1  function sum($numbers, $condition)
2  {
3      $result = 0;
4      foreach($numbers as $number){
5          if($condition($number))
6              {
7                  $result += $number;
8              }
9      }
10     return $result;
11 }
12
13 $myNumbers = [-2, -1, 0, 1, 2, 3, 4, 5];
14
15 $positiveSum = sum($myNumbers, fn($n)=>$n > 0);
16 $evenSum = sum($myNumbers, fn($n) => $n % 2 === 0);
17 echo "Сумма положительных чисел:
    $positiveSum <br /> Сумма четных чисел: $evenSum";

```

Получение типа переменной

Для получения типа переменной применяется функция `gettype()`, которая возвращает название типа переменной, например, `integer` (целое число), `double` (число с плавающей точкой), `string` (строка), `boolean` (логическое значение), `null`, `array` (массив), `object` (объект) или `unknown type`. Например:

```

1  <?php
2  $a = 10;
3  $b = "10";
4  echo gettype($a); // integer
5  echo "<br>";
6  echo gettype($b); // string
7  ?>

```

Также есть ряд специальных функций, которые возвращают `true` или `false` в зависимости от того, представляет ли переменная определенный тип:

- `is_integer($a)`: возвращает значение `true`, если переменная `$a` хранит целое число
- `is_string($a)`: возвращает значение `true`, если переменная `$a` хранит строку
- `is_double($a)`: возвращает значение `true`, если переменная `$a` хранит действительное число
- `is_numeric($a)`: возвращает значение `true`, если переменная `$a` представляет целое или действительное число или является строковым представлением числа. Например:

```

1  $a = 10;
2  $b = "10";
3  echo is_numeric($a);
4  echo "<br>";
5  echo is_numeric($b);

```

- Оба выражения `is_numeric()` возвратят `true`, так как переменная `$a` представляет число, а переменная `$b` является строковым представлением числа
- `is_bool($a)`: возвращает значение `true`, если переменная `$a` хранит значение `true` или `FALSE`
- `is_scalar($a)`: возвращает значение `true`, если переменная `$a` представляет один из простых типов: строку, целое число, действительное число, логическое значение.
- `is_null($a)`: возвращает значение `true`, если переменная `$a` хранит значение `null`
- `is_array($a)`: возвращает значение `true`, если переменная `$a` является массивом
- `is_object($a)`: возвращает значение `true`, если переменная `$a` содержит ссылку на объект

Установка типа. Функция `settype()`

С помощью функции `settype()` можно установить для переменной определенный тип. Она принимает два параметра: `settype("Переменная", "Тип")`. В качестве первого параметра используется переменная, тип которой надо установить, а в качестве второго - строковое описание типа, которое возвращается функцией `gettype()`.

Если удалось установить тип, то функция возвращает `true`, если нет - то значение `false`.

Например, установим для переменной целочисленный тип:

```

1  <?php
2  $a = 10.7;
3  settype($a, "integer");
4  echo $a; // 10
5  ?>

```

Поскольку переменная `$a` представляет действительное число 10.7, то его вполне можно преобразовать в целое число через отсечение дробной части. Поэтому в данном случае функция `settype()` возвратит `true`.

Преобразование типов

По умолчанию PHP при необходимости автоматически преобразует значение переменной из одного типа в другой. По этой причине явные преобразования в PHP не так часто требуются. Тем не менее мы можем их применять.

Для явного преобразования перед переменной в скобках указывается тип, в который надо выполнить преобразование:

```
1 $boolVar = false;
2 $intVar = (int)$boolVar; // 0
3 echo "boolVar = $boolVar<br>intVar = $intVar";
```

В данном случае значение "false" преобразуется в значение типа int, которое будет храниться в переменной \$intVar. А именно значение false преобразуется в число 0. После этого мы сможем использовать данное значение как число.

При использовании выражения echo для вывода на страницу передаваемые значения автоматически преобразуются в строку. И поскольку переменная boolVar равна false, ее значение будет преобразовано в пустую строку. Тогда как значение 0 преобразуется в строку "0".

В PHP могут применяться следующие преобразования:

- (int), (integer): преобразование в int (в целое число)
- (bool), (boolean): преобразование в bool
- (float), (double), (real): преобразование в float
- (string): преобразование в строку
- (array): преобразование в массив
- (object): преобразование в object

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №14

1. Напишите программу на PHP для определения текущей даты и времени с использованием специальной функции для работы с датами.
2. Разработайте программу для форматирования текущей даты в удобочитаемый формат, например, "год-месяц-день час:минута:секунда".
3. Создайте программу для генерации случайного числа от 1 до 100 с использованием специальной функции PHP для работы с случайными числами.

4. Напишите программу для шифрования пароля с использованием специальной функции РНР для хеширования данных, например, функции `password_hash()`.
5. Разработайте программу для проверки соответствия введенного пользователем пароля хешу, хранящемуся в базе данных, с использованием специальной функции РНР для сравнения хешей, например, функции `password_verify()`.
6. Создайте программу для поиска подстроки в строке с использованием специальной функции РНР для работы со строками, например, функции `strpos()`.
7. Напишите программу для перевода текста в верхний регистр с использованием специальной функции РНР для работы со строками, например, функции `strtoupper()`.
8. Разработайте программу для сортировки массива чисел в порядке убывания с использованием специальной функции РНР для работы с массивами, например, функции `rsort()`.
9. Создайте программу для извлечения части массива, начиная с определенного индекса и до конца массива, с использованием специальной функции РНР для работы с массивами, например, функции `array_slice()`.
10. Напишите программу для объединения двух массивов в один с использованием специальной функции РНР для работы с массивами, например, функции `array_merge()`.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как определить стрелочную функцию в РНР? Какие особенности у стрелочных функций?
2. Какие встроенные функции РНР можно использовать для определения типа переменной? Какие значения они возвращают?
3. Как установить тип переменной в РНР с помощью функции `settype()`? Какие параметры принимает эта функция?
4. Для чего используется явное преобразование типов в РНР? Какие операторы используются для явного преобразования типов?
5. Как РНР автоматически преобразует типы переменных? Какие типы преобразуются друг в друга?

6. Какие функции предназначены для явного преобразования значения в строку, целое число, действительное число и другие типы данных?
7. Какие особенности преобразования типов происходят при использовании функции `echo` для вывода значений на страницу?
8. В каких случаях полезно использовать стрелочные функции в PHP? Какие преимущества они предоставляют по сравнению с обычными анонимными функциями?
9. Какие операторы используются для определения переменной как целочисленной, булевой, действительной и других типов данных?
10. Какие специальные функции предназначены для проверки типа переменной в PHP? Какие значения они возвращают при успешной и неудачной проверке?

ПРАКТИЧЕСКАЯ РАБОТА №15. СОЗДАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ

Цель работы: овладеть навыками работы с файлами в PHP, включая чтение, запись, создание и удаление файлов, а также обработку их содержимого.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Работа с файловой системой

Как и большинство языков программирования, PHP поддерживает работу с файлами, которые являются одним из способов хранения информации.

Чтение и запись файлов

Открытие и закрытие файлов

Для открытия файлов в PHP определена функция `fopen()`. Она имеет следующее определение: `resource fopen(string $filename, string $mode)`. Первый параметр `$filename` представляет путь к файлу, а второй - режим открытия. В зависимости от цели открытия и типа файла данный параметр может принимать следующие значения:

- `'r'`: файл открывается только для чтения. Если файла не существует, возвращает `false`
- `'r+'`: файл открывается только для чтения с возможностью записи. Если файла не существует, возвращает `false`
- `'w'`: файл открывается для записи. Если такой файл уже существует, то он перезаписывается, если нет - то он создается
- `'w+'`: файл открывается для записи с возможностью чтения. Если такой файл уже существует, то он перезаписывается, если нет - то он создается
- `'a'`: файл открывается для записи. Если такой файл уже существует, то данные записываются в конец файла, а старые данные остаются. Если файл не существует, то он создается
- `'a+'`: файл открывается для чтения и записи. Если файл уже существует, то данные дозаписываются в конец файла. Если файла нет, то он создается

Результатом функции `fopen` будет дескриптор файла. Этот дескриптор используется для операций с файлом и для его закрытия.

После окончания работы файл надо закрыть с помощью функции `fclose()`, которая принимает в качестве параметра дескриптор файла. Например, откроем и закроем файл:

```
1 $fd = fopen("form.php", 'r') or die("не удалось открыть файл");
2 fclose($fd);
```

Конструкция `or die("текст ошибки")` позволяет прекратить работу скрипта и вывести некоторое сообщение об ошибке, если функция `fopen` не смогла открыть файл.

Чтение файла

Для чтения файла можно использовать несколько функций. Для построчного чтения используется функция `fgets()`, которая получает дескриптор файла и возвращает одну считанную строку. Пройдем построчно по всему файлу:

```
1 <?php
2 $fd = fopen("form.php", 'r') or die("не удалось открыть файл");
3 while(!feof($fd))
4 {
5     $str = htmlentities(fgets($fd));
6     echo $str;
7 }
8 fclose($fd);
9 ?>
```

При каждом вызове `fgets()` PHP будет помещать указатель в конец считанной строки. Чтобы проследить окончание файла, используется функция `feof()`, которая возвращает `true` при завершении файла. И пока не будет достигнут конец файла, мы можем применять функцию `fgets()`.

Чтение файла полностью

Если нам надо прочитать файл полностью, то мы можем облегчить себе жизнь, применив функцию `file_get_contents()`:

```
1 <?php
2 $str = htmlentities(file_get_contents("form.php"));
3 echo $str;
4 ?>
```

При этом нам не надо открывать явно файл, получать дескриптор, а затем закрывать файл.

Поблочное считывание

Также можно провести поблочное считывание, то есть считывать определенное количество байт из файла с помощью функции `fread()`:

```
1  <?php
2  $fd = fopen("form.php", 'r') or die("не удалось открыть файл");
3  while(!feof($fd))
4  {
5      $str = htmlentities(fread($fd, 600));
6      echo $str;
7  }
8  fclose($fd);
9  ?>
```

Функция `fread()` принимает два параметра: дескриптор считываемого файла и количество считываемых байтов. При считывании блока указатель в файле становится в конец этого блока. И также с помощью функции `feof()` можно отследить завершение файла.

Запись файла

Для записи файла применяется функция `fwrite()`, которая записывает в файл строку:

```
1  <?php
2  $fd = fopen("hello.txt", 'w') or die("не удалось создать файл");
3  $str = "Привет мир";
4  fwrite($fd, $str);
5  fclose($fd);
6  ?>
```

Аналогично работает другая функция `fputs()`:

```
1  <?php
2  $fd = fopen("hello.txt", 'w') or die("не удалось создать файл");
3  $str = "Привет мир";
4  fputs($fd, $str);
5  fclose($fd);
6  ?>
```

Работа с указателем файла

При открытии файла для чтения или записи в режиме `'w'`, указатель в файле помещается в начало. При считывании данных РНР перемещает указатель в файле в конец блока считанных данных. Однако мы также вручную можем управлять указателем в файле и устанавливать его в произвольное место. Для этого

надо использовать функцию `fseek`, которая имеет следующее формальное определение:

```
int fseek (resource $handle , int $offset [, int $whence = SEEK_SET ] )
```

Параметр `$handle` представляет дескриптор файла. Параметр `$offset` - смещение в байтах относительно начала файла, с которого начнется считывание/запись. Третий необязательный параметр задает способ установки смещения. Он может принимать три значения:

- `SEEK_SET`: значение по умолчанию, устанавливает смещение в `offset` байт относительно начала файла
- `SEEK_CUR`: устанавливает смещение в `offset` байт относительно начала текущей позиции в файле
- `SEEK_END`: устанавливает смещение в `offset` байт от конца файла

В случае удачной установки указателя функция `fseek ()` возвращает 0, а при неудачной установке возвращает -1.

Пример использования функции:

```
1 $fd = fopen ("hello.txt", 'w+') or die ("не удалось открыть файл");
2 $str = "Привет мир!"; // строка для записи
3 fwrite ($fd, $str); // запишем строку в начало
4 fseek ($fd, 0); // поместим указатель файла в начало
5 fwrite ($fd, "Хрю"); // запишем в начало строку
6 fseek ($fd, 0, SEEK_END); // поместим указатель в конец
7 fwrite ($fd, $str); // запишем в конце еще одну строку
8 fclose($fd);
```

Перемещение файла

Для перемещения файла применяется функция `rename ()`:

```
1 <?php
2 if (! rename ("hello.txt", "subdir/hello.txt"))
3     echo "Ошибка перемещения файла";
4 else echo "Файл перемещен";
5 ?>
```

Если у нас в каталоге файла `hello.txt` имеется подкаталог `subdir`, то файл будет в него перемещен. Если файл был успешно перемещен, функция возвратит значение `true`.

Копирование файла

Для копирования файла используется функция `copy ()`. Она принимает имя копируемого файла, и имя копии файла. И если копирование прошло успешно, возвращает значение `true`:

```
1 <?php
2 if (copy ("hello.txt", "hello_copy.txt"))
3     echo "Копия файла создана";
4 else echo "Ошибка копирования файла";
5 ?>
```

Удаление файла

Для удаления файла применяется функция `unlink`, которая принимает имя файла и возвращает значение `true` при успешном удалении файла:

```
1 <?php
2 if (unlink("hello_copy.txt"))
3     echo "Файл удален";
4 else echo "Ошибка при удалении файла";
5 ?>
```

Создание каталога

Для создания каталога используется функция `mkdir ()`:

```
1 if(mkdir("newdir"))
2     echo "Каталог создан";
3 else
4     echo "Ошибка при создании каталога";
```

В данном случае `mkdir` создает новый каталог "newdir" в текущем каталоге. Если создание пройдет успешно, то функция возвращает значение `true`, иначе - `false`

Для создания новой папки в корневом каталоге можно использовать выражение `mkdir("/newdir")`.

Удаление каталога

Для удаления каталога применяется функция `rmdir()`. Ее использование аналогично `mkdir ()`:

```
1 if(rmdir("newdir"))
2     echo "Каталог удален";
3 else
4     echo "Ошибка при удалении каталога";
```

Операции с каталогами

Для получения абсолютного пути к текущему каталогу используется функция `getcwd()`, которая возвращает путь в виде строки:

```
1 $path = getcwd();
```

```
2 echo $path; // C:\localhost
```

Функция `opendir()` открывает определенный каталог для считывания из него информации о файлах и каталогах. При успешном открытии каталога функция возвращает дескриптор открытого каталога. После окончания работы с каталогом его надо закрыть функцией `closedir()`.

Для считывания имени отдельного файла в открытом каталоге применяется функция `readdir()`.

Теперь объединим эти функции и выведем на страницу все файлы и подкаталоги из текущего каталога:

```
1 <?php
2 $dir = getcwd(); // получаем текущий каталог
3
4 if (is_dir($dir)) // является ли путь каталогом
5 {
6     if ($dh = opendir($dir)) // открываем каталог
7     {
8         // считываем по одному файл или подкаталогу
9         // пока не дойдем до конца
10        while (($file = readdir($dh)) !== false)
11        {
12            // пропускаем символы .. и .
13            if ($file=='.' || $file=='..') continue;
14            // если каталог или файл
15            if(is_dir($file)) echo "каталог: $file <br>";
16            else echo "файл:  $file <br>";
17        }
18        closedir($dh); // закрываем каталог
19    }
20 }
21 ?>
```

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №15

1. Напишите программу на PHP для создания текстового файла и записи в него простого текстового сообщения.
2. Разработайте программу для чтения содержимого созданного в первом задании текстового файла и вывода его на экран.
3. Создайте программу для добавления новой строки в конец текстового файла без удаления его текущего содержимого.
4. Напишите программу для поиска конкретного слова в текстовом файле и вывода всех строк, в которых это слово встречается.

5. Разработайте программу для создания CSV-файла и записи в него данных, представленных в виде таблицы (например, имя, возраст, email).
6. Создайте программу для чтения данных из созданного CSV-файла и вывода их на экран в удобочитаемом формате.
7. Напишите программу для удаления файла с заданным именем, если он существует.
8. Разработайте программу для копирования содержимого одного файла в другой.
9. Создайте программу для перемещения файла из одной директории в другую.
10. Напишите программу для создания архива из нескольких файлов и их последующей распаковки.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая функция в РНР используется для открытия файлов? Какие параметры принимает эта функция?
2. Какие режимы открытия файлов существуют в РНР? Для чего используются каждый из них?
3. Как можно закрыть открытый файл в РНР? Почему важно закрывать файлы после работы с ними?
4. Какая функция РНР используется для построчного чтения файла? Как организовать цикл для построчного чтения файла до его конца?
5. Как можно прочитать содержимое файла целиком, без использования цикла? Какая функция здесь используется?
6. Как осуществить поблочное считывание файла в РНР? Какие функции для этого применяются?
7. Какая функция используется для записи данных в файл? Какие параметры принимает эта функция?
8. Как управлять указателем файла в РНР? Какая функция используется для этого? Какие параметры принимает эта функция?
9. Какие способы существуют для управления указателем файла? Когда и почему может быть полезно перемещать указатель в файле?
10. Какие могут быть последствия неправильного использования функций для работы с файлами в РНР? Как

можно предотвратить возможные проблемы при работе с файловой системой?

ПРАКТИЧЕСКАЯ РАБОТА №16. SQL – ЗАПРОСЫ И ИХ ОБРАБОТКА С ПОМОЩЬЮ PHP

Цель работы: овладеть навыками выполнения SQL-запросов к базе данных с использованием PHP, обработки результатов запросов и отображения информации на веб-странице.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

PHP поддерживает работу с базой данных MySQL. Специальные встроенные функции для работы с MySQL позволяют просто и эффективно работать с этой СУБД: выполнять любые запросы, читать и записывать данные, обрабатывать ошибки.

Сценарий, который подключается к БД, выполняет запрос и показывает результат, будет состоять всего из нескольких строк. Для работы с MySQL не надо ничего дополнительно устанавливать и настраивать; всё необходимое уже доступно вместе со стандартной поставкой PHP.

`mysqli` (MySQL Improved) — это расширение PHP, которое добавляет в язык полную поддержку баз данных MySQL. Это расширение поддерживает множество возможностей современных версий MySQL.

Типичный процесс работы с СУБД в PHP-сценарии состоит из нескольких шагов:

1. Установить подключение к серверу СУБД, передав необходимые параметры: адрес, логин, пароль.
2. Убедиться, что подключение прошло успешно: сервер СУБД доступен, логин и пароль верные и так далее.
3. Сформировать правильный SQL запрос (например, на чтение данных из таблицы).
4. Убедиться, что запрос был выполнен успешно.
5. Получить результат от СУБД в виде массива из записей.
6. Использовать полученные записи в своём сценарии (например, показать их в виде таблицы).

Функция `mysqli connect`: соединение с MySQL

Перед началом работы с данными внутри MySQL, нужно открыть соединение с сервером СУБД. В PHP это делается с помощью стандартной функции `mysqli_connect ()`. Функция возвращает результат — ресурс соединения. Данный ресурс используется для всех следующих операций с MySQL.

Но чтобы выполнить соединение с сервером, необходимо знать как минимум три параметра:

- Адрес сервера СУБД;
- Логин;
- Пароль.

Если вы следовали стандартной процедуре установки MySQL или используете OpenServer, то адресом сервера будет `localhost`, логином — `root`. При использовании OpenServer пароль для подключения — это пустая строка `''`, а при самостоятельной установке MySQL пароль вы задавали в одном из шагов мастера установки.

Базовый синтаксис функции `mysqli_connect ()`:

`mysqli_connect (<адрес сервера>, <имя пользователя>, <пароль>, <имя базы данных>);`

Проверка соединения

Первое, что нужно сделать после соединения с СУБД — это выполнить проверку, что оно было успешным. Эта проверка нужна, чтобы исключить ошибку при подключении к БД. Неверные параметры подключения, неправильная настройка или высокая нагрузка заставит MySQL отвергать новые подключения. Все эти ситуации приведут к невозможности соединения, поэтому программист должен проверить успешность подключения к серверу, прежде чем выполнять следующие действия.

Соединение с MySQL устанавливается один раз в сценарии, а затем используется при всех запросах к БД.

Результатом выполнения функции `mysqli_connect ()` будет значение специального типа — ресурс. Если подключение к MySQL не удалось, то функция `mysqli_connect ()` вместо ресурса вернёт логическое значение типа «ложь» — `false`. Хорошей практикой будет всегда проверять результат выполнения этой функции и сравнивать его с ложью.

Соединение с MySQL и проверка на ошибки:

```

<?php
$link = mysqli_connect ("localhost", "root", "");

if ($link == false) {
    print ("Ошибка: Невозможно подключиться к MySQL " . mysqli_connect_error());
}
else {
    print("Соединение установлено успешно");
}
?>

```

Функция `mysqli_connect_error()` просто возвращает текстовое описание последней ошибки MySQL.

Установка кодировки

После установки соединения желательно явно задать кодировку, которая будет использоваться при обмене данными с MySQL. Если этого не сделать, то вместо записей со значениями, написанными кириллицей, можно получить последовательность из знаков вопроса: ???????????????. Вызовите эту функцию сразу после успешной установки соединения:

```
mysqli_set_charset($con, "utf8");
```

Выполнение запросов

Установив соединение и определив кодировку, мы готовы выполнить свои первые SQL-запросы. Вы уже умеете составлять корректные SQL команды и выполнять их через консольный или визуальный интерфейс MySQL-клиента. Те же самые запросы можно отправлять без изменений и из PHP-сценария. Помогут в этом несколько встроенных функций языка.

Два вида запросов

Следует разделять все SQL-запросы на две группы:

1. Чтение информации (SELECT).
2. Модификация (UPDATE, INSERT, DELETE).

При выполнении запросов из среды PHP, запросы из второй группы возвращают только результат их исполнения: успех или ошибку.

Запросы первой группы при успешном выполнении возвращают специальный ресурс результата. Его, в свою очередь, можно преобразовать в ассоциативный массив (если нужна одна запись) или в двумерный массив (если требуется список записей).

Добавление записи

Вернёмся к нашему проекту — дневнику наблюдений за погодой. Начнём практическую работу с заполнения таблиц данными. Для начала добавим хотя бы один город в таблицу cities.

Выражение INSERT INTO используется для добавления новых записей в таблицу базы данных.

Составим корректный SQL-запрос на вставку записи с именем города, а затем выполним его путём передачи этого запроса в функцию mysqli_query(), чтобы добавить новые данные в таблицу.

```
<?php
$link = mysqli_connect ("localhost", "root", "");

$sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';
$result = mysqli_query($link, $sql);

if ($result == false) {
    print("Произошла ошибка при выполнении запроса");
}
```

Обратите внимание, что первым параметром для функции mysqli_query () передаётся ресурс подключения, полученный от функции mysqli_connect (), вторым параметром следует строка с SQL-запросом.

При запросах на изменение данных (не SELECT) результатом выполнения будет логическое значение — true или false, которое будет означать, что запрос выполнить не удалось. Для получения строки с описанием ошибки существует функция mysqli_error(\$link).

Функция insert id: как получить идентификатор добавленной записи

Следующим шагом будет добавление погодной записи для нового города. Погодные записи хранит таблица weather_log, но, чтобы сослаться на город, необходимо знать идентификатор записи из таблицы cities.

Здесь пригодится функция mysqli_insert_id(). Она принимает единственный аргумент — ресурс соединения, а возвращает идентификатор последней добавленной записи.

Теперь у нас есть всё необходимое, чтобы добавить погодную запись. Вот как будет выглядеть комплексный пример с подключением к MySQL и добавлением двух новых записей:

```
<?php
$link = mysqli_connect("localhost", "root", "");

if ($link == false){
    print("Ошибка: Невозможно подключиться к MySQL " . mysqli_connect_error());
}
else {
    $sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';
    $result = mysqli_query($link, $sql);

    if ($result == false) {
        print("Произошла ошибка при выполнении запроса");
    }
    else {
        $city_id = mysqli_insert_id($link);

        $sql = 'INSERT INTO weather_log SET city_id = ' . $city_id . ', day = "2017-09-03",
temperature = 10, cloud = 1';

        $result = mysqli_query($link, $sql);

        if ($result == false) {
            print("Произошла ошибка при выполнении запроса");
        }
    }
}
}
```

Чтение записей

Другая частая операция при работе с базами данных в PHP — это получение записей из таблиц (запросы типа SELECT). Составим SQL-запрос, который будет использовать SELECT выражение. Затем выполним этот запрос с помощью функции `mysqli_query()`, чтобы получить данные из таблицы.

В этом примере показано, как вывести все существующие города из таблицы `cities`:

```
<?php

$sql = 'SELECT id, name FROM cities';

$result = mysqli_query($link, $sql);

while ($row = mysqli_fetch_array($result)) {
    print("Город: " . $row['name'] . "; Идентификатор: " . $row['id'] . "<br>");
}
```

В примере выше результат выполнения функции `mysqli_query()` сохранён в переменной `$result`. В этой переменной находятся не данные из таблицы, а специальный тип данных — так называемая ссылка на результаты запроса.

Чтобы получить действительные данные, то есть записи из таблицы, следует использовать другую функцию — `mysqli_fetch_array()` — и передать ей единственным параметром эту самую ссылку. Теперь каждый вызов функции `mysqli_fetch_array()` будет возвращать следующую запись из всего результирующего набора записей в виде ассоциативного массива.

Цикл `while` здесь используется для «прохода» по всем записям из полученного набора записей. Значение поля каждой записи можно узнать, просто обратившись по ключу этого ассоциативного массива.

Как получить сразу все записи в виде двумерного массива

Иногда бывает удобно после запроса на чтение не вызывать в цикле `mysqli_fetch_array` для извлечения очередной записи по порядку, а получить их сразу все одним вызовом. PHP так тоже умеет.

Функция `mysqli_fetch_all($res, MYSQLI_ASSOC)` вернёт двумерный массив со всеми записями из результата последнего запроса. Перепишем пример с показом существующих городов с её использованием:

```
<?php
$sql = 'SELECT id, name FROM cities';
$result = mysqli_query($link, $sql);

$rows = mysqli_fetch_all($result, MYSQLI_ASSOC)

foreach ($rows as $row) {
    print("Город: " . $row['name'] . "; Идентификатор: " . $row['id'] . "<br>");
}
```

Как узнать количество записей

Часто бывает необходимо узнать, сколько всего записей вернёт выполненный SQL-запрос. Это может помочь при организации постраничной навигации или просто в качестве информации. Узнать число записей поможет функция

`mysqli_num_rows()`, которой следует передать ссылку на результат запроса.

ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №16

1. Напишите программу на PHP для подключения к базе данных MySQL и выполнения простого запроса SELECT для получения всех записей из определенной таблицы.
2. Разработайте программу для выполнения запроса SELECT с использованием условия WHERE для выборки конкретных записей из таблицы.
3. Создайте программу для выполнения запроса INSERT для добавления новой записи в таблицу базы данных.
4. Напишите программу для выполнения запроса UPDATE для обновления данных в существующей записи таблицы.
5. Разработайте программу для выполнения запроса DELETE для удаления записи из таблицы.
6. Создайте программу для выполнения запроса SELECT с использованием условия WHERE и сортировки данных с помощью оператора ORDER BY.
7. Напишите программу для выполнения запроса SELECT с использованием функции COUNT() для подсчета числа записей в таблице.
8. Разработайте программу для выполнения запроса SELECT с использованием оператора JOIN для объединения данных из нескольких таблиц.
9. Создайте программу для выполнения запроса SELECT с использованием оператора GROUP BY для группировки данных по определенному столбцу.
10. Напишите программу для выполнения запроса SELECT с использованием оператора LIMIT для выборки определенного количества записей из таблицы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие основные цели преследует работа с базами данных в PHP?

2. Что такое расширение `mysqli` в PHP, и как оно связано с работой с базой данных MySQL?
3. Как устанавливается соединение с сервером базы данных MySQL в PHP? Какие параметры необходимо указать при использовании функции `mysqli_connect()`?
4. Почему важно проверить успешность соединения с базой данных после вызова функции `mysqli_connect()`? Как это можно сделать?
5. Какая функция используется для установки кодировки при работе с базой данных MySQL в PHP? Почему это важно?
6. Какие два вида запросов можно различить при работе с базами данных? В чем основное отличие между ними?
7. Какие действия выполняются при выполнении запросов из первой и второй групп запросов?
8. Как сформировать и выполнить SQL-запрос на добавление новой записи в таблицу базы данных с помощью PHP?
9. Как проверить успешность выполнения SQL-запроса на добавление записи? Какие действия могут быть предприняты в случае возникновения ошибки?
10. Почему важно обрабатывать возможные ошибки при выполнении SQL-запросов в PHP? Как это может повлиять на работу приложения?

ЛАБОРАТОРНАЯ РАБОТА №1. ОСНОВЫ РАБОТЫ С JS-ФРЕЙМВОРКОМ: VUE.JS. ОСНОВЫ ДОСТУПА К ДАННЫМ. ВЫВОД ИНФОРМАЦИИ НА СТРАНИЦУ

Цель работы: Изучение основ работы с JS-фреймворком Vue.js, освоение базовых методов доступа к данным и вывода информации на веб-страницу.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Vue — прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов, Vue создавался пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), упрощая интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для разработки сложных одностраничных приложений (SPA, Single-Page Applications), если использовать его в комбинации с современными инструментами и дополнительными библиотеками (opens new window).

Декларативная отрисовка

В ядре Vue.js находится система, позволяющая декларативно отрисовывать данные в DOM с помощью простого синтаксиса шаблонов:

```
<div id="counter">
  Счётчик: {{ counter }}
</div>
const Counter = {
  data() {
    return {
      counter: 0
    }
  }
}
Vue.createApp(Counter).mount('#counter')
```

Вот и первое Vue-приложение! Хотя и выглядит как простая отрисовка строкового шаблона, но Vue выполнил немало работы. Данные и DOM теперь **реактивно** связаны. Как в этом убедиться? Посмотрите на пример ниже, где свойство counter увеличивается каждую секунду и увидите, как изменяется DOM:

```
const Counter = {
```

```

data() {
  return {
    counter: 0
  },
},
mounted() {
  setInterval(() => {
    this.counter++
  }, 1000)
}
}

```

Кроме интерполяции текста также можно связывать данные с атрибутами элементов:

```

<div id="bind-attribute">
  <span v-bind:title="message">
    Наведи на меня курсор на пару секунд, чтобы
    увидеть динамически связанное значение title!
  </span>
</div>
const AttributeBinding = {
  data() {
    return {
      message: 'Страница загружена ' + new Date().toLocaleString()
    }
  }
}
Vue.createApp(AttributeBinding).mount('#bind-attribute')

```

Здесь повстречались с чем-то новым. Атрибут `v-bind` называется **директивой**. Директивы именуются с префикса `v-`, который обозначает, что это специальные атрибуты Vue, и, как уже можно догадаться, они добавляют особое реактивное поведение отрисованному DOM. В примере выше директива говорит *«сохраняй значение title этого элемента актуальным при изменении свойства message в текущем активном экземпляре»*.

Работа с пользовательским вводом

Чтобы позволить пользователям взаимодействовать с приложением, можно использовать директиву `v-on` для обработчиков событий, которые будут вызывать методы экземпляра:

```

<div id="event-handling">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Перевернуть сообщение</button>
</div>

```

```
const EventHandling = {
  data() {
    return {
      message: 'Привет, Vue.js!'
    }
  },
  methods: {
    reverseMessage() {
      this.message = this.message
        .split("")
        .reverse()
        .join("")
    }
  }
}
Vue.createApp(EventHandling).mount('#event-handling')
```

Обратите внимание, в методе не трогаем DOM и обновляем только состояние приложения — всеми манипуляциями с DOM занимается Vue, а в коде фокусируемся на логике работы.

Vue также предоставляет директиву `v-model`, которая реализует двустороннюю привязку между элементом формы и состоянием приложения:

```
<div id="two-way-binding">
  <p>{{ message }}</p>
  <input v-model="message" />
</div>
const TwoWayBinding = {
  data() {
    return {
      message: 'Привет, Vue!'
    }
  }
}
Vue.createApp(TwoWayBinding).mount('#two-way-binding')
```

Композиция приложения из компонентов

Компонентная система является ещё одной важной концепцией во Vue, потому что это абстракция, которая позволяет создавать большие приложения, состоящие из небольших, автономных и часто переиспользуемых компонентов. Если задуматься, то почти любой тип интерфейса приложения можно абстрактно представить в виде дерева компонентов (рис. 1.1)

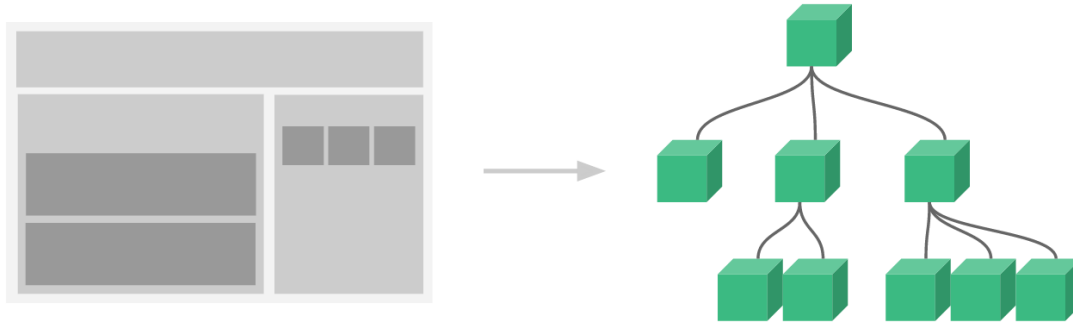


Рисунок 1.1 – Дерево компонентов

Компонент во Vue — по сути экземпляр с предустановленными опциями. Его регистрация также проста: нужно создать объект компонента, как это уже делали с объектом `app`, и указать его в родительской опции `components`:

```
const TodoItem = {
  template: `<li>Это одна из задач</li>`
}

// Создаём Vue-приложение
const app = Vue.createApp({
  components: {
    TodoItem // Регистрируем новый компонент
  },
  ... // Остальные свойства для компонента
})

// Монтируем приложение Vue
app.mount(...)
```

После этого можно использовать его в шаблоне другого компонента:

```
<ol>
  <!-- Создание экземпляра компонента todo-item -->
  <todo-item></todo-item>
</ol>
```

Пока что во всех элементах списка будет один и тот же текст, что не очень-то интересно. Должна быть возможность передавать данные в дочерние компоненты из родительской области видимости. Доработаем компонент, чтобы он принимал входной параметр:

```
const TodoItem = {
  props: ['todo'],
  template: `<li>{{ todo.text }}</li>`
}
```

Теперь можно передавать свой текст для каждого из компонентов с помощью v-bind:

```
<div id="todo-list-app">
  <ol>
    <!--
      Теперь можно передавать каждому компоненту todo-item объект с информацией
      о задаче, который может динамически изменяться. Также каждому компоненту
      определяем "key", назначение которого разберём далее в руководстве.
    -->
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id"
    ></todo-item>
  </ol>
</div>
const TodoItem = {
  props: ['todo'],
  template: `<li>{{ todo.text }}</li>`
}
const TodoList = {
  data() {
    return {
      groceryList: [
        { id: 0, text: 'Vegetables' },
        { id: 1, text: 'Cheese' },
        { id: 2, text: 'Whatever else humans are supposed to eat' }
      ]
    }
  },
  components: {
    TodoItem
  }
}
const app = Vue.createApp(TodoList)
app.mount('#todo-list-app')
```

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №1

1. Настройка окружения:

- Установите Node.js и npm, если они еще не установлены на вашем компьютере.
- Создайте новый проект, используя Vue CLI. Для этого выполните команду `vue create project_name`.
- В процессе создания проекта выберите конфигурацию, соответствующую вашим предпочтениям, или используйте настройки по умолчанию.

- После завершения создания проекта перейдите в его директорию с помощью команды `cd project_name`.
 - Убедитесь, что все зависимости установлены, выполнив команду `npm install`.
2. Создание компонентов
 - Создайте базовый компонент "App.vue", который будет являться основной структурой вашего приложения.
 - Создайте дополнительные компоненты, необходимые для отображения данных на странице, например, "UserData.vue" для отображения информации о пользователе.
 3. Изучить основные идеи заложенные в фреймворк Vue. Изучить синтаксис, используемый в шаблонах (текстовой интерполяции, `v-if`, `v-show`, `v-for` и `key`, `v-on`, `v-bind`, `v-model`) и используемые сокращения директив.
 4. Создать новую страницу `vue.html` в проекте из предыдущих лабораторных работ. Добавить ссылку на эту страницу в меню макета.
 5. К созданной странице подключить Vue с CDN и файл для вашего кода.
 6. В `html` добавить элемент, к которому будет подключен экземпляр Vue-приложения.
 7. В созданный элемент добавить форму с текстовым полем для создания записей с кнопкой о добавлении новой записи и секцию для добавленных записей.
 8. Получение и обработка данных:
 - Используйте методы жизненного цикла Vue.js, такие как `mounted`, для выполнения запросов к API или загрузки данных из других источников.
 - Обработайте полученные данные, приведите их к нужному формату и сохраните в состоянии вашего приложения, используя `Vueх` или локальное состояние компонента.
 9. Вывод информации:
 - Используйте директивы Vue.js, такие как `v-for` и `v-if`, для динамического вывода данных на веб-страницу.
 - Создайте шаблоны для отображения информации, учитывая возможность динамического изменения данных.
 10. Тестирование и отладка:

- Проверьте корректность отображения данных на веб-странице, а также правильность работы функциональности.
- В случае обнаружения ошибок выполните отладку кода, используя инструменты разработчика браузера или инструменты отладки Vue.js.

11. Анализ работы:

- Сделайте выводы о процессе разработки с использованием Vue.js, отметьте преимущества и недостатки фреймворка по сравнению с чистым JavaScript.
- Предложите возможные улучшения или дополнительные функциональности для вашего приложения.

12. Дополнительные задания:

- Реализуйте функционал редактирования данных на странице с использованием Vue.js.
- Добавьте анимации для улучшения пользовательского опыта при изменении данных.
- Интегрируйте сторонние API для получения дополнительной информации и ее отображения на странице.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключается идея реактивности?
2. Что подразумевается под прогрессивным фреймворком?
3. Чем библиотека отличается от фреймворка?
4. Зачем в принципе использовать фреймворки Vue/Angular?
5. Для чего используется виртуальный DOM в react/vue?
6. Назовите основные директивы во Vue.
7. Двустороннее и одностороннее связывание.
8. Методы и вычисляемые значения.
9. Разница между v-if и v-show.
10. Какие есть возможности у API интернационализации Intl?

ЛАБОРАТОРНАЯ РАБОТА №2. СОЗДАНИЕ ДИНАМИЧЕСКИХ АТТРИБУТОВ. ПОДКЛЮЧЕНИЕ И ОБРАБОТКА СОБЫТИЙ

Цель работы: овладение навыками создания динамических атрибутов и подключения/обработки событий в JavaScript для создания интерактивных пользовательских интерфейсов.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Экземпляры приложения & компонента

Создание экземпляра приложения

Создание любого приложения Vue начинается с создания нового **экземпляра приложения** с помощью функции `createApp`:

```
const app = Vue.createApp({
  /* опции */
})
```

Экземпляр приложения используется для регистрации «глобальных» вещей, которые будут затем использоваться компонентами внутри этого приложения. Подробнее познакомимся дальше в руководстве, но в качестве небольшого примера:

```
const app = Vue.createApp({})

app.component('SearchInput', SearchInputComponent)
app.directive('focus', FocusDirective)
app.use(LocalePlugin)
```

Большинство из методов экземпляра приложения возвращают этот же экземпляр, что позволяет составлять вызовы в цепочку:

```
Vue.createApp({})
  .component('SearchInput', SearchInputComponent)
  .directive('focus', FocusDirective)
  .use(LocalePlugin)
```

Полный список API приложения можно посмотреть в разделе Справочник API.

Корневой компонент

Опции, передаваемые в `createApp`, используются для настройки **корневого компонента**. При монтировании приложения он используется как стартовая точка для отрисовки.

Приложению требуется примонтироваться в DOM-элемент. Например, если потребуется примонтировать приложение Vue в `<div id="app"></div>`, то необходимо передать `#app`:

```
const RootComponent = {
  /* опции */
}
const app = Vue.createApp(RootComponent)
const vm = app.mount('#app')
```

В отличие от большинства других методов приложения, `mount` не возвращает экземпляр приложения — вместо этого он возвращает экземпляр корневого компонента.

Vue хоть и не реализует в полной мере паттерн MVVM (opens new window), но архитектура фреймворка во многом вдохновлена им. Поэтому, как правило, переменную с экземпляром приложения называют `vm` (сокращение от `ViewModel`).

Пусть все примеры на этой странице и нуждаются лишь в одном компоненте, большинство реальных приложений организованы в дерево вложенных, многократно переиспользуемых компонентов. Например, дерево компонентов приложения `todo`-списка может быть таким:

```
Корневой компонент
├─ TodoList
│   └─ TodoItem
│       ├── DeleteTodoButton
│       └─ EditTodoButton
├─ TodoListFooter
│   └─ ClearTodosButton
└─ TodoListStatistics
```

Каждый компонент будет иметь свой собственный экземпляр компонента `vm`. У некоторых компонентов, таких как `TodoItem`, вероятно, будет несколько экземпляров, отображаемых в один момент времени. Все экземпляры компонентов в этом приложении будут иметь один и тот же экземпляр приложения.

Подробнее на системе компонентов остановимся позже. Сейчас достаточно запомнить, что корневой компонент на самом деле ничем не отличается от любого другого компонента. Опции конфигурации такие же, как и поведение соответствующего экземпляра компонента.

Свойства экземпляра компонента

Раньше в руководстве встречались свойства `data`. Все свойства, объявленные в `data`, доступны через экземпляр компонента:

```
const app = Vue.createApp({
  data() {
    return { count: 4 }
  }
})
const vm = app.mount('#app')
console.log(vm.count) // => 4
```

Есть и другие опции компонента, добавляющие в экземпляр компонента пользовательские свойства, например `methods`, `props`, `computed`, `inject` и `setup`. Подробнее о каждой из них поговорим далее в руководстве. Все свойства экземпляра компонента, независимо от того, как они определены, будут доступны в шаблоне компонента.

Через экземпляр компонента `Vue` также предоставляет доступ к некоторым встроенным свойствам, например `$attrs` и `$emit`. Такие свойства всегда именуются с префиксом `$`, чтобы избежать конфликта имён с пользовательскими свойствами.

Хуки жизненного цикла

При создании каждый экземпляр проходит через серию шагов инициализации — например, устанавливает наблюдение за данными, компилирует шаблон, монтирует экземпляр в DOM, обновляет DOM при изменении данных. Между шагами вызываются функции, называемые **хуками жизненного цикла**, предоставляющие возможность выполнять код на определённых этапах.

Например, хук `created` можно использовать для запуска кода после создания экземпляра:

```
Vue.createApp({
  data() {
    return { count: 1 }
  },
  created() {
    // `this` указывает на экземпляр vm
    console.log('счётчик: ' + this.count) // => "счётчик: 1"
  }
})
```

Также есть и другие хуки, которые будут вызываться на различных этапах жизненного цикла экземпляра, например `mounted`, `updated` и `unmounted`. Все хуки вызываются с контекстом `this`, указывающим на текущий активный экземпляр, который их вызвал.

Не используйте стрелочные функции (opens new window) в свойствах экземпляра и в коллбэках, например `created: () => console.log(this.a)` или `vm.$watch('a', newVal => this.myMethod())`. Так как стрелочные функции не имеют собственного `this`, то `this` в коде будет обрабатываться как любая другая переменная и её поиск будет производиться до тех пор, пока она не будет найдена в области видимости выше, что часто приводит к ошибкам, таким как `Uncaught TypeError: Cannot read property of undefined` или `Uncaught TypeError: this.myMethod is not a function`.

Диаграмма жизненного цикла

Ниже представлена диаграмма жизненного цикла экземпляра (рис. 2.1).

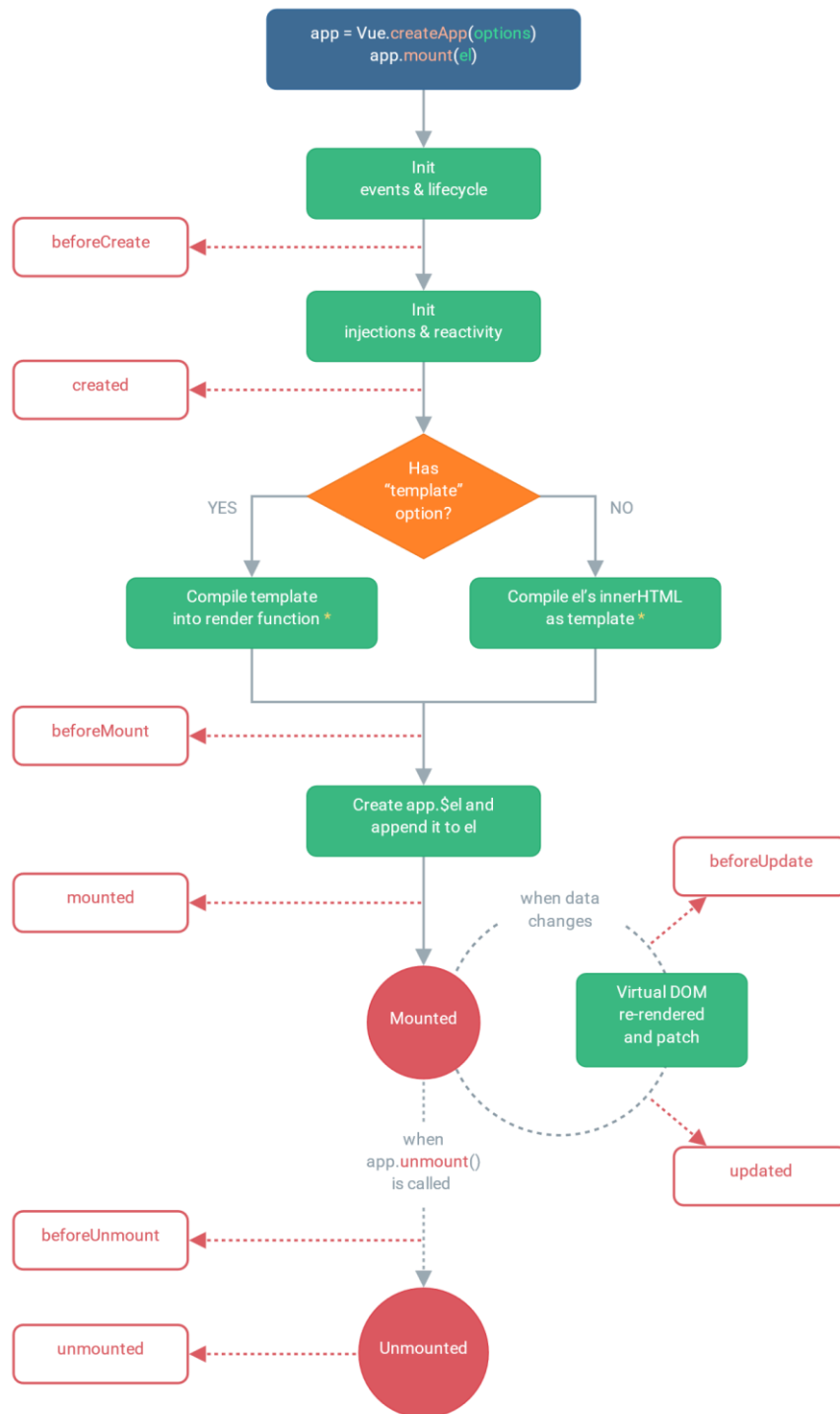


Рисунок 2.1 – Диаграмма жизненного цикла экземпляра

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №2

1. Создание базовой HTML-структуры:
 - Создайте файл HTML с базовой структурой, который будет содержать элементы, с которыми вы будете работать.

- В структуре HTML определите элементы, которые будут взаимодействовать с пользователем, например, кнопки, поля ввода или блоки текста.
2. Создание динамических атрибутов:
 - Используйте JavaScript для добавления динамических атрибутов к элементам на странице, например, изменение цвета или размера элементов в зависимости от действий пользователя.
 - Реализуйте функционал, который будет изменять атрибуты элементов в реальном времени в ответ на действия пользователя.
 3. Подключение и обработка событий:
 - Добавьте обработчики событий к элементам на странице, например, события клика, наведения курсора или ввода текста.
 - Реализуйте функционал, который будет реагировать на события пользователя, например, изменять содержимое элементов или вызывать другие функции.
 4. Тестирование и отладка:
 - Протестируйте созданный функционал, убедитесь, что динамические атрибуты изменяются корректно в зависимости от действий пользователя.
 - Выполните отладку, если необходимо, чтобы исправить возможные ошибки в коде.
 5. Анализ работы:
 - Оцените процесс создания динамических атрибутов и обработки событий в JavaScript.
 - Сделайте выводы о преимуществах и недостатках подхода, используемого в вашем коде.
 - Предложите возможные улучшения функционала или оптимизации кода.
 6. Дополнительные задания:
 - Реализуйте более сложные взаимодействия с элементами страницы, например, создание анимаций или изменение стилей элементов в зависимости от времени.

- Изучите и добавьте возможности использования библиотеки jQuery для более удобной работы с динамическими атрибутами и событиями.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как создать экземпляр приложения во Vue.js? Какие опции можно передать функции createApp?
2. Какие компоненты используются в приложении Vue.js? Как организуется дерево компонентов в приложении?
3. Какие свойства доступны через экземпляр компонента в Vue.js? Какие опции компонента могут добавлять пользовательские свойства?
4. Что такое хуки жизненного цикла в Vue.js? Какие этапы инициализации проходит каждый экземпляр приложения?
5. Какие встроенные свойства предоставляет Vue.js через экземпляр компонента? Как они именуются и для чего используются?
6. Какие методы экземпляра приложения возвращает себя же? Как это может быть полезно при организации кода?
7. Как можно примонтировать приложение Vue в DOM-элемент? Какие действия происходят при монтировании приложения в DOM?

ЛАБОРАТОРНАЯ РАБОТА №3. СОЗДАНИЕ ПРОСТЫХ КОМПОНЕНТОВ И ИХ ИСПОЛЬЗОВАНИЕ НА СТРАНИЦЕ

Цель работы: научиться создавать простые компоненты и использовать их на веб-странице для повышения модульности и повторного использования кода.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Создание компонентов

Компоненты представляют элементы, к которым компилятор Vue прикрепляет некоторое поведение. Компоненты позволяют инкапсулировать код и затем использовать его многократно в различных частях приложения.

Для создания компонента используется функция `Vue.component(tagName, options)`, где параметр `tagName` - кастомный элемент html, который будет представлять компонент, а параметр `options` представляет конфигурацию компонента.

Например, определим простейший компонент:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Компоненты Vue.js</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <div id="app">
9    <hello></hello>
10   <hello></hello>
11 </div>
12 <script src="https://unpkg.com/vue@2.7.14/dist/vue.js"></script>
13 <script>
14   Vue.component('hello', {
15     template: '<h2>Hello</h2>'
16   })
17   new Vue({
18     el: "#app"
19   });
20 </script>
21 </body>
```

22 </html>

Здесь определен компонент для элемента `hello`. И неважно, что такого элемента в `html` нет, мы его сами создаем. Более того мы не можем использовать для компонента встроенные элементы `html` типа `div` или `h2`. То есть по сути компонент называется "hello".

В объекте `options`, который передается компоненту, через свойство `template` можно задать `html`-разметку, которую будет содержать компонент. В итоге данная разметка будет вставляться вместо элемента `<hello></hello>`.

При этом компонент должен быть определен до элемента `Vue`, в котором он используется.

И при рендеринге страницы вместо элемента `hello` будет вставлено содержимое компонента (рис. 3.1).

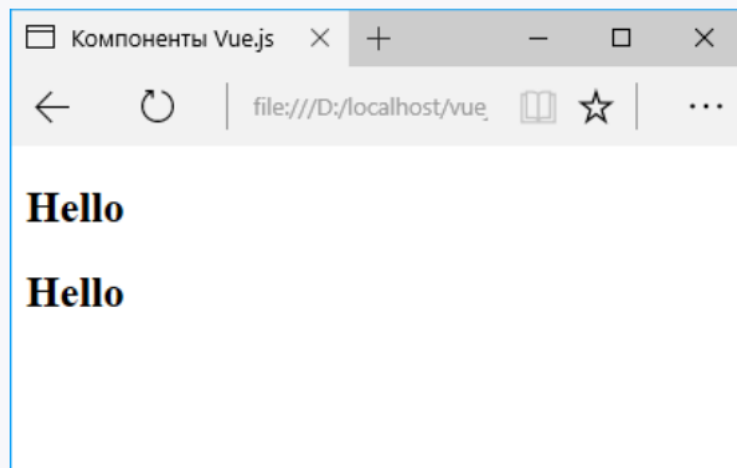


Рисунок 3.1 – Создание компонентов

При этом мы можем многократно использовать компонент в рамках приложения `Vue`, и в каждом случае будет происходить рендеринг компонента.

Причем компонент может использоваться только в рамках того элемента, к которому прикреплен объект `Vue`. То есть мы не можем написать наподобие, вынеся использование компонента во вне:

```
1  <hello></hello>
2  <div id="app">
3  </div>
```

Локальная и глобальная регистрация компонентов

Компоненты могут быть зарегистрированы локально и глобально. Глобальные компоненты доступны для любого объекта Vue на веб-странице. Локальные компоненты доступны только в рамках определенных объектов Vue.

Для локальной регистрации компонентов у объекта Vue устанавливается свойство `components`:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Компоненты Vue.js</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <div id="app">
9    <section-header></section-header>
10   <section-content></section-content>
11 </div>
12 <script src="https://unpkg.com/vue@2.7.14/dist/vue.js"></script>
13 <script>
14 Vue.component('section-header',{
15   template:<h3>Header</h3>'
16 });
17 var comp = {
18   template:<div>Hello World</div>'
19 };
20
21 new Vue({
22   el: "#app",
23   components:{
24     'section-content':comp
25   }
26 });
27 </script>
28 </body>
29 </html>

```

Здесь определен глобальный компонент - `section-header`. Глобальный компонент определяется с помощью метода `Vue.component()`.

И также здесь определен компонент `comp`. По сути, он представляет объект, который передается в качестве второго параметра в `Vue.component()` и может иметь все те же свойства,

например, свойство `template`. Этот компонент локально регистрируется в объекте `Vue`:

```
1 components:{
2   'section-content':comp
3 }
```

Причем для его рендеринга будет использоваться элемент `<section-content>` (рис. 3.2).

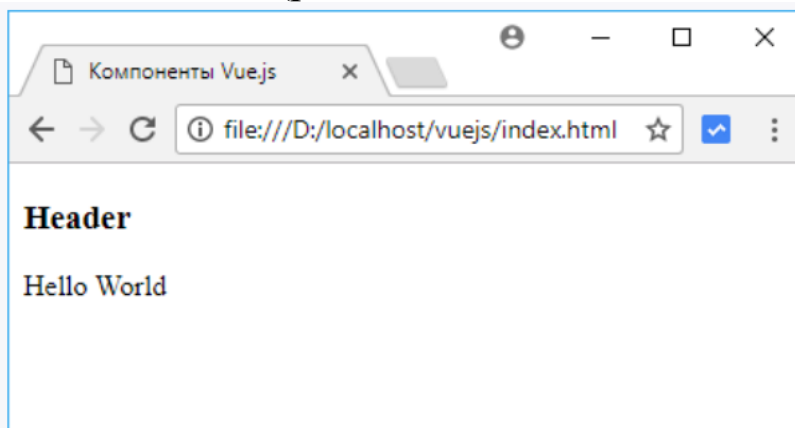


Рисунок 3.2 – Использование элемента `section-content`

Если бы компонент `comp` не был бы зарегистрирован в объекте `Vue`, то тогда мы бы его не могли использовать, либо пришлось бы делать его глобальным.

Рассмотрим еще один пример:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Компоненты Vue.js</title>
5 <meta charset="utf-8" />
6 </head>
7 <body>
8 <div id="app1">
9   <section-header></section-header>
10  <section-content></section-content>
11  <section-footer></section-footer>
12 </div>
13 <hr/>
14 <div id="app2">
15   <section-header></section-header>
16   <section-content></section-content>
17 </div>
18 <script src="https://unpkg.com/vue@2.7.14/dist/vue.js"></script>
19 <script>
```

```
20  Vue.component('section-header',{
21    template:<h2>Header</h2>'
22  });
23  var comp1 = {
24    template:<div>Content 1</div>'
25  };
26  var comp2 = {
27    template:<div>Content 2</div>'
28  };
29  var footer = {
30    template:<p><b>Footer</b></p>'
31  };
32  new Vue({
33    el: "#app1",
34    components:{
35      'section-content':comp1,
36      'section-footer':footer
37    }
38  });
39  new Vue({
40    el: "#app2",
41    components:{
42      'section-content':comp2
43    }
44  });
45  </script>
46  </body>
47  </html>
```

Здесь определено два объекта Vue. Компонент section-header является глобальным и поэтому доступен из любого объекта Vue. В дополнение к нему первый объект локально регистрирует два компонента - comp1 и footer, а второй объект Vue - один компонент comp2 (рис. 3.3).

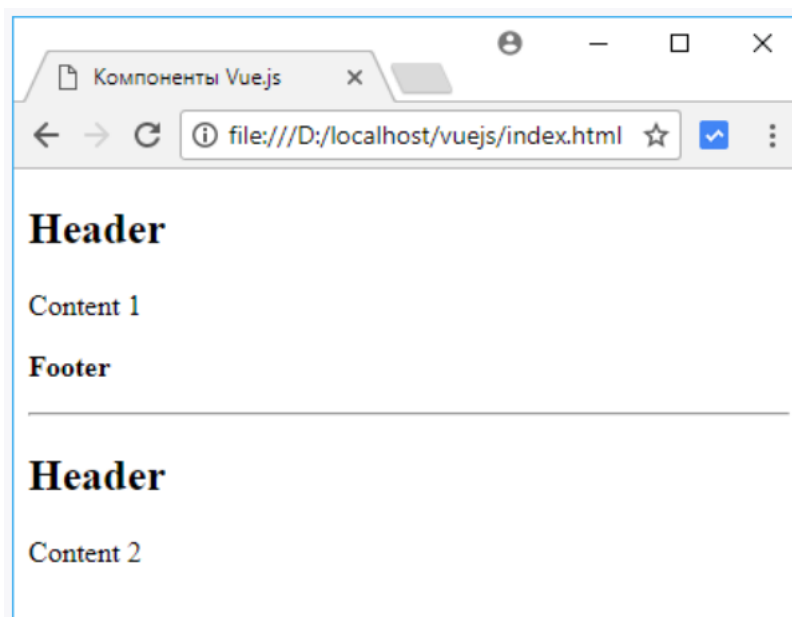


Рисунок 3.3 – Определение двух объектов

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №3

1. Создание базовых компонентов:
 - Создайте файлы JavaScript для каждого компонента, который вы планируете использовать на странице.
 - Определите структуру компонентов, включая HTML-разметку, CSS-стили и функциональность JavaScript.
 - Реализуйте базовые компоненты, такие как заголовок, кнопка, форма ввода и прочие, согласно вашим потребностям.
2. Использование компонентов на странице:
 - В HTML-файле вашей страницы создайте контейнеры для каждого компонента, которые вы создали на предыдущем шаге.
 - Подключите файлы JavaScript с вашими компонентами к вашей странице.
 - Вставьте компоненты на страницу, используя теги или специальные атрибуты, предоставленные фреймворком или библиотекой, с которыми вы работаете (Vue.js).
3. Конфигурирование компонентов:
 - Передайте параметры ваших компонентов, если это необходимо, для настройки их внешнего вида или поведения.

- Изучите возможности компонентов для взаимодействия друг с другом, например, через передачу событий или данных между компонентами.
- 4. Тестирование и отладка:
 - Протестируйте каждый компонент на корректность отображения и функциональность.
 - Выполните отладку, если необходимо, для исправления возможных ошибок или неполадок в работе компонентов.
- 5. Анализ работы:
 - Оцените удобство использования компонентов на странице и их вклад в модульность вашего кода.
 - Сделайте выводы о преимуществах и недостатках использования компонентов в вашем проекте.
- 6. Дополнительные задания:
 - Реализуйте более сложные компоненты, включающие интерактивные элементы или анимации.
 - Изучите возможности переиспользования компонентов в других проектах и разработайте стратегию для их хранения и организации.
 - Исследуйте методы тестирования компонентов, такие как модульное тестирование или тестирование пользовательского интерфейса, и примените их для вашего проекта.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как создать компонент в Vue.js? Какие параметры принимает функция `Vue.component`?
2. В чем разница между локальной и глобальной регистрацией компонентов в Vue.js? Какие преимущества и недостатки каждого подхода?
3. Как использовать созданный компонент в приложении Vue.js? Каким образом компоненты встраиваются в разметку HTML?
4. Какие свойства доступны в объекте `options` при создании компонента? Как использовать свойство `template` для определения внешнего вида компонента?

5. Можно ли использовать встроенные HTML-элементы в качестве компонентов в Vue.js? Почему?
6. Что такое локальные компоненты в Vue.js? Какие шаги необходимо выполнить для их регистрации и использования?
7. Какой порядок регистрации компонентов важен в Vue.js? Почему компонент должен быть определен до объекта Vue, в котором он используется?
8. Какие ограничения есть на использование компонентов в рамках приложения Vue.js? Можно ли использовать компоненты за пределами объекта Vue?
9. Каким образом можно повторно использовать компоненты в Vue.js? Какие принципы рекомендуется придерживаться при создании компонентов для повышения их модульности и удобства использования?

ЛАБОРАТОРНАЯ РАБОТА №4. РАБОТА СО СЛОЖНЫМИ КОМПОНЕНТАМИ

Цель работы: научиться создавать и работать с более сложными компонентами, которые включают в себя вложенные компоненты, использование состояния и передачу данных между компонентами.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Родительские и дочерние компоненты

Одни компоненты (родительские компоненты) могут содержать другие (дочерние компоненты). Например, один компонент выводит список объектов, а для вывода отдельного объекта используется еще один компонент (рис. 4.1):

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Компоненты Vue.js</title>
5  <meta charset="utf-8" />
6  <style>
7  .userdetails{
8    border-bottom: 1px solid #888;
9  }
10 </style>
11 </head>
12 <body>
13 <div id="app">
14   <userslist :users="users"></userslist>
15 </div>
16 <script src="https://unpkg.com/vue@2.7.14/dist/vue.js"></script>
17 <script>
18 Vue.component('userdetails', {
19   props: ["user"],
20   template: `<div class="userdetails">
21     <p>Name: {{user.name}}</p>
22     <p>Age: {{user.age}}</p>
23   </div>`
24 });
25 Vue.component('userslist', {
26   props: ["users"],
27   template: `<div>
```

```

28     <userdetails v-for="user in users" :key="user.name"
29 :user="user"></userdetails>
30     </div>`
31 });
32 new Vue({
33   el: "#app",
34   data: {
35     users: [{
36       name: 'Tom',
37       age: 18
38     }, {
39       name: 'Bob',
40       age: 23
41     }, {
42       name: 'Alice',
43       age: 21
44     }]
45   }
46 });
47 </script>
48 </body>
</html>

```

Здесь в компонент `userslist` передается список объектов `users`. Для вывода каждого отдельного объекта `userslist` использует еще один компонент `userdetails`. С помощью директивы `v-for` происходит перебор списка объектов, и каждый объект передается в компонент `userdetails`.

В принципе мы могли бы определить все в одном компоненте, однако выделение отдельного компонента `userdetails` позволяет развивать и обновлять его разметку отдельно от родительского компонента. Например, если потребуется изменить структуру разметки `html` в компоненте, то достаточно это сделать в коде компонента `userdetails`. К тому же мы можем повторно использовать данный компонент в других компонентах и частях программы (рис. 4.1).

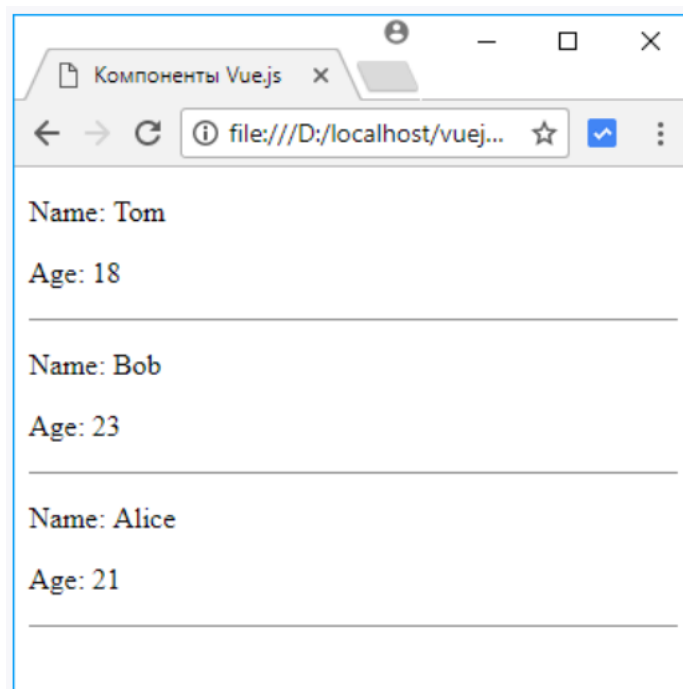


Рисунок 4.1 – Использование родительских и дочерних компонентов

Получение состояния вложенного компонента

С помощью ссылки `this.$refs` из объекта `Vue` мы можем ссылаться на различные элементы веб-страницы внутри объекта шаблона `Vue`. Но кроме того, подобным образом мы можем ссылаться также и на вложенные компоненты и, таким образом, обращаться к внутреннему состоянию компонента.

Например, установим в объекте `Vue` изменение свойства, которое определено в компоненте (рис. 4.2):

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Компоненты Vue.js</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <div id="app">
9    <userdetails :user="user" ref="details"></userdetails>
10   <button v-on:click="toggle()">Show</button>
11 </div>
12 <script src="https://unpkg.com/vue@2.7.14/dist/vue.js"></script>
13 <script>
14 Vue.component('userdetails', {
15   props: ["user"],
16   template: `<div>
```

```

17     <h2>Информация о пользователе</h2>
18     <div v-if="visible">
19         <p>Name: {{user.name}}</p>
20         <p>Age: {{user.age}}</p>
21     </div>
22 </div>`,
23 data: function(){
24     return{
25         visible: false
26     }
27 }
28 });
29 new Vue({
30     el: "#app",
31     data: {
32         user:{
33             name: 'Tom',
34             age: 18
35         }
36     },
37     methods: {
38         toggle: function(){
39             this.$refs.details.visible = !this.$refs.details.visible;
40         }
41     }
42 });
43 </script>
44 </body>
45 </html>

```

Здесь для компонента `userdetails` с помощью атрибута `ref` установлена ссылка `details`, через которую можно сослаться на данный компонент.

```
1 <userdetails :user="user" ref="details"></userdetails>
```

В самом компоненте `userdetails` определено свойство `visible`, которое управляет видимостью частью шаблона компонента. Для изменения значения этого свойства в объекте `Vue` предусмотрена кнопка, по нажатию на которую срабатывает метод `toggle()`:

```

1 toggle: function(){
2     this.$refs.details.visible = !this.$refs.details.visible;
3 }

```

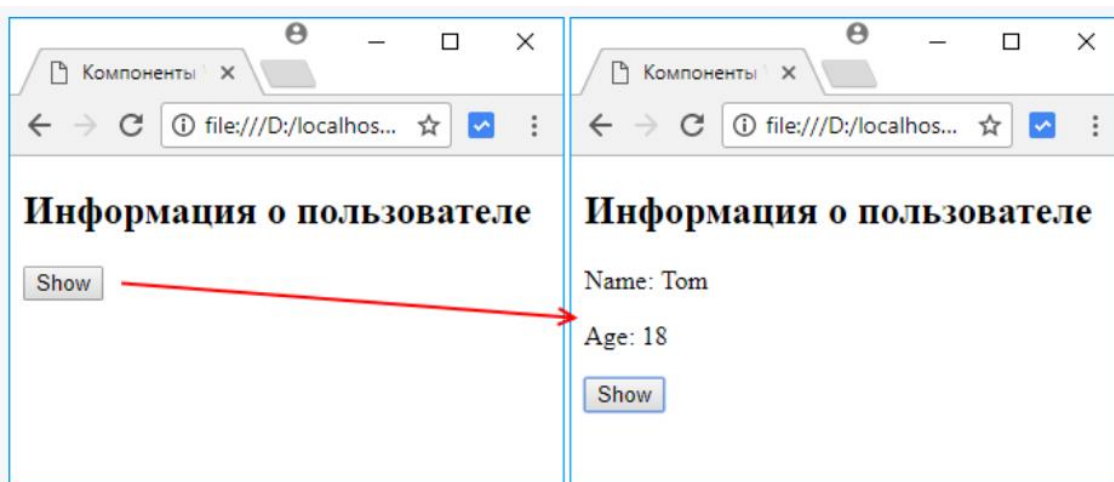


Рисунок 4.2 – Получение состояния вложенного компонента

Передача функций обратного вызова в компоненты

Функции обратного вызова представляют еще один способ взаимодействия между родительским и дочерним компонентами: родительские компоненты могут определять функции, а вызываются они в дочерних компонентах.

Например, определим следующую веб-страницу (рис. 4.3):

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Компоненты Vue.js</title>
5  <meta charset="utf-8" />
6  </head>
7  <body>
8  <div id="app">
9    <h2>Список пользователей</h2>
10   <userform :addfn="add"></userform>
11   <div>
12     <useritem v-for="(user, index) in users"
13       :user="user"
14       :key="index"
15       :index="index"
16       :removefn="remove">
17     </useritem>
18   </div>
19 </div>
20 <script src="https://unpkg.com/vue@2.7.14/dist/vue.js"></script>
21 <script>
22   Vue.component('userform', {
23     props: ["addfn"],

```

```

24     data: function () {
25         return {
26             user: { name:"", age:18}
27         }
28     },
29     template: `<div>
30         <input type="text" v-model="user.name" />
31         <input type="number" v-model="user.age" />
32         <button v-on:click="addfn({ name:user.name, age: user.age})">Add</button>
33     </div>`
34 });
35 Vue.component('useritem', {
36     props: ["user", "index", "removefn"],
37     template: `<div>
38         <p>Name: {{user.name}} <br> Age: {{user.age}}</p>
39         <button v-on:click="removefn(index)">Delete</button>
40     </div>`
41 });
42 new Vue({
43     el: "#app",
44     data: {
45         users:[
46             {name: 'Tom', age: 23},
47             {name: 'Bob', age: 26},
48             {name: 'Alice', age: 28}
49         ]
50     },
51     methods:{
52         remove: function(index){
53             this.users.splice(index, 1);
54         },
55         add: function(user){
56             this.users.push(user);
57         }
58     }
59 });
60 </script>
61 </body>
62 </html>

```

Объект Vue выводит на страницу массив элементов и определяет два метода для управления элементами: add (для добавления) и remove (для удаления).

Для добавления нового элемента определен компонент `userform`, который представляет форму с полями ввода. В этот компонент передается метод `add` в виде функции `addfn`:

```
1 <userform :addfn="add"></userform>
```

Причем функция `addfn` определена в компоненте `userform` через `props`:

```
1 Vue.component('userform', {
2   props: ["addfn"],
```

При нажатии на кнопку эта функция будет вызываться, и ей будут передаваться введенные данные. Что фактически приведет к вызову метода `add` в родительском объекте `Vue`.

Аналогичная ситуация с компонентом `useritem`, который получает метод `remove` в виде функции `removefn`, которая также определяется через `props`. При нажатии на кнопку удаления вызывается функция `removefn`, ей передается индекс удаляемого элемента, и фактически будет идти вызов метода `remove` из объекта `Vue`.

Таким образом, в самих дочерних компонентах никаких действий по управлению массивом и объектами и по генерации событий не определено. Дочерние компоненты просто вызывают те методы, которые определены в родительском объекте.

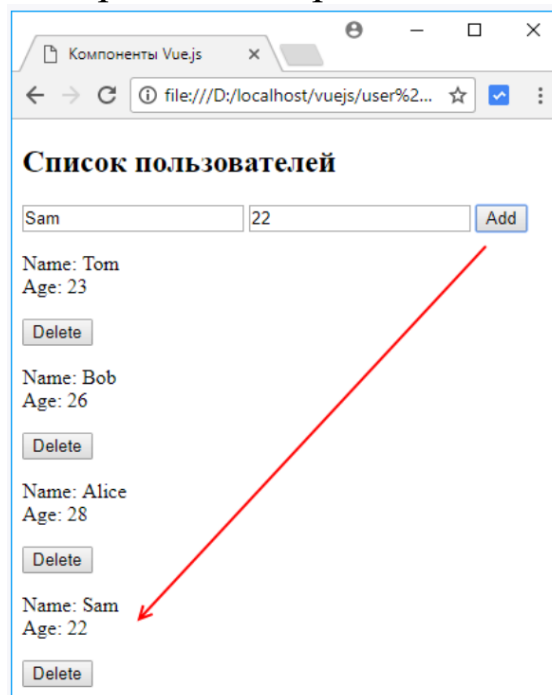


Рисунок 4.3 – Реализация функции добавления

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №4

1. Создание сложных компонентов:
 - Создайте базовые компоненты, которые будут служить основой для создания более сложных компонентов. Например, компоненты для отображения списка элементов, формы ввода данных, модального окна и т. д.
 - Разбейте функциональность на отдельные компоненты для удобства поддержки и повторного использования кода.
2. Использование вложенных компонентов:
 - Вставьте созданные базовые компоненты в другие компоненты, чтобы создать более сложные компоненты.
 - Проектируйте структуру вложенных компонентов таким образом, чтобы они были логически связаны и выполняли четко определенные функции.
3. Работа с состоянием и передача данных:
 - Используйте состояние компонентов для хранения данных и управления их поведением.
 - Передавайте данные между компонентами с помощью пропсов (props) и событий (events) для обеспечения взаимодействия между ними.
4. Тестирование и отладка:
 - Протестируйте работу более сложных компонентов на корректность отображения и функциональность.
 - Выполните отладку, если необходимо, чтобы исправить возможные ошибки или неполадки в работе компонентов.
5. Анализ работы:
 - Оцените удобство использования и поддержки более сложных компонентов в вашем проекте.
 - Сделайте выводы о преимуществах и недостатках использования сложных компонентов и их влиянии на модульность и читаемость кода.
6. Дополнительное задание:
 - Реализуйте более сложные компоненты, которые включают в себя анимации, динамическое изменение содержимого или взаимодействие с API.

- Изучите возможности использования библиотек или фреймворков, которые предоставляют готовые решения для создания сложных компонентов (например, Material-UI для React).
- Проведите анализ производительности более сложных компонентов и оптимизируйте их, если это необходимо.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличаются родительские и дочерние компоненты в Vue.js? Какие преимущества предоставляет использование вложенных компонентов?
2. Каким образом можно передать данные от родительского компонента к дочернему в Vue.js? Какие методы позволяют это сделать?
3. Что такое props в компонентах Vue.js? Какие параметры принимает опция props при определении компонента?
4. Как осуществляется передача функций обратного вызова от родительского компонента к дочернему в Vue.js? Зачем это нужно?
5. Каким образом можно получить доступ к внутреннему состоянию дочернего компонента из родительского в Vue.js? Какую роль играет объект \$refs в этом процессе?
6. Как использовать директиву v-for для повторного использования компонентов в Vue.js? Какие параметры могут использоваться при использовании директивы v-for для работы с компонентами?
7. Как передавать параметры и данные между компонентами, находящимися на одном уровне в иерархии компонентов в Vue.js?
8. Каким образом можно обновлять состояние родительского компонента из дочернего в Vue.js? Какие подходы можно использовать для этого?
9. Какие принципы проектирования компонентов следует соблюдать для обеспечения их удобства использования и повторного использования в приложениях Vue.js?

ЛАБОРАТОРНАЯ РАБОТА №5. РЕАЛИЗАЦИЯ ПРОГРАММ ПО МЕТОДОЛОГИИ ООП СРЕДСТВАМИ СЕРВЕРНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

Цель работы: овладение методологией объектно-ориентированного программирования (ООП) и ее применение при разработке программ с использованием средств серверного языка программирования.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ Объектно-ориентированное программирование

Объекты и классы

При создании программы на РНР и отдельных ее блоков нам вполне может хватить той функциональности, которую представляют функции. Однако РНР имеет и другие возможности по созданию программ, которые представляет объектно-ориентированное программирование. В ряде случаев программы, использующие ООП, проще в понимании, их легче поддерживать и изменять.

Ключевыми понятиями парадигмы ООП являются понятия "класс" и "объект". Описанием объекта является класс, а объект представляет экземпляр этого класса. Можно провести следующую аналогию: например, у каждого человека есть имя, определенный возраст, вес, какие-то другие параметры. То есть некоторый шаблон, который содержит набор параметров человека - этот шаблон можно назвать классом. А реально же существующий человек с конкретным именем, возрастом, весом и т.д. является объектом или экземпляром этого класса.

Для создания класса в РНР используется ключевое слово `class`, после которого идет название класса и фигурные скобки `{ }` - блок кода класса. Например, новый класс, представляющий пользователя:

```
1 class Person
2 { }
```

Чтобы создать объект класса `Person`, применяется ключевое слово `new`:

```
1 <?php
```

```

2  class Person
3  { }
4
5  $person = new Person();
6  print_r($person);
7  ?>

```

В данном случае переменная `$person` является объектом класса `Person`. С помощью функции `print_r()` можно вывести содержимое объекта, как и в случае с массивами.

При этом неважно, определяется класс до или после создания объекта. Например, мы можем сначала определить переменную класса, а потом определить этот класс:

```

1  <?php
2  $person = new Person();
3
4  class Person
5  { }
6  ?>

```

Свойства и методы

Класс может содержать переменные, которые описывают какие-то признаки объекта, его состояние и которые еще называют свойствами или атрибутами. И также класс может содержать функции, которые еще называют методами и которые определяют его поведение.

Так, добавим в класс `Person` несколько свойств и методов:

```

1  <?php
2  class Person
3  {
4      public $name, $age;
5
6      function hello()
7      {
8          echo "Hello!<br>";
9      }
10 }
11 $tom = new Person();
12 $tom->name = "Tom"; // установка свойства $name
13 $tom->age = 36; // установка свойства $age
14 $personName = $tom->name; // получение значения свойства $name
15 echo "Имя пользователя: " . $personName . "<br>";
16 $tom->hello(); // вызов метода hello()

```

```
17 print_r($tom);
18 ?>
```

Здесь класс `Person` содержит два свойства: `$name` и `$age`. Свойства объявляются как обычные переменные, перед которыми стоит модификатор доступа - в данном случае модификатор `public`.

Методы представляют обычные функции, которые выполняют определенные действия. Здесь функция `hello()` просто выводит приветствие.

После создания объекта класса `Person`:

```
1 $tom = new Person();
```

Мы можем через имя переменной класса обращаться к его свойствам и методам. Чтобы обратиться к свойствам и методам объекта применяется оператор доступа `->`. Например, установить значения свойств:

```
1 $tom->name = "Tom"; // установка свойства $name
2 $tom->age = 36; // установка свойства $age
```

Или получить значение (например, присвоить его переменной):

```
1 $personName = $tom->name; // получение значения свойства $name
```

Или вызвать методы объекта:

```
1 $tom->hello(); // вызов метода hello()
```

В итоге мы получим следующий вывод браузера:

Имя пользователя: Tom

Hello!

Person Object ([name] => Tom [age] => 36)

При этом свойствам можно задать в классе некоторые начальные значения:

```
1 <?php
2 class Person
3 {
4     public $name = "Undefined", $age = 18;
5
6     function hello()
7     {
8         echo "Hello!<br>";
9     }
```

```

10  }
11  $tom = new Person();
12  $tom->age = 36; // установка свойства $age
13
14  echo "Имя пользователя: " . $tom->name . "<br>";
15  echo "Возраст пользователя: " . $tom->age . "<br>";
16  ?>

```

Вывод браузера:

Имя пользователя: Undefined

Возраст пользователя: 36

Ключевое слово **this**

Для обращения к свойствам и методам объекта внутри его класса применяется ключевое слово **this**. Например, определим в классе метод для вывода информации об объекте:

```

1  <?php
2  class Person
3  {
4      public $name = "Undefined", $age = 18;
5
6      function displayInfo()
7      {
8          echo "Name: " . $this->name . "; Age: " . $this->age . "<br>";
9          // также можно написать
10         // echo "Name: $this->name; Age: $this->age<br>";
11     }
12 }
13 $tom = new Person();
14 $tom -> name = "Tom";
15 $tom -> displayInfo(); // Name: Tom; Age: 18
16 ?>

```

Для обращения к полям и методам внутри класса также применяется оператор доступа **->**, перед которым идет **\$this**. Причем это **\$this** указывает именно на текущий объект. Что это значит в практическом плане? Например:

```

1  <?php
2  class Person
3  {
4      public $name = "Undefined", $age = 18;
5
6      function displayInfo()

```

```

7      {
8          echo "Name: $this->name; Age: $this->age<br>";
9      }
10     }
11     $tom = new Person();
12     $tom -> name = "Tom";
13     $tom -> displayInfo();
14
15     $bob = new Person();
16     $bob -> name = "Bob";
17     $bob -> age = 25;
18     $bob -> displayInfo();
19     ?>

```

При вызове

```
1 $tom -> displayInfo();
```

`$this` фактически будет указывать на переменную `$tom`.
Тогда как при вызове

```
1 $bob -> displayInfo();
```

`$this` будет указывать на переменную `$bob`.

Вывод браузера:

Name: Tom; Age: 18

Name: Bob; Age: 25

Сравнение объектов

При сравнении объектов классов следует принимать во внимание ряд особенностей. В частности, при использовании оператора равенства `==` два объекта считаются равными, если они представляют один и тот же класс и их свойства имеют одинаковые значения.

А при использовании оператора эквивалентности `===` оба объекта считаются равными, если обе переменных классах указывают на один и тот же экземпляр класса.

Рассмотрим на примере:

```

1 <?php
2 class Person
3 {
4     public $name, $age;
5     function displayInfo()
6     {
7         echo "Name: $this->name; Age: $this->age<br>";

```

```

8      }
9  }
10 $tom = new Person();
11 $tom -> name = "Tom";
12 $tom -> age = 36;
13
14 $tomas = new Person();
15 $tomas -> name = "Tom";
16 $tomas -> age = 36;
17
18 if($tom == $tomas) echo "переменные tom и tomas равны<br>";
19 else echo "переменные tom и tomas НЕ равны<br>";
20
21 if($tom === $tomas) echo "переменные tom и tomas эквивалентны";
22 else echo "переменные tom и tomas НЕ эквивалентны";
23 ?>

```

Здесь сравниваются две переменных - \$tom и \$tomas. Они представляют один и тот же класс Person, и их свойства имеют одни и те же значения. Однако они представляют разные объекты. Поэтому при сравнении оператор == возвратит true, а оператор === - false:

переменные tom и tomas равны

переменные tom и tomas НЕ эквивалентны

Возьмем другой пример, когда обе переменных представляют один и тот же объект:

```

1 $person = new Person();
2
3 $tom = $person;
4 $tom -> name = "Tom";
5 $tom -> age = 36;
6
7 $tomas = $person;
8
9 if($tom == $tomas) echo "переменные tom и tomas равны<br>";
10 else echo "переменные tom и tomas НЕ равны<br>";
11
12 if($tom === $tomas) echo "переменные tom и tomas эквивалентны";
13 else echo "переменные tom и tomas НЕ эквивалентны";

```

Здесь объект класса Person создается только один раз: \$person = new Person();. И затем обе переменных \$tom и

\$tomas будут указывать на этот объект. При этом не имеет значения, для какой именно переменной мы устанавливаем свойства. Так как в реальности это будет один и тот же объект. В итоге и оператор `==`, и оператор `===` при сравнении возвратят `true`.

переменные `tom` и `tomas` равны

переменные `tom` и `tomas` эквивалентны

Наследование

Наследование является одним из основных аспектов объектно-ориентированного программирования. Наследование позволяет классу взять функционал уже имеющихся классов и при необходимости переопределить его. Если у нас есть какой-нибудь класс, в котором не хватает пары функций, то гораздо проще переопределить имеющийся класс, написав пару строк, чем создавать новый с нуля, переписывая кучу кода.

Чтобы наследовать один класс от другого, нам надо применить оператор `extends`. Стоит отметить, что в PHP мы можем унаследовать класс только от одного класса. Множественное наследование не поддерживается.

Например, унаследуем класс `Employee` от класса `Person`:

```

1  <?php
2  class Person
3  {
4      public $name;
5      function __construct($name)
6      {
7          $this->name = $name;
8      }
9      function displayInfo()
10     {
11         echo "Имя: $this->name<br>";
12     }
13 }
14 class Employee extends Person
15 {}
16
17 $tom = new Employee("Tom");
18 $tom -> displayInfo();
19 ?>
```


В данном случае предположим, что класс `Person` представляет человека в целом, а класс `Employee` - работника некоего предприятия. В этой связи каждый работник представляет человека. И чтобы не дублировать один и тот же функционал, лучше в данном случае унаследовать класс работника - `Employee` от класса человека - `Person`. В этой паре класс `Person` еще называется родительским или базовым классом, а класс - `Employee` - производным классом или классом-наследником.

И так как класс `Employee` унаследован от `Person`, для объектов класса `Employee` мы можем использовать функционал родительского класса `Person`. Так, для создания объекта `Employee` в данном случае вызывается конструктор, который определен в классе `Person` и который в качестве параметра принимает имя человека:

```
1 $tom = new Employee("Tom");
```

И также у переменной типа `Employee` вызывается метод `displayInfo`, который определен в классе `Person`:

```
1 $tom -> displayInfo();
```

Переопределение функционала

Унаследовав функционал от родительского класса, класс-наследник может добавить свои свойства и методы или переопределить унаследованный функционал. Например, изменим класс `Employee`, добавив в него данные о компании, где работает работник:

```
1 <?php
2 class Person
3 {
4     public $name;
5     function __construct($name)
6     {
7         $this->name = $name;
8     }
9     function displayInfo()
10    {
11        echo "Имя: $this->name<br>";
12    }
13 }
14 class Employee extends Person
```

```

15  {
16      public $company;
17      function __construct($name, $company)
18      {
19          $this->name = $name;
20          $this->company = $company;
21      }
22      function displayInfo()
23      {
24          echo "Имя: $this->name<br>";
25          echo "Работает в $this->company<br>";
26      }
27  }
28
29  $tom = new Employee("Tom", "Microsoft");
30  $tom -> displayInfo();
31  ?>

```

Здесь класс Employee добавляет новое свойство - \$company, которое хранит компанию работника. Также класс Employee переопределил конструктор, в который передаются данные для имени и компании. А также переопределен метод displayInfo(). Соответственно для создания объекта класса Employee, теперь необходимо использовать переопределенный в классе Employee конструктор:

```
1  $tom = new Employee("Tom", "Microsoft");
```

Класс-наследник переопределяет конструктор родительского класса, то для создания объекта класса-наследника необходимо использовать переопределенный в нем конструктор.

И также изменится поведение метода displayInfo(), который кроме имени также выведет и компанию работника:

```
Имя: Tom
Работает в Microsoft
```

Вызов функционала родительского класса

Если мы посмотрим на код класса-наследника Employee, то можем увидеть части кода, которые повторяют код класса Person. Например, установка имени в конструкторе:

```
1  $this->name = $name;
```

Также вывод имени работника в методе `displayInfo()`:

```
1 echo "Имя: $this->name<br>";
```

В обоих случаях речь идет об одной строке кода. Однако, что, если конструктор `Employee` повторяет установку не одного, а десятка свойств. Соответственно, что, если метод `displayInfo` в классе-наследнике повторяет гораздо больше действий родительского класса. В этом случае гораздо рациональнее не писать повторяющийся код в классе-наследнике, а вызвать в нем соответствующий функционал родительского класса.

Если нам надо обратиться к методу родительского класса, то мы можем использовать ключевое слово `parent`, после которого используется двойное двоеточие `::` и затем вызываемый метод.

Например, перепишем предыдущий пример:

```
1 <?php
2 class Person
3 {
4     public $name;
5     function __construct($name)
6     {
7         $this->name = $name;
8     }
9     function displayInfo()
10    {
11        echo "Имя: $this->name<br>";
12    }
13 }
14 class Employee extends Person
15 {
16     public $company;
17     function __construct($name, $company)
18     {
19         parent::__construct($name);
20         $this->company = $company;
21     }
22     function displayInfo()
23     {
24         parent::displayInfo();
25         echo "Работает в $this->company<br>";
26     }
```

```

27 }
28
29 $tom = new Employee("Tom", "Microsoft");
30 $tom -> displayInfo();
31 ?>

```

Теперь в конструкторе Employee вызывается конструктор базового класса:

```

1 parent::__construct($name);

```

В нем, собственно, и происходит установка имени. И подобным образом в методе displayInfo() вызывается реализация метода класса Person:

```

1 parent::displayInfo();

```

В итоге мы получим тот же самый результат.

Стоит отметить, что в реальности ключевое слово parent заменяет название класса. То есть мы также могли вызывать функционал родительского класса через имя этого класса:

```

1 class Employee extends Person
2 {
3     public $company;
4     function __construct($name, $company)
5     {
6         Person::__construct($name);
7         $this->company = $company;
8     }
9     function displayInfo()
10    {
11        Person::displayInfo();
12        echo "Работает в $this->company<br>";
13    }
14 }

```

Оператор instanceof

Оператор instanceof позволяет проверить принадлежность объекта определенному классу. Слева от оператора располагается объект, который надо проверить, а справа - название класса. И если объект представляет класс, то оператор возвращает true. Например:

```

1 class Person

```

```

2  {
3      public $name;
4      function __construct($name)
5      {
6          $this->name = $name;
7      }
8      function displayInfo()
9      {
10         echo "Имя: $this->name<br>";
11     }
12 }
13 class Employee extends Person
14 {
15     public $company;
16     function __construct($name, $company)
17     {
18         Person::__construct($name);
19         $this->company = $company;
20     }
21     function displayInfo()
22     {
23         Person::displayInfo();
24         echo "Работает в $this->company<br>";
25     }
26 }
27 class Manager{ }
28
29 $tom = new Employee("Tom", "Microsoft");
30
31 $tom instanceof Employee; // true
32 $tom instanceof Person;   // true
33 $tom instanceof Manager;  // false

```

Здесь переменная `$tom` представляет класс `Employee`, поэтому `$tom instanceof Employee` возвращает `true`.

Так как класс `Employee` унаследован от `Person`, то переменная `$tom` также представляет класс `Person` (работник также является человеком).

А вот класс `Manager` переменная `$tom` не представляет, поэтому выражение `$tom instanceof Manager` возвращает `false`.

Запрет наследования и оператор `final`

В примере выше метод `displayInfo()` переопределялся классом-наследником. Однако иногда возникают ситуации, когда надо запретить переопределение методов. Для этого в классе-родителе надо указать методы с модификатором `final`:

```

1  class Person
2  {
3      public $name;
4      function __construct($name)
5      {
6          $this->name = $name;
7      }
8      final function displayInfo()
9      {
10         echo "Имя: $this->name<br>";
11     }
12 }
13 class Employee extends Person
14 {
15     public $company;
16     function __construct($name, $company)
17     {
18         Person::__construct($name);
19         $this->company = $company;
20     }
21     function displayEmployeeInfo()
22     {
23         Person::displayInfo();
24         echo "Работает в $this->company<br>";
25     }
26 }
27
28 $tom = new Employee("Tom", "Microsoft");
29 $tom -> displayEmployeeInfo();

```

В этом случае во всех классах-наследниках от класса `Person` мы уже не сможем определить метод с таким же именем. Поэтому в данном случае в классе `Employee` определен новый метод - `displayEmployeeInfo`.

Также мы можем вообще запретить наследование от класса. Для этого данный класс надо определить с модификатором `final`:

```

1  final class Person
2  {

```

```

3     public $name;
4     function __construct($name)
5     {
6         $this->name = $name;
7     }
8     final function displayInfo()
9     {
10        echo "Имя: $this->name<br>";
11    }
12 }

```

Теперь мы не сможем унаследовать класс Employee (да и никакой другой класс) от класса Person.

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №5

1. Определение классов и объектов:
 - Выберите предметную область, для которой будет разрабатываться программа.
 - Определите основные классы и объекты в этой предметной области, выделив их характеристики и методы.
2. Реализация классов и объектов:
 - Напишите код на php, реализующий определенные классы и объекты.
 - Организуйте классы и объекты в соответствии с основными принципами ООП, такими как инкапсуляция, наследование и полиморфизм.
3. Взаимодействие между объектами:
 - Определите взаимодействие между объектами, например, через методы или события.
 - Реализуйте эту логику в коде, обеспечивая корректное взаимодействие объектов.
4. Тестирование и отладка:
 - Протестируйте созданные классы и объекты на корректность работы и соответствие требованиям предметной области.
 - Выполните отладку, если необходимо, для устранения ошибок или неполадок в коде.
5. Анализ работы:

- Оцените эффективность применения методологии ООП при разработке программы.
 - Сделайте выводы о преимуществах и недостатках использования ООП в контексте вашего проекта.
6. Дополнительное задание:
- Реализуйте более сложную логику ваших классов, включающую взаимодействие с базой данных или внешними сервисами.
 - Примените паттерны проектирования для улучшения структуры вашего кода и повышения его читаемости и поддерживаемости.
 - Изучите возможности многопоточного программирования или параллельных вычислений и примените их в вашем проекте.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как создать класс в РНР? Приведите пример определения класса.
2. Как создать объект класса в РНР? Приведите пример создания объекта и вызова его методов.
3. Что такое свойства (атрибуты) и методы класса в объектно-ориентированном программировании? Приведите пример класса с несколькими свойствами и методами.
4. Как установить значения свойств объекта класса? Приведите пример установки значений свойств объекта.
5. Как получить доступ к значениям свойств объекта класса в РНР? Приведите пример.
6. Что такое ключевое слово `$this` в РНР? Как оно используется в объектно-ориентированном программировании?
7. Как можно вызвать метод объекта класса из самого класса в РНР? Приведите пример.
8. Как происходит сравнение объектов класса в РНР? Чем отличается оператор `==` от оператора `===` при сравнении объектов?

ЛАБОРАТОРНАЯ РАБОТА №6. СОЗДАНИЕ БАЗЫ ДАННЫХ MYSQL С ПОМОЩЬЮ УТИЛИТЫ PHPMYADMIN

Цель работы: научиться создавать базу данных с использованием MySQL и утилиты phpMyAdmin, а также освоить основные операции с базой данных, такие как создание таблиц, добавление данных и выполнение запросов.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

База данных (БД) — это организованная структура, предназначенная для хранения/изменения/обработки взаимосвязанной информации. MySQL — это система управления базами данных (СУБД). В свою очередь, phpMyAdmin — это специальное веб-приложение, предназначенное для администрирования СУБД MySQL. Рассмотрим подробно как работать с интерфейсом phpMyAdmin, как создать базу данных в этом приложении и как ее удалить.

Создание новой базы в phpMyAdmin: инструкция 2024

Переходим в phpMyAdmin. Для создания базы данных выбираем команду «Создать БД» в левой части интерфейса. (рис. 6.1)

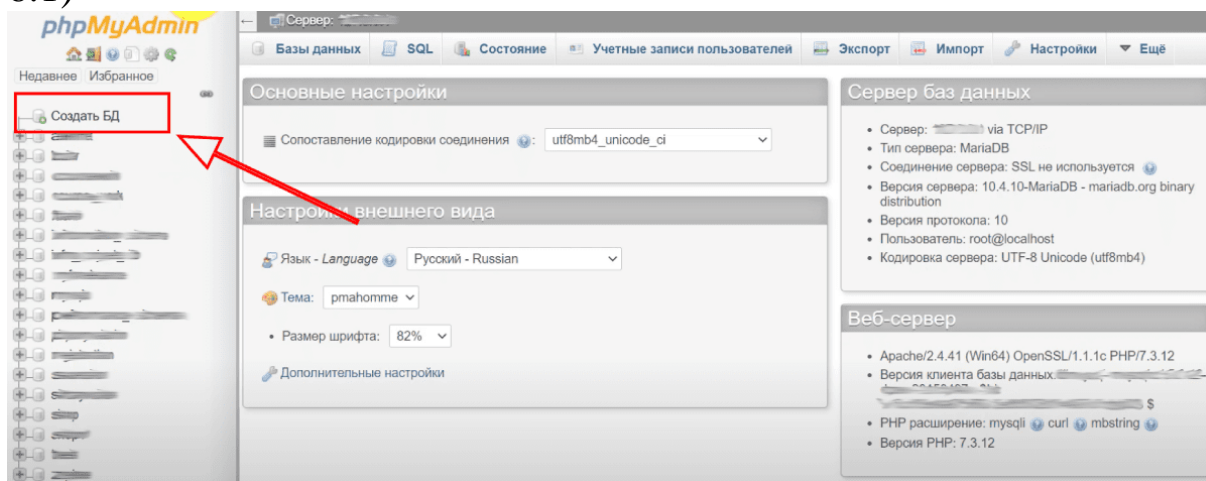


Рисунок 6.1 – Создание БД

В поле «Имя базы данных» придумываем название, обычно его берут из названия сайта/домена. Кодировку оставляем «utf8mb4_general_ci». Нажимаем кнопку «Создать» (рис. 6.2).

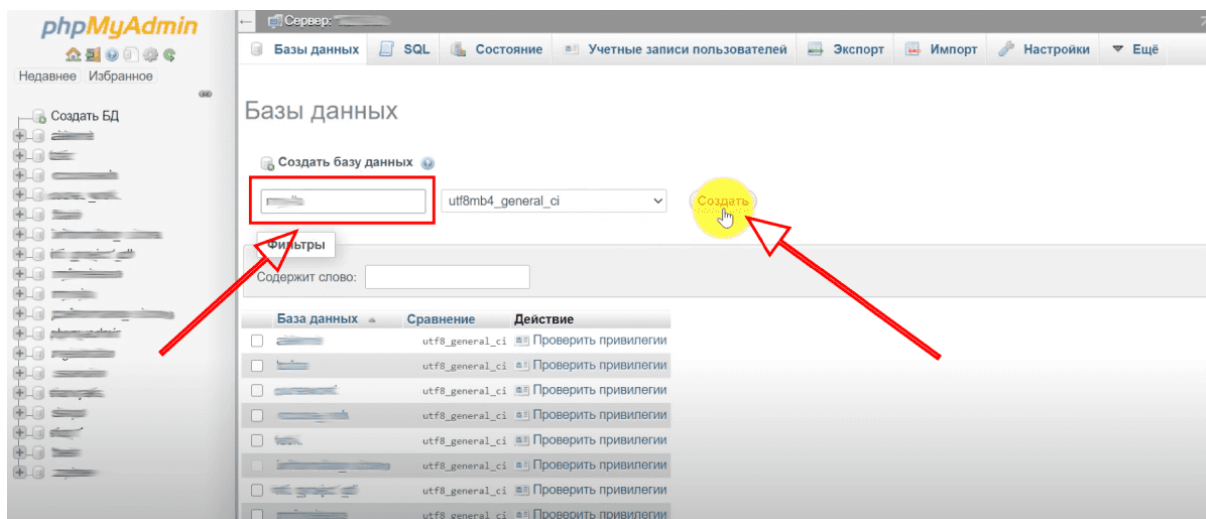


Рисунок 6.2 – Создание БД

В столбце слева отобразится новая база данных. Для примера создадим таблицу с пользователями — назовем ее «users». Количество столбцов зададим «4» (если вы планируете собирать больше данных о пользователях, то введите большее число). Нажимаем кнопку «Вперед» (рис. 6.3).

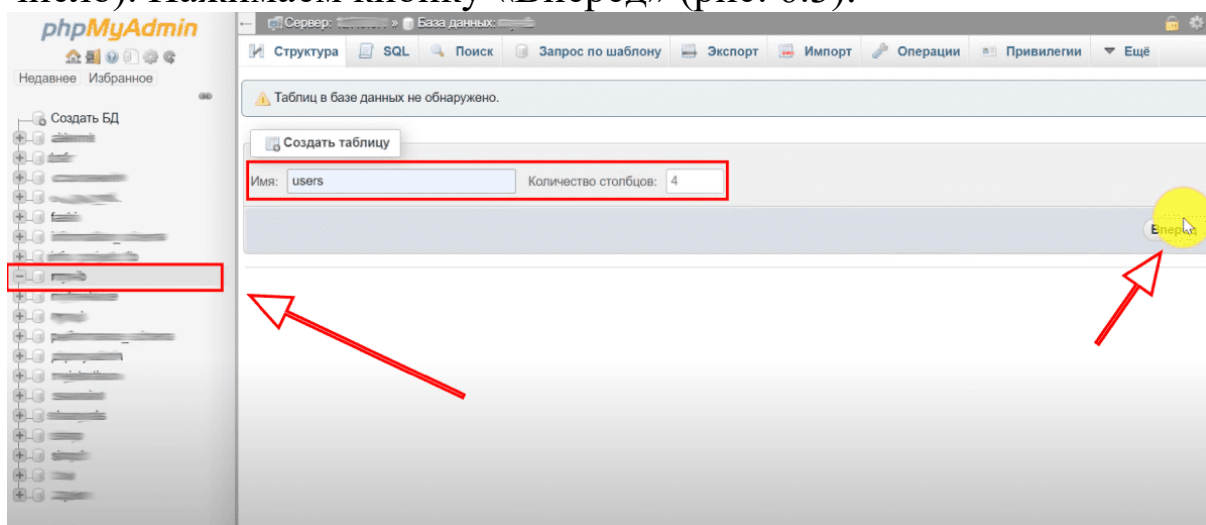


Рисунок 6.3 – Создание таблицы users

Теперь в режиме конструктора мы должны разработать структуру таблицы «users».

Первое поле — поле идентификации (userID). Тип — целочисленный (INT), так как это автоинкремент. Поэтому сразу укажем это, нажав на галочку в пункте «A...I». Имя индекса — «PRIMARY» (первичный ключ), так как значения в поле «userID» повторяться не будут. Нажимаем кнопку «Вперед» (рис. 6.4 – 6.5).

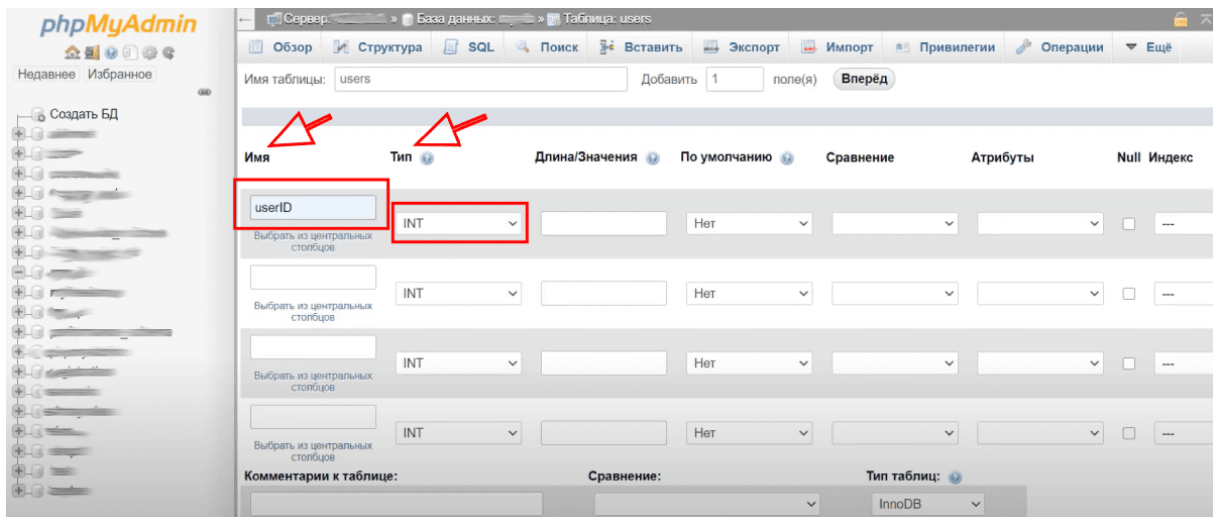


Рисунок 6.4 – Определение поля userID

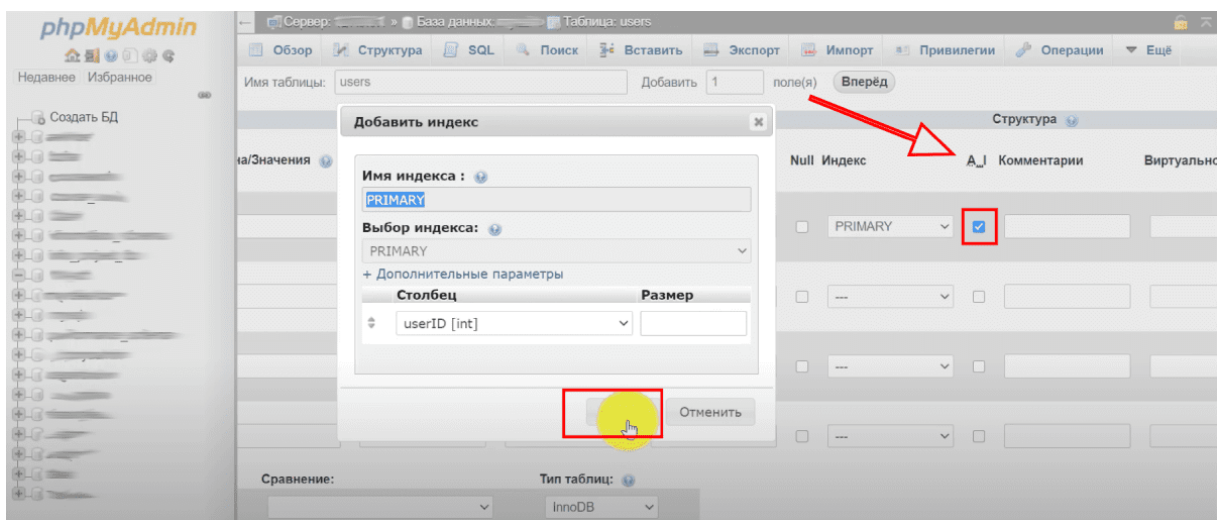


Рисунок 6.5 – Определение поля userID

Второе поле — userlogin. Тип — VARCHAR, то есть переменный набор символов. Зададим максимальное значение — 20. Добавим индекс: так как в нашей базе данных логины пользователей не должны повторяться, а первичный ключ у нас уже есть, то выбираем UNIQUE.

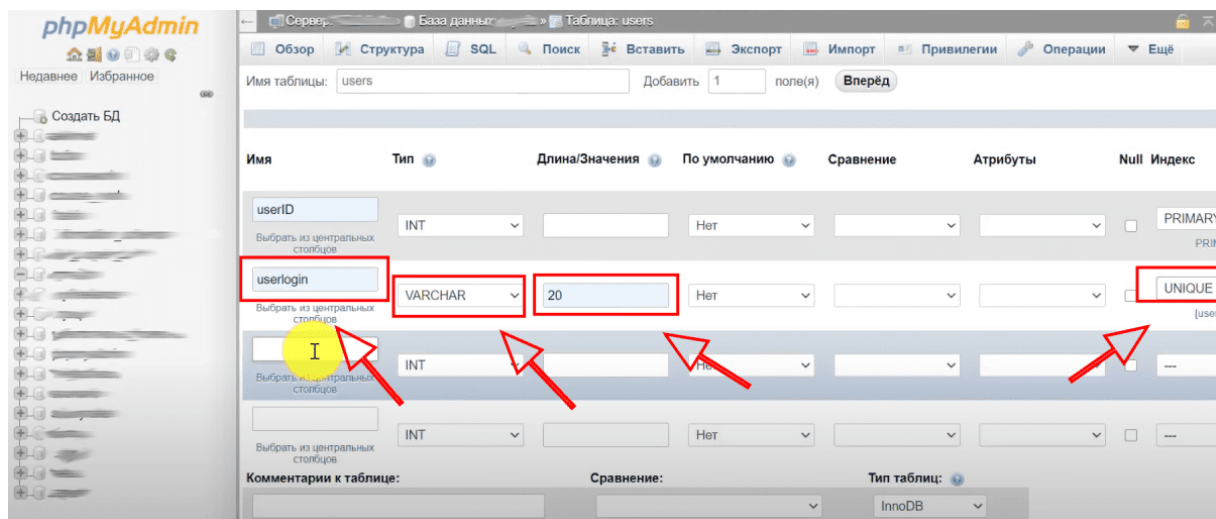


Рисунок 6.6 – Определение поля userlogin

Третье поле — пароли (userpassword). Тип — VARCHAR. Оставим максимальное значение — 20. Никаких свойств больше не применяем, так как пароли могут повторяться между пользователями (рис. 6.7).

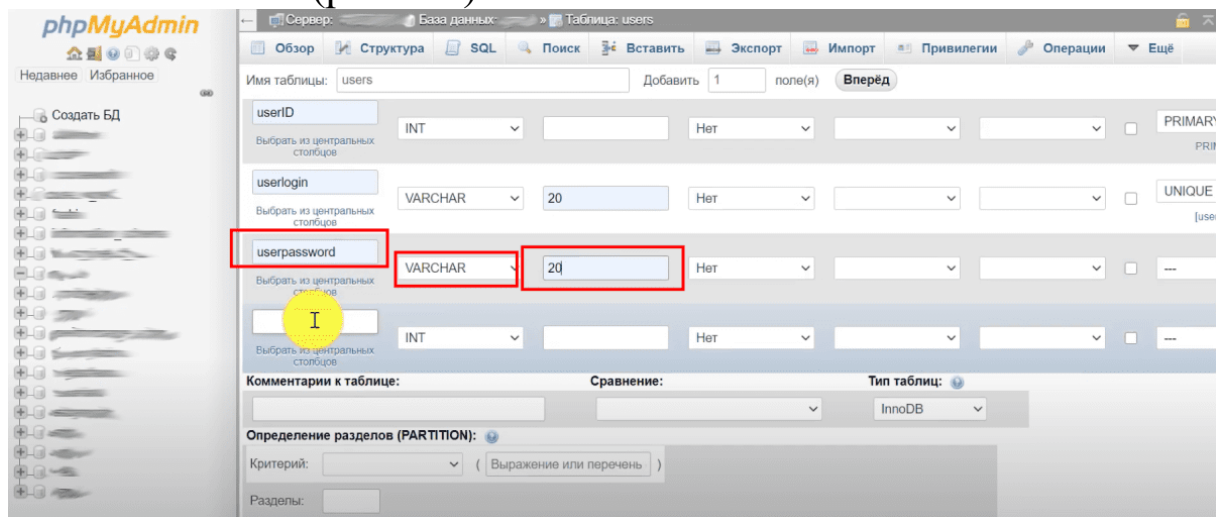


Рисунок 6.7 – Определение поля userpassword

Вы можете создать столько полей, сколько нужно для вашего проекта. Для примера мы ограничимся этими тремя полями. Нажимаем «Сохранить» (рис 6.8).

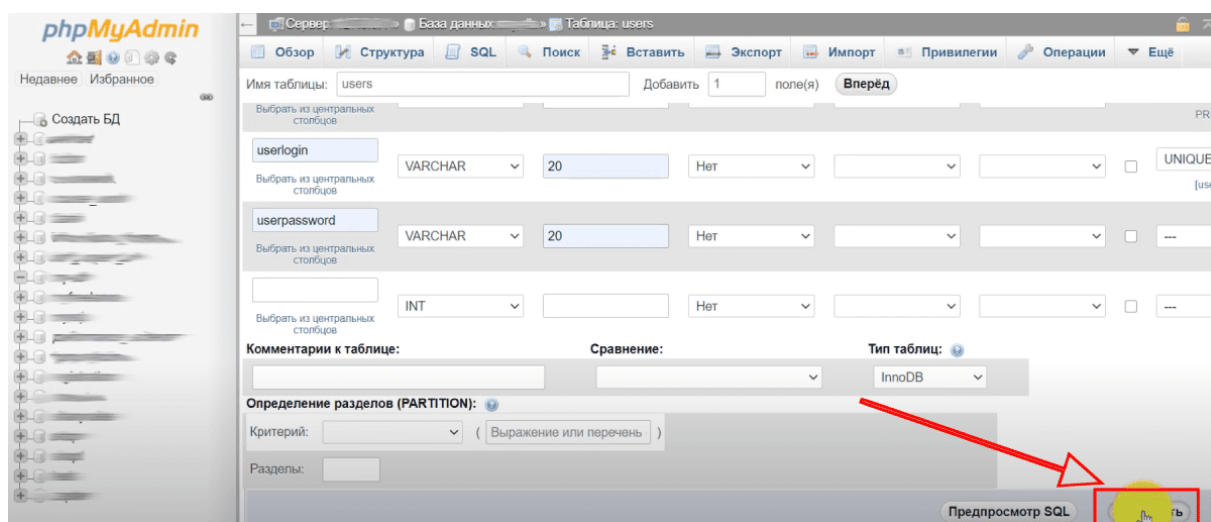


Рисунок 6.8 – Сохранение таблицы

Отлично, база данных создана успешно.

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №6

1. Установка и настройка phpMyAdmin:
 - Установите и настройте phpMyAdmin на вашем сервере или локальной машине.
 - Убедитесь, что вы имеете доступ к управлению базой данных через интерфейс phpMyAdmin.
2. Создание базы данных:
 - Зайдите в phpMyAdmin и создайте новую базу данных, назовите её в соответствии с темой вашего проекта.
 - Убедитесь, что создана пустая база данных, готовая к использованию.
3. Создание таблиц:
 - Определите структуру вашей базы данных, выделите основные сущности и их атрибуты.
 - Создайте таблицы в вашей базе данных, соответствующие этим сущностям, используя интерфейс phpMyAdmin.
 - Укажите правильные типы данных и ограничения для каждого атрибута таблицы.
4. Добавление данных:
 - Добавьте данные в созданные таблицы, используя функционал phpMyAdmin.
 - Убедитесь, что данные были успешно добавлены и отображаются корректно в таблице.
5. Выполнение запросов:

- Используйте интерфейс phpMyAdmin для выполнения SQL-запросов к вашей базе данных.
 - Выполните несколько простых запросов, таких как выборка данных из таблицы или изменение данных.
6. Тестирование и отладка:
- Протестируйте работу вашей базы данных, убедитесь, что данные хранятся и извлекаются корректно.
 - Выполните отладку, если необходимо, для устранения возможных ошибок или неполадок в работе базы данных.
7. Анализ работы:
- Оцените удобство использования phpMyAdmin для создания и управления базой данных MySQL.
 - Сделайте выводы о преимуществах и недостатках этого подхода к работе с базами данных.
8. Дополнительные задания:
- Изучите возможности phpMyAdmin для создания индексов, внешних ключей и других ограничений на таблицы.
 - Проведите анализ производительности вашей базы данных с использованием инструментов phpMyAdmin.
 - Изучите возможности резервного копирования и восстановления данных в phpMyAdmin и проведите соответствующие операции.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие основные компоненты используются для создания базы данных с помощью MySQL и phpMyAdmin?
2. Как создать новую базу данных в phpMyAdmin? Опишите процесс создания новой базы данных с использованием этого инструмента.
3. Какие параметры можно задать при создании новой базы данных в phpMyAdmin, и что они означают? Объясните значимость параметров "Имя базы данных" и "Кодировка".
4. Как создать новую таблицу в созданной базе данных с помощью phpMyAdmin? Опишите шаги создания таблицы и определения её структуры.
5. Как определить структуру таблицы в phpMyAdmin? Какие параметры определяются для каждого поля таблицы?

6. Какие типы данных могут быть использованы для определения полей таблицы в MySQL через phpMyAdmin? Приведите примеры типов данных.
7. Что такое первичный ключ (PRIMARY KEY) в таблице базы данных, и как его определить в phpMyAdmin?
8. Какие дополнительные параметры могут быть определены для полей таблицы в phpMyAdmin, и для чего они используются? Приведите примеры дополнительных параметров.
9. Как сохранить определение таблицы в phpMyAdmin после её создания? Какие действия необходимо выполнить после определения структуры таблицы?

ЛАБОРАТОРНАЯ РАБОТА №7. РАЗРАБОТКА МОДУЛЯ АВТОРИЗАЦИИ

Цель работы: создание функционального модуля авторизации на языке PHP для веб-приложения с целью обеспечения безопасного доступа пользователей к защищенным ресурсам.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Авторизация — это процесс проверки подлинности пользователя перед предоставлением доступа к защищенным ресурсам. Она обеспечивает безопасность и контроль доступа к информации веб-приложения.

Система авторизации будет основана на работе механизма сессий. Работа с сессиями включает в себя три основных этапа:

1. **Открытие сессии:** Старт сессии осуществляется на всех страницах, предполагающих работу с сессиями, с помощью функции `session_start()`. Этот этап необходим для инициализации сеанса веб-приложения, что позволяет сохранять данные между запросами пользователя.
2. **Регистрация сессионных переменных:** На этом этапе осуществляется регистрация сессионных переменных, которые будут использоваться для хранения данных в рамках текущей сессии. Это позволяет сохранять и передавать информацию между различными страницами в пределах одного сеанса.
3. **Разрегистрация сессионных переменных и закрытие сессии:** Этот этап включает в себя разрегистрацию сессионных переменных при помощи функции `unset()` и завершение сеанса с помощью функции `session_destroy()`. Это позволяет освободить ресурсы, занятые сессией, и завершить сеанс пользователя.

Шаг 1. Для создания необходимых файлов (`index.php`, `contact.php`, `admin.php`) проводится их инициализация. Ограничение доступа к файлу `admin.php` происходит путем ограничения расширения файла `.php`. Код каждого файла представляет собой базовое меню со ссылками на другие

страницы и индивидуальный текст для каждой страницы, обеспечивающий их различение.

Шаг 2. Для реализации ограничения доступа к странице администратора необходимо применить соответствующий алгоритм. При доступе к странице администратора первоначально производится проверка наличия заданной метки в сессии или, более просто, существования определенной сессионной переменной (также возможна проверка соответствия значения сессионной переменной определенному значению). Отсутствие указанной переменной свидетельствует о том, что пользователь, запрашивающий данную страницу, не авторизован, и в этом случае происходит перенаправление его на страницу авторизации, где ему предлагается заполнить форму с именем и паролем. Предложенный алгоритм прост в реализации и предполагает следующие шаги:

```
<?php
session_start();
if(!$_SESSION['admin']){
    header("Location: enter.php");
    exit;
}
?>
```

В данном фрагменте кода происходит начало сессии с помощью функции `session_start()`. Затем проводится проверка наличия переменной `$_SESSION['admin']`. В случае ее отсутствия происходит перенаправление пользователя на страницу авторизации (`enter.php`) с использованием функции `header()`. Функция `exit()` гарантирует прекращение дальнейшего выполнения скрипта после перенаправления.

Шаг 3. Создание страницы авторизации.

Для создания страницы авторизации с именем `enter.php` следует скопировать код из, например, страницы `contact.php`, создать новый файл и вставить в него скопированный код. Файл сохраняется под именем `enter.php`. На данной странице необходимо создать простую форму для ввода логина и пароля:

```
<p><a href="index.php">Главная</a> | <a href="contact.php">Контакты</a> | <a
href="admin.php">Админка</a></p>
<hr />
Это страница авторизации.
<br />
<form method="post">
```

```

Username: <input type="text" name="user" /><br />
Password: <input type="password" name="pass" /><br />
<input type="submit" name="submit" value="Войти" />
</form>

```

В форме присутствуют два поля: поле для ввода логина с именем "user" и поле для ввода пароля с именем "pass". Также создана кнопка с именем "submit", при нажатии на которую данные формы будут отправлены. Данные отправляются методом POST, что указано в атрибуте "method" тега формы, и будут обработаны на этой же странице. Теперь можно попытаться получить доступ к странице администратора. Если все выполнено без ошибок, доступ будет ограничен, и пользователь будет перенаправлен на страницу авторизации.

Шаг 4. Создание обработчика формы авторизации.

Для обработки данных, полученных из формы, и сравнения введенного логина и пароля с данными администратора, необходимо написать код на странице авторизации. Для этого открываем в верхней части страницы конструкцию РНР и начинаем кодировать. Прежде всего, нам следует инициировать сессию, поскольку именно здесь будет создаваться метка в сессии в случае успешной аутентификации. Также на этой же странице будем хранить логин и зашифрованный пароль администратора.

В данном случае логин установлен как "admin" и хранится в переменной \$admin, а пароль установлен как "mypass" и зашифрован с помощью функции md5(). Помним, что хранение паролей в открытом виде противоречит принципам безопасности, поэтому пароль хранится в зашифрованном виде. Для шифрования используется функция md5(), которая преобразует строку в хеш.

```

<?php
session_start();
$admin = 'admin';
$pass = 'a029d0df84eb5549c641e04a9ef389e5'; // зашифрованный пароль
?>
<p><a href="index.php">Главная</a> | <a href="contact.php">Контакты</a> | <a
href="admin.php">Админка</a></p>
<hr />
Это страница авторизации.
<br />
<form method="post">

```

```

Username: <input type="text" name="user" /><br />
Password: <input type="password" name="pass" /><br />
<input type="submit" name="submit" value="Войти" />
</form>

```

На данной странице представлена форма для ввода логина и пароля. Поля ввода имеют имена "user" и "pass" соответственно. После заполнения и нажатия кнопки "Войти" данные из формы будут отправлены методом POST для обработки на этой же странице.

Шаг 5. Проверка полученных данных из формы.

Теперь необходимо проверить, соответствуют ли данные, полученные из формы, логину и паролю, которые у нас хранятся в переменных. Для этого проверка будет осуществляться при условии, что кнопка формы была нажата. Для определения нажатия кнопки мы можем проверить наличие элемента с именем "submit" в массиве \$_POST. Если данный элемент существует, это означает, что кнопка была нажата, и мы можем провести проверку введенных данных. В обратном случае никакие действия выполняться не будут.

```

if($_POST['submit']){
    if($admin == $_POST['user'] AND $pass == md5($_POST['pass'])){
        $_SESSION['admin'] = $admin;
        header("Location: admin.php");
        exit;
    }else echo '<p>Логин или пароль неверны!</p>';
}

```

Условие проверки логина и пароля устроено двойным. Для этого используется логический оператор AND (или его альтернативная запись "&&"). Условие можно трактовать следующим образом: "если переменная \$admin равна значению элемента 'user' в массиве \$_POST И переменная \$pass равна хешу значения элемента 'pass' в массиве \$_POST, то выполняется блок действий, иначе выводится сообщение 'Логин или пароль неверны!'".

Если пара логин-пароль совпадает, то мы регистрируем сессионную переменную \$_SESSION['admin'] и перенаправляем пользователя на страницу администратора (admin.php).

Теперь проведем тестирование созданной функциональности. При вводе неверных данных в форму авторизации мы получим предупреждающее сообщение "Логин

или пароль неверны!". При вводе правильных данных мы будем перенаправлены на страницу администратора, при условии отсутствия ошибок в предыдущих шагах.

Шаг 6. Реализация защиты от доступа к странице авторизации для авторизованных пользователей.

Следующим шагом в разработке системы авторизации является обеспечение защиты от доступа к странице авторизации для пользователей, которые уже авторизованы в системе. Это необходимо для предотвращения возможности прямого доступа к форме авторизации через адресную строку браузера после успешной авторизации.

Для реализации этой функциональности мы можем использовать подход, аналогичный проверке наличия метки в сессии на странице администратора. Теперь, на странице `enter.php`, после инициализации сессии, мы добавим следующий код:

```
if($_SESSION['admin']){  
    header("Location: admin.php");  
    exit;  
}
```

Этот фрагмент кода проверяет наличие метки 'admin' в сессии. Если такая метка обнаруживается (то есть пользователь уже авторизован), происходит автоматическое перенаправление на страницу администратора (`admin.php`) с помощью функции `header()`. После этого выполнение скрипта прерывается с помощью функции `exit()`.

Таким образом, если авторизованный пользователь попытается получить доступ к странице авторизации через адресную строку браузера, он будет немедленно перенаправлен на страницу администратора. В то же время неавторизованный пользователь сможет свободно получить доступ к странице авторизации и ввести свои учетные данные.

Шаг 7. Реализация выхода авторизованного пользователя.

Далее, необходимо предусмотреть возможность выхода авторизованного пользователя из системы. Допустим, администратор завершил свою работу и желает выйти из учетной записи, чтобы предотвратить возможность использования его

учетных данных посторонними лицами. Для этого добавим на странице admin.php ссылку "Выход". Ссылка будет вести на эту же страницу, однако будет содержать дополнительный параметр. Параметр добавляется к URL страницы с помощью вопросительного знака:

```
<a href="admin.php?do=logout">Выход</a>
```

Эту ссылку можно разместить в любом удобном месте на странице. Что касается параметра "do", то он будет передан методом GET, который используется для передачи данных через URL. При использовании этого метода параметры добавляются к адресу и отделяются от него вопросительным знаком. Мы передаем один параметр с названием "do" и значением "logout".

Теперь перейдем к процессу разавторизации пользователя. При загрузке страницы admin.php мы можем проверить значение параметра "do" из массива \$_GET. Если это значение равно "logout", то мы просто удаляем сессионную переменную \$_SESSION['admin'] и разрушаем сессию с помощью функции session_destroy(). В результате, пользователь будет выведен из системы, и дальнейший доступ к защищенным ресурсам потребует повторной авторизации.

Таким образом, на странице admin.php добавим следующее условие после инициализации сессии (перед проверкой наличия метки):

```
if($_GET['do'] == 'logout'){
    unset($_SESSION['admin']);
    session_destroy();
}
```

Теперь мы можем протестировать функциональность выхода из системы, перейдя по ссылке "Выход". После выхода из системы мы будем перенаправлены на страницу авторизации, где снова увидим форму для ввода учетных данных. Пока мы не авторизуемся, доступ к странице администратора будет недоступен.

Шаг 8. Расширение механизма ограничения доступа к страницам.

Важным завершающим шагом в процессе разработки системы авторизации является возможность ограничения доступа не только к странице администратора, но и к любой

другой странице. Это можно достичь путем открытия сессии и проверки наличия соответствующей метки в ней. Чтобы избежать дублирования этого кода на каждой странице, его можно вынести в отдельный файл (например, auth.php), который затем будет подключаться на страницах, требующих ограничения доступа.

Содержимое файла auth.php будет следующим:

```
<?php
session_start();

if($_GET['do'] == 'logout'){
    unset($_SESSION['admin']);
    session_destroy();
}

if(!$_SESSION['admin']){
    header("Location: enter.php");
    exit;
}
?>
```

Теперь, вместо вставки этого кода на каждую страницу, мы просто подключаем файл auth.php на странице admin.php следующим образом:

```
<?php
require "auth.php";
?>
```

Таким образом, этот файл можно будет также подключать на любой другой странице, где требуется ограничить доступ по паролю.

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №7

1. Создание страницы для ввода данных пользователем:
 - Создайте HTML-форму с полями для ввода логина и пароля.
 - Определите метод передачи данных формы (обычно POST).
 - Убедитесь, что форма имеет атрибут action, который указывает на PHP-скрипт, обрабатывающий данные.
2. Обработка введенных данных:
 - Создайте PHP-скрипт, который будет принимать данные из формы.

- Используйте функции PHP для получения введенных пользователем данных.
 - Проверьте данные на корректность: например, убедитесь, что оба поля (логин и пароль) были заполнены.
 - Проверьте соответствие введенных данных данным в базе данных: выполните запрос к базе данных для проверки существования пользователя с таким логином и паролем.
3. Хеширование паролей:
- Перед сохранением пароля в базе данных, примените хеширование к паролю. Используйте функции PHP для генерации хеша.
 - Используйте алгоритмы хеширования, такие как bcrypt или Argon2, которые обеспечивают высокую стойкость к атакам перебора паролей.
4. Вывод результатов авторизации:
- Создайте отдельную страницу для отображения результатов авторизации.
 - Используйте механизм перенаправления (например, функцию header() в PHP) для перехода на эту страницу после обработки данных входа.
 - Выведите соответствующее сообщение о результате входа (успешно или неудачно).
5. Защита от атак:
- Реализуйте механизмы ограничения количества попыток входа и временной блокировки учетных записей после нескольких неудачных попыток.
 - Используйте подготовленные запросы или ORM для защиты от SQL-инъекций.
6. Тестирование:
- Протестируйте модуль на различных сценариях входа: успешный вход, неверный логин, неверный пароль и т.д.
 - Проверьте, что модуль корректно обрабатывает ошибки и атаки.
7. Документирование кода:
- Комментируйте код, объясняя каждый его участок и логику работы.

- Создайте документацию, описывающую структуру и функциональность разработанного модуля. Это может быть как отдельный файл, так и часть комментариев в коде.
8. Восстановление пароля:
- Разработайте механизм восстановления пароля через отправку временной ссылки на электронную почту пользователя.
 - Создайте страницу для ввода нового пароля после перехода по этой ссылке.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое авторизация и зачем она нужна в веб-приложениях?
2. Какие основные шаги нужно выполнить для реализации модуля авторизации на PHP?
3. Какие функции PHP обычно используются для работы с сессиями?
4. Чем отличается авторизация от аутентификации?
5. Какие методы безопасности следует учитывать при разработке модуля авторизации на PHP?
6. Какие механизмы защиты от атак типа "Brute Force" могут быть применены в модуле авторизации?
7. Как можно сохранить пароли пользователей в базе данных безопасным способом?
8. Как можно реализовать "запомнить меня" функционал в модуле авторизации?
9. Какие методы аутентификации бывают помимо логина и пароля, и как их можно реализовать в PHP?

ЛАБОРАТОРНАЯ РАБОТА №8. РАЗРАБОТКА МОДУЛЯ РЕГИСТРАЦИИ

Цель работы: разработка функционального модуля регистрации на языке PHP для веб-приложения с целью обеспечения возможности пользователям создавать учетные записи и получать доступ к защищенным ресурсам.

ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Система регистрации и авторизации необходима для любого сайта, который хранит информацию о своих пользователях. Такие системы используются на сайтах самой разнообразной тематики – от образовательных платформ, которые хранят сведения о прохождении обучающих курсов и оценках, до онлайн-магазинов, которые сохраняют историю покупок и адреса пользователей. В этой работе мы создадим форму регистрации.

Форма будет создана с помощью PHP и MySQL. Для создания формы регистрации используется таблица users (рис. 8.1). Составим список полей, которые мы должны будем принимать от пользователя для регистрации:

- nickname – должен быть уникальным и содержать только символы латинского алфавита и цифры;
- email – должен быть уникальным и быть корректным email-ом;
- password – должен быть не менее 8 символов (будет зашифрован и будет храниться в поле password_hash).

Все остальные поля будут заполняться значениями по умолчанию на стороне сервера.

| | | | | | | | | | |
|---|---|----|----------|-----------------|--------------|-------|---------------|------------|---------------------|
| ✓ Отображение строк 0 - 1 (2 всего, Запрос занял 0,0000 сек.) | | | | | | | | | |
| SELECT * FROM `users` | | | | | | | | | |
| <input type="checkbox"/> Профилирование [Построчное редактирование] [Изменить] [Анализ SQL за | | | | | | | | | |
| <input type="checkbox"/> Показать все Количество строк: 50 ▼ Фильтровать строки: Поиск в таблице Сортировать по индексу: Ниодного | | | | | | | | | |
| + Параметры | | | | | | | | | |
| ← T → | ▼ | id | nickname | email | is_confirmed | role | password_hash | auth_token | created_at |
| <input type="checkbox"/> | | 1 | admin | admin@gmail.com | 1 | admin | hash1 | token1 | 2018-06-26 21:59:20 |
| <input type="checkbox"/> | | 2 | user | user@gmail.com | 1 | user | hash2 | token2 | 2018-06-26 21:59:20 |

Рисунок 8.1. – Таблица users

Создадим контроллер для работы с пользователями:
src/MyProject/Controllers/UsersController.php

```
<?php
namespace MyProject\Controllers;
use MyProject\View\View;
class UsersController
{
    /** @var View */
    private $view;
    public function __construct()
    {
        $this->view = new View(__DIR__ . '/../../templates');
    }
    public function signUp()
    {
        echo 'здесь будет код для регистрации пользователей';
    }
}
```

Теперь давайте пропишем роутинг для данной странички.
Пусть это будет myproject.loc/users/register

```
<?php
return [
    '~^articles/(\d+)$~' => [\MyProject\Controllers\ArticlesController::class, 'view'],
    '~^articles/(\d+)/edit$~' => [\MyProject\Controllers\ArticlesController::class, 'edit'],
    '~^articles/add$~' => [\MyProject\Controllers\ArticlesController::class, 'add'],
    '~^users/register$~' => [\MyProject\Controllers\UsersController::class, 'signUp'],
    '~^$~' => [\MyProject\Controllers\MainController::class, 'main'],
];
```

Проверим работу написанного кода (рис. 8.2)

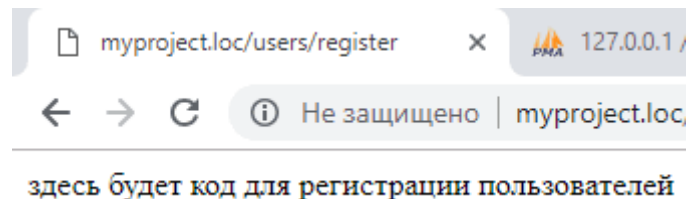


Рисунок 8.2 – Страница регистрации

Теперь создадим шаблон для страницы регистрации.
Напишем следующий код:

```
<?php include __DIR__ . '/../header.php'; ?>
<div style="text-align: center;">
    <h1>Регистрация</h1>
    <form action="/users/register" method="post">
        <label>Nickname <input type="text" name="nickname"></label>
        <br><br>
        <label>Email <input type="text" name="email"></label>
        <br><br>
        <label>Пароль <input type="password" name="password"></label>
```

```

        <br><br>
        <input type="submit" value="Зарегистрироваться">
    </form>
</div>
<?php include __DIR__ . '/../footer.php'; ?>

```

И теперь отобразим этот шаблон в контроллере, написанным ранее.

src/MyProject/Controllers/UsersController.php

```

<?php
namespace MyProject\Controllers;
use MyProject\View\View;
class UsersController
{
    /** @var View */
    private $view;
    public function __construct()
    {
        $this->view = new View(__DIR__ . '/../templates');
    }
    public function signUp()
    {
        $this->view->renderHtml('users/signUp.php');
    }
}

```

Проверим работу написанного кода (рис. 8.3).

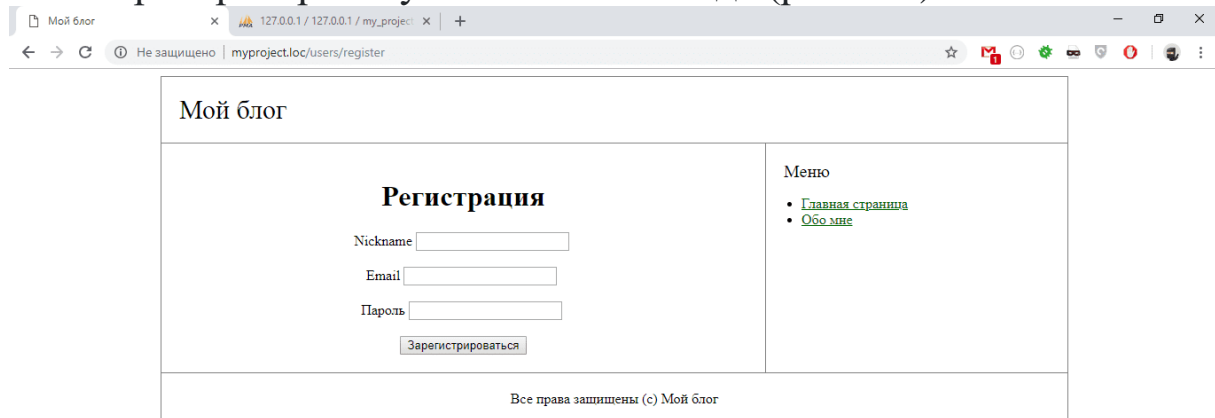


Рисунок 8.3 – Страница регистрации

Перейдем к программированию логики регистрации.

После подготовительного этапа переходим к основной задаче. Наша цель - обработать данные, полученные от пользователя. Прием данных из запроса будет осуществляться внутри контроллера, однако вся логика проверки этих данных

будет инкапсулирована в модели пользователя. Следовательно, принятие данных из запроса — это работа контроллера.

Далее задача контроллера - передать эти данные модели для выполнения определенных действий (например, проверки на валидность и сохранения в базе данных). Контроллеры не несут в себе бизнес-логику, то есть логику обработки и хранения данных. Благодаря тому, что такая логика всегда реализуется в моделях, код становится более гибким и повторно используемым в различных частях приложения. Например, мы можем обращаться к одному и тому же коду, хранящемуся в модели, из разных контроллеров.

Создадим в модели пользователя статический метод, который принимает массив с данными, полученными от пользователя, и пытается создать нового пользователя, сохраняя его в базе данных.

src/MyProject/Models/Users/User.php

```
...
public static function signUp(array $userData)
{
    var_dump($userData);
}
```

Именно этот код мы и будем вызывать в контроллере, если пришел POST-запрос.

src/MyProject/Controllers/UsersController.php

```
public function signUp()
{
    if (!empty($_POST)) {
        $user = User::signUp($_POST);
    }
    $this->view->renderHtml('users/signUp.php');
}
```

Пропишем проверку входных данных на пустоту.

В первую очередь необходимо обеспечить проверку наличия всех требуемых данных, переданных в запросе. В случае их отсутствия планируется генерация исключения для уведомления об этой ситуации. Предполагается выделить отдельное исключение для обработки подобных сценариев.

src/MyProject/Exceptions/InvalidArgumentException.php

```
<?php
namespace MyProject\Exceptions;
class InvalidArgumentException extends \Exception
{
```

```
}
```

Данный механизм будет применяться для обработки ситуаций, возникающих при передаче неверных параметров или данных. Продолжим с разработкой проверок наличия всех переданных данных.

```
src/MyProject/Models/Users/User.php
public static function signUp(array $userData)
{
    if (empty($userData['nickname'])) {
        throw new InvalidArgumentException('Не передан nickname');
    }
    if (empty($userData['email'])) {
        throw new InvalidArgumentException('Не передан email');
    }
    if (empty($userData['password'])) {
        throw new InvalidArgumentException('Не передан password');
    }
}
```

В контроллере теперь нужно научиться обрабатывать эти исключения:

```
src/MyProject/Controllers/UsersController.php
public function signUp()
{
    if (!empty($_POST)) {
        try {
            $user = User::signUp($_POST);
        } catch (InvalidArgumentException $e) {
            $this->view->renderHtml('users/signUp.php', ['error' => $e->getMessage()]);
            return;
        }
    }
    $this->view->renderHtml('users/signUp.php');
}
```

Обратите внимание, что мы начали передавать переменную "error" в шаблон. Важно иметь возможность выводить её содержимое в шаблоне, только если она не пустая.

```
templates/users/signUp.php
<?php include __DIR__ . '/../header.php'; ?>
<div style="text-align: center;">
    <h1>Регистрация</h1>
    <?php if (!empty($error)): ?>
        <div style="background-color: red;padding: 5px;margin: 15px"><? $error ?></div>
    <?php endif; ?>
    <form action="/users/register" method="post">
        <label>Nickname <input type="text" name="nickname"></label>
        <br><br>
```

```

<label>Email <input type="text" name="email"></label>
<br><br>
<label>Пароль <input type="password" name="password"></label>
<br><br>
<input type="submit" value="Зарегистрироваться">
</form>
</div>
<?php include __DIR__ . '/../footer.php'; ?>

```

Теперь попробуем отправить форму регистрации с пустыми полями (рис . 8.4).

| | |
|--|--|
| Мой блог | |
| <div style="text-align: center;"> <h2>Регистрация</h2> <div style="background-color: red; color: white; padding: 5px; margin: 10px 0;">Не передан nickname</div> <p>Nickname <input type="text"/></p> <p>Email <input type="text"/></p> <p>Пароль <input type="password"/></p> <p style="text-align: center;"><input type="submit" value="Зарегистрироваться"/></p> </div> | <div> <p>Меню</p> <ul style="list-style-type: none"> Главная страница Обо мне </div> |
| Все права защищены (с) Мой блог | |

Рисунок 8.4 – Отправка формы регистрации с пустыми полями

Система выводит сообщение об ошибке, указывающее на незаполненное поле электронной почты. Этот результат свидетельствует о корректной работе кода, который осуществляет проверку наличия необходимых полей. Однако стоит отметить, что данные, введенные пользователем перед отправкой формы, теряются, что представляет собой неудобство, так как требуется возвращение к предыдущему состоянию. Предлагается реализовать вывод в шаблоне данных, переданных в запросе, с использованием атрибута "value" у тегов "input".

```

<?php include __DIR__ . '/../header.php'; ?>
<div style="text-align: center;">
  <h1>Регистрация</h1>
  <?php if (!empty($error)): ?>
    <div style="background-color: red;padding: 5px;margin: 15px"><?= $error ?></div>
  <?php endif; ?>
  <form action="/users/register" method="post">
    <label>Nickname <input type="text" name="nickname" value="<?=
$_POST['nickname'] ?? ">" ?>"></label>
    <br><br>

```

```

<label>Email <input type="text" name="email" value="<?=$_POST['email'] ?? "
?>"></label>
<br><br>
<label>Пароль <input type="password" name="password" value="<?=$_POST['password'] ?? " ?>"></label>
<br><br>
<input type="submit" value="Зарегистрироваться">
</form>
</div>
<?php include __DIR__ . '/../footer.php'; ?>

```

Попробуем снова отправить форму с заполненным полем nickname (рис. 8.5)

| | |
|---|---|
| Мой блог | |
| <p style="text-align: center;">Регистрация</p> <div style="background-color: red; color: white; text-align: center; padding: 2px;">Не передан email</div> <p>Nickname <input type="text" value="kek"/></p> <p>Email <input type="text"/></p> <p>Пароль <input type="password"/></p> <p style="text-align: center;"><input type="button" value="Зарегистрироваться"/></p> | <p>Меню</p> <ul style="list-style-type: none"> • Главная страница • Обо мне |
| Все права защищены (с) Мой блог | |

Ошибка, связанная с незаполненным полем, все еще присутствует, однако данные формы сохранены. Это позволяет утверждать, что пользовательский интерфейс стал более удобным, поскольку в случае возникновения ошибки не требуется повторного ввода всех данных.

Проверка данных на валидность

Добавим также проверку на то, что длина пароля не менее восьми символов, а также что nickname содержит только допустимые символы, а email является корректным email-ом:

```

src/MyProject/Models/Users/User.php
public static function signUp(array $userData)
{
    if (empty($userData['nickname'])) {
        throw new InvalidArgumentException('Не передан nickname');
    }
    if (!preg_match('/^[a-zA-Z0-9]+$/', $userData['nickname'])) {

```

```

        throw new InvalidArgumentException('Nickname может состоять только из символов
латинского алфавита и цифр');
    }
    if (empty($userData['email'])) {
        throw new InvalidArgumentException('Не передан email');
    }
    if (!filter_var($userData['email'], FILTER_VALIDATE_EMAIL)) {
        throw new InvalidArgumentException('Email некорректен');
    }
    if (empty($userData['password'])) {
        throw new InvalidArgumentException('Не передан password');
    }
    if (mb_strlen($userData['password']) < 8) {
        throw new InvalidArgumentException('Пароль должен быть не менее 8 символов');
    }
}

```

Поиск дубликатов

Теперь стоит проверить, что пользователя с такими email и nickname нет в базе. Для этого нам понадобится метод, позволяющий находить записи в базе по одному столбцу. Давайте добавим его в класс ActiveRecordEntity.

```

src/MyProject/Models/ActiveRecordEntity.php
public static function findOneByColumn(string $columnName, $value): ?self
{
    $db = Db::getInstance();
    $result = $db->query(
        'SELECT * FROM ' . static::getTableName() . " WHERE " . $columnName . " = :value
LIMIT 1;',
        [':value' => $value],
        static::class
    );
    if ($result === []) {
        return null;
    }
    return $result[0];
}

```

Этот метод будет принимать два параметра:

- имя столбца, по которому искать;
- значение, которое мы ищем в этом столбце.

Если ничего не найдено – вернётся null. Если же что-то нашлось – вернётся первая запись.

С помощью этого метода мы сможем искать пользователей по email и nickname:

```

src/MyProject/Models/Users/User.php
public static function signUp(array $userData)

```



```

{
    ...
    if (static::findOneByColumn('nickname', $userData['nickname']) !== null) {
        throw new InvalidArgumentException('Пользователь с таким nickname уже
существует');
    }
    if (static::findOneByColumn('email', $userData['email']) !== null) {
        throw new InvalidArgumentException('Пользователь с таким email уже существует');
    }
}

```

После завершения проверок напомним процедуру, которая необходима чтобы создать нового пользователя.

public static function signUp(array \$userData): **User**

```

{
    // ... проверки
    $user = new User();
    $user->nickname = $userData['nickname'];
    $user->email = $userData['email'];
    $user->passwordHash = password_hash($userData['password'],
PASSWORD_DEFAULT);
    $user->isConfirmed = false;
    $user->role = 'user';
    $user->authToken = sha1(random_bytes(100)) . sha1(random_bytes(100));
    $user->save();
    return $user;
}

```

Рассмотрим подробнее использование параметра "authToken". Данный параметр представляет собой специально сгенерированный случайным образом токен, который будет использоваться для аутентификации пользователей. После успешной аутентификации на сайте не передается ни пароль пользователя, ни его хеш. Вместо этого используется только указанный токен, который является уникальным для каждого пользователя и не связан с его паролем, что усиливает безопасность системы.

В завершении метода производится сохранение нового пользователя в базе данных, а затем метод возвращает его. Отмечается, что тип возвращаемого значения метода теперь указан как "User".

Необходимо отметить, что в контроллере требуется получить результат работы данного метода и проверить, что был возвращен только что созданный пользователь. В случае успешного завершения операции контроллер уведомляет об этом на форме регистрации.

```

src/MyProject/Controllers/UsersController.php
public function signUp()
{
    if (!empty($_POST)) {
        try {
            $user = User::signUp($_POST);
        } catch (InvalidArgumentException $e) {
            $this->view->renderHtml('users/signUp.php', ['error' => $e->getMessage()]);
            return;
        }
        if ($user instanceof User) {
            $this->view->renderHtml('users/signUpSuccessful.php');
            return;
        }
    }
    $this->view->renderHtml('users/signUp.php');
}

```

Решено избежать загромождения основного шаблона и создать отдельный шаблон для успешной регистрации - "signUpSuccessful.php". Как можно заметить, именно этот шаблон используется в контроллере в случае успешного завершения операции.

```

templates/users/signUpSuccessful.php
<?php include __DIR__ . '/../header.php'; ?>
<div style="text-align: center;">
    <h1>Регистрация прошла успешно!</h1>
    Ссылка для активации вашей учетной записи отправлена вам на email.
</div>
<?php include __DIR__ . '/../footer.php'; ?>

```

Пробуем теперь заполнить форму корректными данными и отправляем запрос (рис. 8.6).

| | |
|---|---|
| Мой блог | |
| <p style="text-align: center;">Регистрация прошла успешно!</p> <p style="text-align: center;">Ссылка для активации вашей учетной записи отправлена вам на email.</p> | <p>Меню</p> <ul style="list-style-type: none"> • Главная страница • Обо мне |
| Все права защищены (с) Мой блог | |

Рисунок 8.6 – Отправка заполненной формы

Заглянем теперь в базу и убедимся, что действительно появился новый пользователь (рис. 8.7).

Сервер: 127.0.0.1:3306 » База данных: my_project » Таблица: users

Обзор Структура SQL Поиск Вставить Экспорт Импорт Привилегии Операции Триггеры

Отображение строк 0 - 2 (3 всего, Запрос занял 0.0000 сек.)

SELECT * FROM `users`

Профилирование [Построчное редактирование] [Изменить] [Анализ SQL запроса] [Создать PHP-код] [Обновить]

Показать все | Количество строк: 50 | Фильтровать строки: Поиск в таблице | Сортировать по индексу: Ниодного

+ Параметры

| | id | nickname | email | is_confirmed | role | password_hash | auth_token |
|--------------------------|----|----------|--------------------|--------------|-------|---|-----------------------------|
| <input type="checkbox"/> | 1 | admin | admin@gmail.com | 1 | admin | hash1 | token1 |
| <input type="checkbox"/> | 2 | user | user@gmail.com | 1 | user | hash2 | token2 |
| <input type="checkbox"/> | 3 | cheburek | cheburek@gmail.com | 0 | user | \$2y\$10\$09gHmq5x4k2zP78t4lUEOnfrmPj7/ScyoiPhePqkb/... | 9634efc1471825b938d7156e288 |

↑ ☐ Отметить все | С отмеченными: ☐ ☐ ☐

Рисунок 8.7 – Таблица users с новым пользователем

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №8

- Создание страницы для регистрации новых пользователей:
 - Разработка HTML-формы с полями для ввода данных, такими как логин, пароль, адрес электронной почты и т.д.
 - Указание атрибута `method="POST"` в теге `<form>` для безопасной передачи данных на сервер.
 - Создание кнопки "Зарегистрироваться", которая будет отправлять данные на сервер.
- Обработка и проверка введенных данных:
 - Создание PHP-скрипта для обработки данных, полученных из формы регистрации.
 - Валидация введенных данных, например, проверка на пустые поля, правильность формата электронной почты и длину пароля.
 - Проверка уникальности логина в базе данных, чтобы избежать конфликтов.
- Сохранение данных в базе данных:
 - Написание SQL-запроса для вставки данных нового пользователя в таблицу базы данных.
 - Использование подготовленных запросов для безопасной передачи данных и предотвращения атак на основе SQL-инъекций.
 - Обработка возможных ошибок при выполнении запроса и вывод сообщений об успешной или неудачной регистрации.
- Отправка уведомлений:
 - Разработка функции для отправки уведомлений пользователю, например, письма с подтверждением

регистрации или просто сообщения об успешной регистрации.

- Использование встроенных функций PHP для отправки электронной почты (например, `mail()`) или библиотеки стороннего SMTP-сервера (например, PHPMailer).

5. Тестирование модуля:

- Проведение тестирования модуля на различных сценариях, таких как успешная регистрация с корректными данными, попытка регистрации с уже существующим логином, пустым паролем и т.д.
- Проверка работы валидации данных и обработки возможных ошибок.

6. Документирование кода:

- Добавление подробных комментариев к коду, описывающих его структуру, функции и переменные.
- Создание краткой документации, которая объясняет, как использовать модуль регистрации, какие данные он ожидает и какие уведомления могут быть отправлены пользователям.

7. Дополнительное задание:

- Реализация механизма подтверждения регистрации через электронную почту, включая генерацию уникального токена или ссылки для активации учетной записи.
- Создание дополнительных страниц для обработки запросов на активацию учетной записи и вывода сообщений об успешной активации.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое модуль регистрации в веб-приложении и зачем он нужен?
2. Какие основные элементы содержит форма регистрации пользователя?
3. Каким образом можно проверить уникальность логина или адреса электронной почты в процессе регистрации?
4. Какие методы хеширования паролей вы знаете и почему именно они используются для хранения паролей в базе данных?

5. Какие функции PHP используются для работы с базой данных MySQLi и PDO, и как они отличаются друг от друга?
6. Как можно организовать отправку электронной почты из PHP, и какие моменты следует учесть при этом?
7. Как можно реализовать механизм подтверждения регистрации пользователя через электронную почту?
8. Какие этапы включает в себя обработка формы регистрации на серверной стороне?
9. Какие шаги нужно предпринять для тестирования модуля регистрации, и почему это важно для обеспечения качества приложения?

САМОСТОЯТЕЛЬНАЯ РАБОТА №1

Раздел 1. Определение потребностей клиента, проектирование и планирование веб- приложений.

Тема 1. Современные технологии разработки веб-приложений. Устройство и функционирование современных информационных ресурсов.

Тема 2. Основные требования, предъявляемые к дизайну графических интерфейсов, способам передачи информации в текстовом, графическом, звуковом, видео- и других мультимедийных форматах сети Интернет.

Тема 3. Стандарты оформления технической документации (ГОСТ 19.201-78 Техническое задание, требования к содержанию оформлению).

Тема 4. Программное обеспечение для планирования и организации работ с клиентами 1.UML; 2. IDEF, MS Project

ЗАДАНИЕ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ №1

1. Ответьте на контрольные вопросы в практической работе №1, практической работе №2 и практической работе №3.
2. Составьте отчет о самостоятельной работе.

САМОСТОЯТЕЛЬНАЯ РАБОТА №2

Раздел 2. Языки программирования для разработки клиентской части веб-приложений.

Тема 1. Общие понятия о языках сценариев. Основы JavaScript. Области использования. Методы встраивания сценариев. Редакторы кода. Консоль разработчика.

Тема 2. Общие понятия о языках сценариев. Основы JavaScript. Области использования. Методы встраивания сценариев. Редакторы кода. Консоль разработчика.

Тема 3. Управляющие конструкции: if – else if – else, цикл while. Операторы инкремента и декремента. Циклы: for, for ...in, for ...of. Цикл do while. Функции. Понятие функций. Декларация функций. Аргументы функции.

Тема 4. Объектная модель документа: DOM. Изменение документа. Стили и классы. Размеры и прокрутка элементов. Размеры и прокрутка окна. События и их обработка. Введение в браузерные события. Всплытие и погружение. Делегирование событий.

ЗАДАНИЕ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ №2

1. Ответьте на контрольные вопросы в практических работах №5 – №9 и лабораторным работам №1 – №4.
2. Составьте отчет о самостоятельной работе.

САМОСТОЯТЕЛЬНАЯ РАБОТА №3

Раздел 3. Языки программирования для разработки серверной части веб-приложений

Тема 3.1. Основы клиент-серверного взаимодействия в Internet. Установка Web-сервера. Создание сценариев на серверном ЯП. Типы данных, переменные, операторы.

Тема 3.2. Операции и управляющие конструкции: ветвления, циклы. Массивы. Функции серверного языка программирования. Регулярные выражения.

Тема 3.3. Запросы HTTP (GET, POST), параметры URL и формы HTML. Cookies и сессии серверного языка программирования.

Тема 3.4. Объектно-ориентированное программирование на серверном языке программирования. Средства серверного языка программирования для работы с базами данных.

ЗАДАНИЕ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ №2

1. Ответьте на контрольные вопросы в практических работах №10 – №16 и лабораторным работам №5 – №8.
2. Составьте отчет о самостоятельной работе.

СПИСОК ЛИТЕРАТУРЫ

1. Гаврилов, М. В. Информатика и информационные технологии: учебник для СПО / Гаврилов М. В., Климов В. А.. – 4-е изд., пер. и доп. – Москва : Юрайт, 2021. – 383 с. – ISBN 978-5-534-03051-8. – URL: <https://urait.ru/book/informatika-i-informacionnye-tehnologii-469424> (дата обращения: 12.02.2024). – Текст : электронный.
2. Голицына, О. Л. Информационные системы и технологии : Учебное пособие / О. Л. Голицына, Н. В. Попов И. И. Максимов. – Москва : НИЦ ИНФРА-М, 2023. – 400 с. – ISBN 978-5-00091-592-9. – URL: <https://znanium.com/catalog/document?id=427489> (дата обращения: 12.02.2024). – Текст : электронный.
3. Красильникова, О. И. JavaScript в разработке клиентской части веб-страниц : учебное пособие / О. И. Красильникова. — Санкт-Петербург : ГУАП, 2022. — 87 с. — ISBN 978-5-8088-1690-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/263951> (дата обращения: 12.02.2024). — Режим доступа: для авториз. пользователей.
4. Земцов, А. В. Анализ и проектирование микросервисной архитектуры в современных веб-приложениях / А. В. Земцов ; Чувацкий государственный педагогический университет им. И.Я. Яковлева, Физико-математический факультет, Кафедра информатики и информационно-коммуникационных технологий. – Чебоксары : б.и., 2020. – 73 с. : ил., табл. – Режим доступа: по подписке. – URL: <https://biblioclub.ru/index.php?page=book&id=578249> (дата обращения: 01.04.2024). – Текст : электронный.
5. Государев, И. Б. Введение в веб-разработку на языке JavaScript : учебное пособие / И. Б. Государев. — Санкт-Петербург : Лань, 2022. — 144 с. — ISBN 978-5-8114-3539-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/206588> (дата обращения: 12.02.2024). — Режим доступа: для авториз. пользователей.

6. Диков, А. В. Клиентские технологии веб-дизайна. HTML5 и CSS3 : учебное пособие / А. В. Диков. — Санкт-Петербург : Лань, 2022. — 188 с. — ISBN 978-5-8114-3822-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/206879> (дата обращения: 12.02.2024). — Режим доступа: для авториз. пользователей.