

Министерство науки и высшего образования Российской Федерации

Калужский филиал  
федерального государственного бюджетного образовательного  
учреждения высшего образования  
**«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»**  
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

Лабораторный практикум по дисциплине  
«Перспективные языки программирования»: учебное пособие

Калуга – 2022

Рецензенты:

зав. кафедрой «Информационно-вычислительные системы и  
технологии программирования»

КФ МГТУ им. Н.Э. Баумана, канд. техн. наук, доц. И.В. Чухраев

директор по исследованиям и развитию ООО «НПФ Эверест»,  
канд. физ.-мат. наук В.Ю. Кириллов

Утверждено Методической комиссией КФ МГТУ им. Н.Э. Баумана  
(протокол № X от XX.XX.2022 г., рег. номер XXX/МК2022-XX)

Белов Ю.С., Гришунов С.С.

Лабораторный практикум по дисциплине «Перспективные  
языки программирования»: учебное пособие / Ю.С. Белов, С.С.  
Гришунов – Калуга: КФ МГТУ им. Н.Э. Баумана, 2022. – 80 с.

В учебном пособие приведены теоретические сведения,  
примеры и задания для выполнения лабораторных работ по  
дисциплине «Кроссплатформенная разработка программного  
обеспечения».

Учебное пособие предназначено для студентов КФ МГТУ им.  
Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04  
«Программная инженерия».

©КФ МГТУ им. Н.Э. Баумана, 2022

©Белов Ю.С., 2022

©Гришунов С.С., 2022

## ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ .....	3
ВВЕДЕНИЕ .....	4
ЛАБОРАТОРНАЯ РАБОТА №1 СТАТИЧНЫЕ САЙТЫ .....	5
ЛАБОРАТОРНАЯ РАБОТА №2 ИСПОЛЬЗОВАНИЕ PHP .....	24
ЛАБОРАТОРНАЯ РАБОТА №3 JAVASCRIPT .....	32
ЛАБОРАТОРНАЯ РАБОТА №4 SESSION. BOOTSTRAP .....	45
ЛАБОРАТОРНАЯ РАБОТА №5 MVC. FLASK. JINJA2 .....	55
ЛАБОРАТОРНАЯ РАБОТА №6 REACTJS .....	67
СПИСОК ЛИТЕРАТУРЫ .....	79

## **ВВЕДЕНИЕ**

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Перспективные языки программирования» на кафедре «Программное обеспечение ЭВМ, информационные технологии» факультета «Информатика и управление» Калужского филиала МГТУ им. Н.Э. Баумана.

Требования к программному обеспечению:

1. Ubuntu 20.04/RedOs 7.2 или др. Linux-система
2. LibreOffice
3. VS code или аналог
4. Интерпретатор Python версии 3.8 и старше
5. Mozilla Firefox 86.0
6. Node.js (+npm)

## **ЛАБОРАТОРНАЯ РАБОТА №1**

### **СТАТИЧНЫЕ САЙТЫ**

**Цель работы:** получить навык разработки статических сайтов с использованием HTML5 и CSS3

Задачи:

1. Установить и настроить веб-сервер.
2. Разработать и разместить на веб-сервере статичный сайт, содержащий таблицу стилей.

**Результатами работы являются:**

1. Разработанный статичный сайт, содержащий HTML и CSS файлы
2. Подготовленный отчет.

## ПРОТОКОЛ HTTP

HTTP — это протокол клиент-серверного взаимодействия, позволяющий получать клиенту различные ресурсы, например, HTML-документы от HTTP-сервера (веб-сервера). Протокол HTTP лежит в основе обмена данными в Интернете. Полученный итоговый документ будет (может) состоять из различных поддокументов, являющихся частью итогового документа: например, из отдельно полученного текста, описания структуры документа, изображений, видео-файлов, скриптов и многого другого.

HTTP-запрос, отправляемый клиентом, состоит из глагола, обозначающего одно из действий, которое может выполнить сервер (GET, POST, OPTIONS, PUT, HEAD, DELETE, PATCH, TRACE, CONNECT), адрес запрашиваемого ресурса на сервере, версию протокола и набор возможных дополнительных заголовков, а также для ряда запросов тело запроса (полезная информация, например, значения заполненных форм). Пример HTTP-запроса:

```
GET / HTTP/1.1
Host: kf.bmstu.ru
Accept-Language: en
```

Ответ, формируемый сервером, содержит версию протокола, код ответа, фразу-состояние ответа, возможные дополнительные заголовки, для некоторых запросов возможна полезная нагрузка (какая-то передаваемая информация, например, HTML-разметка). Пример HTTP-ответа, формируемого сервером и передаваемого клиенту:

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2021 14:28:02 GMT
Server: Apache
Accept-Ranges: bytes
Content-Length: 29812
Content-Type: text/html
```

```
<!DOCTYPE html... (содержимое, общим размером 29812 байт)
```

В настоящий момент наиболее распространены следующие 3 веб-сервера:

### **Apache**

Веб-сервер Apache стоял у истоков развития мирового интернета, он до сих пор лидирует в мировом рейтинге популярности. За свою долгую жизнь (а Apache ведет свою историю с 1995 года) свободный веб-сервер оброс массой модулей и «научился» разворачиваться на всевозможных платформах. До 2005 года Apache широко использовался как единый сервер для всех задач — он выполнял роли и веб-сервера, и обратного прокси, и балансировщик нагрузки. Впрочем, сейчас его позиции пошатнулись — по мере увеличения трафика, количества подключений и объемов данных на страницах Apache перестал справляться с такой многозадачностью. Но это не значит, что Apache уже вышел из игры. Главное его преимущество — огромное количество подключаемых модулей. В сети можно найти библиотеки для любых задач, поэтому «Апач» с большой вероятностью подойдет для разработки необычного сайта. Кроме того, ненужные модули всегда можно отключить, чтобы повысить быстродействие.

### **Nginx**

Nginx — это веб-сервер с открытым исходным кодом. Если необходимо что-то в нем подправить, всегда можно скачать исходный код и подогнать его под себя. Но в большинстве случаев это не требуется — у Nginx и без того широкий функционал, способный удовлетворить потребности не только простеньких проектов, но и сложных сайтов с огромной посещаемостью. Nginx относится к легковесным серверам. При его разработке старались учесть все недостатки более старого Apache. Код сервера подразумевает более эффективное масштабирование — с увеличением потока подключений скорость работы почти не падает. Каждый рабочий процесс Nginx способен обрабатывать по тысячам HTTP-подключений сразу.

### **IIS**

IIS (Internet Information Services) — это веб-сервер, разработанный компанией Microsoft для своих операционных систем. Продукт полностью проприетарный и идет в комплекте с Windows. Первая

версия появилась в Windows NT и продолжает развиваться. IIS уступает Apache и Nginx, в виду архитектурной особенности и строгой работы на ОС Windows, но с другой стороны, как проприетарное решение, имеет поддержку производителя.

## HTML

HTML (HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык разметки документов. Большинство веб-страниц содержат описание разметки на языке HTML. Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Пример HTML-документа:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Тестовая страница</title>
  </head>
  <body>
    <p>Это — моя страница</p>
  </body>
</html>
```

В первой строке традиционно приводится объявление типа документа. Далее в теге `<html>` приводится сам HTML-документ, состоящий из двух составных частей – блока `<head>` и блока `<body>`.

В блоке `<head>` содержится метainформация о странице, т.е. необходимая для работы браузеру, но не присутствующая на самой рисуемой странице: информация о дополнительных зависимостях, иконки и заглавие страницы, сведения об используемой кодировке и другая метainформация.



Блок <body> содержит непосредственно «тело» страницы, т.е. саму гиперразметку текста.

Большинство структурированных текстов состоят из параграфов и заголовков, независимо от того, читаете ли вы рассказ, или газету, или учебник, журнал и т.д.

В HTML каждый абзац заключен в элемент <p>, подобно:

```
<p>Я параграф, да, это я.</p>
```

Каждый заголовок заключен в элемент заголовка <h1>:

```
<h1>Я заголовок истории.</h1>
```

Имеется шесть элементов заголовка: <h1>, <h2>, <h3>, <h4>, <h5> и <h6>. Каждый элемент представляет разный уровень контента в документе; <h1> представляет главный заголовок, <h2> представляет подзаголовки, <h3> представляет под-подзаголовки и так далее.

Для курсивного текста в HTML используется элемент <em> (выделение). Кроме того, чтобы сделать документ более интересным для чтения, они распознаются программами, считывающими с экрана, и произносятся другим тоном.

```
<p>Я <em>рад</em>, что ты не <em>опоздал</em>.</p>
```

Для полужирного текста в HTML используется элемент <strong> (важное значение). Помимо того, что документ становится более полезным, они распознаются программами, считывающими с экрана, и говорят другим тоном.

```
<p>Эта жидкость  
  <strong>очень токсична</strong>  
</p>  
<p>Я рассчитываю на тебя.  
  <strong>Не </strong>  
опаздывай!</p>
```

Также в HTML есть возможность создавать списки двух видов: нумерованные и ненумерованные (для каждого нового элемента будет использоваться стандартный для браузера разделитель):

Неупорядоченные:

```
<ul>
  <li>молоко</li>
  <li>яйца</li>
  <li>хлеб</li>
  <li>хумус</li>
</ul>
```

Упорядоченные

```
<ol>
  <li>Доедьте до конца дороги</li>
  <li>Поверните направо</li>
  <li>Езжайте прямо</li>
  <li>Поверните налево на третьем перекрестке</li>
  <li>Школа справа от вас, в 300 метрах</li>
</ol>
```

Для добавления в текст изображения необходимо использовать тег `<img>`, в атрибутах которого необходимо указать источник изображения и замещающий текст, отображаемый в случае недоступности изображения:

```

```

Для представления сложной структуры текста можно использовать таблицы. Таблица состоит из строк и столбцов ячеек, которые могут содержать текст и рисунки. Для добавления таблицы на веб-страницу используется тег `<table>`. Этот элемент служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются соответственно с помощью тегов `<tr>` и `<td>`. Таблица должна содержать хотя бы одну ячейку.

Допускается вместо тега <td> использовать тег <th>. Текст в ячейке, оформленной с помощью тега <th>, отображается браузером шрифтом жирного начертания и выравнивается по центру ячейки. В остальном, разницы между ячейками, созданными через теги <td> и <th> нет.

Для объединения двух и более ячеек в одну используются атрибуты colspan и rowspan тега <td>. Атрибут colspan устанавливает число ячеек объединяемых по горизонтали. Аналогично работает и атрибут rowspan, с тем лишь отличием, что объединяет ячейки по вертикали. Перед добавлением атрибутов проверьте число ячеек в каждой строке, чтобы не возникло ошибок. Пример:

```
<table border="1">
  <tr>
    <td colspan="2">Строка 1</td>
  </tr>
  <tr>
    <td>Столбец 1</td>
    <td>Столбец 2</td>
  </tr>
</table>
```

## Гиперссылки

Гиперссылки позволяют нам связывать документы с любым другим документом (или ресурсом), с которым необходимо. URL-адрес может указывать на файлы HTML, текстовые файлы, изображения, текстовые документы, видео и аудиофайлы и все остальное. Если веб-браузер не знает, как отображать или обрабатывать файл, он спросит, хотите ли вы открыть файл (в этом случае обязанность открытия или обработки файла передается в соответствующее локальное приложение на устройстве) или загрузить файл.

Простая ссылка создается путем оберты текста (или другого содержимого), который необходимо превратить в ссылку, в элемент <a>, и придания этому элементу атрибута href (который также известен

как гипертекстовая ссылка, или цель), который будет содержать веб-адрес, на который нужно указать ссылку:

```
<p>Ссылка на <a href="http://kf.bmstu.ru" title="Наш  
родной университет">сайт университета</a>.</p>
```

Можно ссылаться на определенную часть документа [HTML](#) (известную как фрагмент документа), а не только на верхнюю часть документа. Для этого вам сначала нужно назначить атрибут id элементу, с которым вы хотите связаться. Обычно имеет смысл ссылаться на определенный заголовок, поэтому это выглядит примерно так:

```
<h2 id="mail">Почтовый адрес</h2>
```

Затем, чтобы связаться с этим конкретным id, необходимо включить его в конец URL-адреса, которому предшествует знак решетки («якорь»), например:

```
<p>Хотите написать мне письмо? Используйте наш  
  <a href="contacts.html#mail">почтовый адрес</a>  
</p>
```

При наличии атрибута download браузер не переходит по ссылке, а предложит скачать документ, указанный в адресе ссылки.

```
<a download>Ссылка</a>
```

По умолчанию, при переходе по ссылке документ открывается в текущем окне или фрейме. При необходимости, это условие может быть изменено атрибутом target тега <a>

В качестве значения используется имя окна или фрейма, заданное атрибутом name. Если установлено несуществующее имя, то будет открыто новое окно. В качестве зарезервированных имен используются следующие:

- `_blank` – загружает страницу в новое окно браузера.
- `_self` – загружает страницу в текущее окно.
- `_parent` – загружает страницу во фрейм-родитель, если фреймов нет, то это значение работает как `_self`.
- `_top` – отменяет все фреймы и загружает страницу в полном окне браузера, если фреймов нет, то это значение работает как `_self`.

## Блочные и строчные элементы

Исторически HTML-элементы было принято делить на блочные и строчные. Блочные элементы занимают всю ширину своего родителя (контейнера), формально создавая «блок» (отсюда и название).

Браузеры обычно отображают блочные элементы с переводом строки до и после элемента. Блочные элементы можно представить в виде стопки коробок.

Блочные элементы: `<div>`, `<dl>`, `<dt>`, `<fieldset>`, `<figcaption>`, `<figure>`, `<footer>`, `<form>`, `<h1>` – `<h6>`, `<header>`, `<hgroup>`, `<hr>`, `<li>`, `<main>`, `<nav>`, `<ol>`, `<p>`, `<pre>`, `<section>`, `<table>`, `<ul>`, `<address>`, `<article>`, `<aside>`, `<blockquote>`, `<details>`, `<dialog>`, `<dd>`.

Все остальные элементы являются строчными и могут начинаться в любом месте строки.

## Нововведения HTML5

Поддержка аудио и видео, элементы для замены небезопасного flash:

- `<audio>`
- `<video>`

Рисование, элемент-холст:

`<canvas>`

Новые атрибуты форм, такие как `placeholder`, `require`, `novalidate`

Использование локального хранилища данных.

Media-запросы для загрузки адаптивных стилей.

Перечень HTML-элементов: <http://htmlbook.ru/html>

## СТИЛИЗАЦИЯ HTML. CSS

Элементам [HTML](#) можно задавать дополнительные настройки стиля с помощью атрибута style. Пример:

```
<p style="font-family:times; color:green;">  
    Этот текст написан зеленым цветом шрифтом Times.  
</p>
```

Однако, это неудобно при наличии на странице большого количества элементов, к которым должны быть применены одинаковые стили – дублирование кода увеличит размер страницы в разы. Для сокращения рекомендуется использовать CSS (Cascading Style Sheets). CSS можно описывать как в блоке <head>, так и в отдельном файле. Рекомендуется использовать второй вариант для сокращения передаваемых данных благодаря кешированию CSS-файлов браузером.

Как и HTML, CSS не является языком программирования. Но это и не язык разметки - это язык таблицы стилей. Это означает, что он позволяет применять стили выборочно к элементам в документах HTML.

Например, чтобы выбрать все элементы абзаца на HTML странице и изменить текст внутри них с черного на красный, можно использовать этот CSS:

```
p {  
    color: red;  
}
```

Структура CSS-правила:

селектор	свойство	значение
body	{ background:	#ffc910; }

Селектор определяет набор элементов, к которым будет применен набор последующих стилистических свойств. Через запятую можно перечислять несколько селекторов. Перечень возможных селекторов приведен в Таблице 1.1. В фигурных скобках указывается набор стилистических правил, состоящий из разделенных точкой с запятой пар из имени свойства и задаваемого ему значения.

Таблица 1.1. Типы селекторов

Имя селектора	Что выбирает	Пример
Селектор элемента (иногда называемый селектором тега или типа)	Все HTML элемент(ы) указанного типа.	p
		Выбирает все <p>
ID селектор	Элемент на странице с указанным ID на данной HTML. Лучше всего использовать один элемент для каждого ID (и конечно один ID для каждого элемента)	#my-id
		Выбирает элемент с аттрибутом id="my-id"
Селектор класса	Элемент(ы) на странице с указанным классом (множество экземпляров класса может объявляться на странице).	.my-class
		Выбирает все элементы с аттрибутом class="my-class"
Селектор атрибута	Элемент(ы) на странице с указанным атрибутом.	img[src]
		Выбирает <img src="myimage.png"> но не <img>
Селектор псевдо-класса	Указанные элемент(ы), но только в случае определенного состояния, например, при наведении курсора.	a:hover
		Выбирает <a>, но только тогда, когда указатель мыши наведен на ссылку.

## Псевдоклассы:

- `:link` – выберет любую ссылку, которая еще не была посещена, даже те, которые уже стилизованы
- `:visited` – позволяет вам выбирать ссылки, которые были посещены.
- `:active` – соответствует элементу в момент, когда он активируется пользователем (ТАВ)
- `:hover` – срабатывает, когда пользователь наводит на элемент мышью, но не обязательно активирует его
- `:focus` – применяется, когда элемент (такой как `input` формы) получает фокус. Обычно он активируется при клике мышью пользователем или при выборе элемента с использованием клавиши "tab" на клавиатуре.
- `:first-child` – находит любой элемент, являющийся первым в своем родителе.
- `:last-child` – любой элемент, являющийся последним в его родителе.
- `:nth-child()` – находит один или более элементов, основываясь на их позиции среди группы соседних элементов ( $2n$ ,  $2n+1$  и т.д.).
- `:nth-last-child()` – находит элемент, имеющий  $a+n-b-1$  потомков после данной позиции в дереве документа, значение для  $n$  может быть положительным или нулевым, а также имеющий родительский элемент
- `:nth-of-type()` – находит один или более элементов с заданным тегом, основываясь на их позиции среди группы соседних элементов.
- `:first-of-type` – находит первого потомка своего типа среди детей родителя.
- `:last-of-type` – находит последнего потомка с заданным тегом в списке детей родительского элемента.
- `:empty` – находит любой элемент, у которого нет потомков.



- :target – представляет уникальный элемент (целевой элемент) с подходящим id URL-фрагментом (модальное окно, вкладка и т.п.)
- :checked – находит отмеченные radio-buttons и check-buttons
- :enabled – находит любой включенный элемент. Элемент включен, если его можно активировать (например, выбрать, нажать на него или ввести текст) или поставить фокус. У элемента также есть отключенное состояние, когда его нельзя активировать или сфокусировать.
- :disabled – находит любой не включенный элемент
- :after – применяется для вставки назначенного контента после содержимого элемента. Этот псевдоэлемент работает совместно со стилевым свойством content, которое определяет содержимое для вставки.
- :before – по своему действию :before аналогичен псевдоэлементу :after, но вставляет контент до содержимого элемента.
- :first-letter – определяет стиль первого символа в тексте элемента, к которому добавляется. Это позволяет создавать в тексте буквицу и выступающий инициал.
- :first-line – определяет стиль первой строки блочного текста.

Если к одному элементу одновременно применяются противоречивые стилевые правила, то более высокий приоритет имеет правило, у которого значение специфичности селектора больше. Специфичность – это некоторая условная величина, вычисляемая следующим образом. За каждый идентификатор (в дальнейшем будем обозначать их количество через *a*) начисляется 100, за каждый класс и псевдокласс (*b*) начисляется 10, за каждый селектор тега и псевдоэлемент (*c*) начисляется 1. Складывая указанные значения в определенном порядке, получим значение специфичности для данного селектора.

Встроенный стиль, добавляемый к тегу через атрибут `style`, имеет специфичность 1000, поэтому всегда перекрывает связанные и

глобальные стили. Однако добавление `!important` перекрывает в том числе и встроенные стили.

Если два селектора имеют одинаковую специфичность, то применяться будет тот стиль, что указан в коде ниже.

## Нововведения CSS3

- Градиенты цветов
- Скругленные элементы
- Тени
- Анимация

Перечень доступных CSS-свойств: <http://htmlbook.ru/css>

## Flexbox

CSS Flexbox — это технология для создания сложных гибких макетов за счет правильного размещения элементов на странице.

Для создания макета необходимо поместить контент внутрь flexbox-контейнера:

```
.container {  
    display: flex; /* or inline-flex */  
}
```

Далее необходимо установить направление основной оси

```
.container {  
    flex-direction: row | row-reverse | column |  
column-reverse;  
}
```

Выравнивание «резиновых» компонентов:

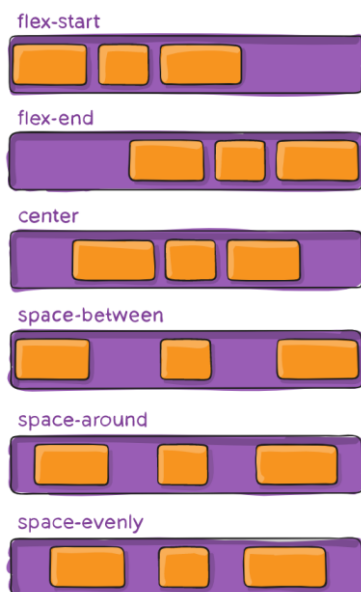


Рис. 1.1. Значение свойства `justify-content` для выравнивания вдоль основной оси

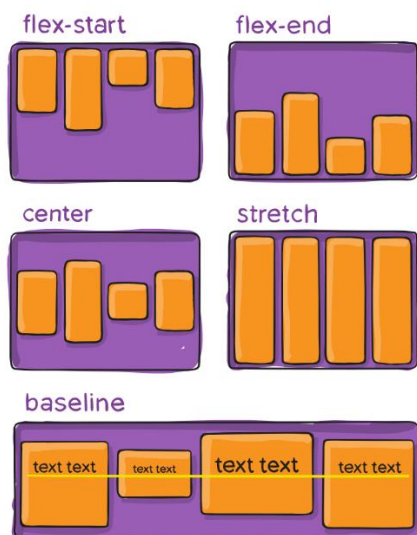


Рис. 1.2 Значения свойства `align-items` для выравнивания вдоль поперечной оси

Для автоматического переноса дочерних элементов можно использовать свойство `flex-wrap`:

```
flex-wrap: wrap;
```

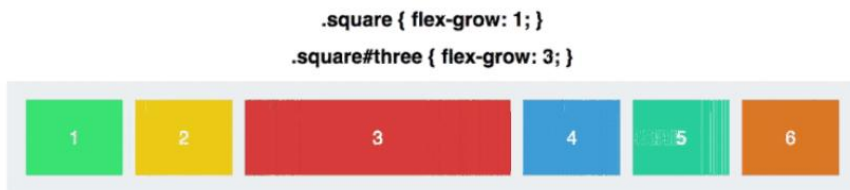
Можно сократить запись `flex-direction` и `flex-wrap` в одну строку, используя следующее свойство:

```
flex-flow: row wrap;
```

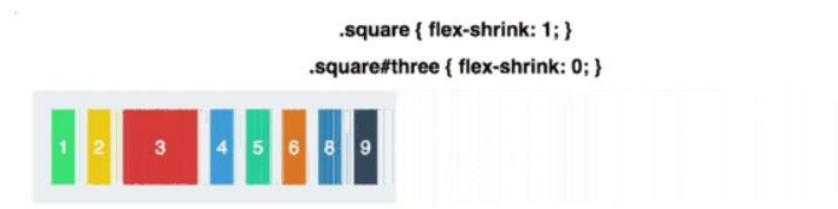
Свойство `flex-basis` отвечает за изначальный размер элементов до того, как они будут изменены другими свойствами CSS Flexbox:



Свойство `flex-grow` определяет на сколько блок может увеличиться в размерах (размеры относительные!). Значение 0 запрещает изменение размера блока.



Свойство `flex-shrink` определяет, насколько блоку можно уменьшиться в размере (размеры относительные!). Значение 0 запрещает изменение размера блока.



Свойство flex заменяет flex-grow, flex-shrink и flex-basis. Значения по умолчанию: 0 (grow) 1 (shrink) auto (basis):

```
.square#one {  
    flex: 2 1 300px;  
}
```

## ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Установить и сконфигурировать для локальной работы один из следующих веб-серверов на свой выбор: Apache, Nginx или IIS.

Разработать HTML-страницы статичного сайта тематики, согласно полученному варианту.

У сайта должны быть заданы заглавие и иконка сайта.

Сайт должен состоять минимум из 2-х страниц с возможностью перехода между ними.

Среди элементов обязательно наличие таблицы (с объединением некоторых ячеек), картинка, а также какой-либо из видов списков.

На сайте должна быть предусмотрена возможность скачивания произвольного PDF-документа.

Необходимо реализовать CSS-файл для стилизации сайта, обязательно должны использоваться селекторы идентификатора и класса, а также псевдокласса.

## **ВАРИАНТЫ ЗАДАНИЙ**

1. Сайт школы
2. Сайт фитнес-центра
3. Новостной сайт
4. Метеорологический сайт
5. Сайт музыкального исполнителя
6. Сайт кинотеатра
7. Сайт отеля
8. Сайт аэропорта
9. Сайт телеканала
10. Сайт радиостанции
11. Сайт автосалона
12. Сайт ресторана
13. Сайт университета
14. Сайт библиотеки
15. Сайт театра
16. Сайт ветеринарной клиники
17. Сайт туристической фирмы
18. Сайт агентства по продаже недвижимости
19. Сайт букмекерской компании
20. Сайт биржи криптовалют

## **КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ**

1. Приведите формат HTTP-запроса и HTTP-ответа.
2. Раскройте назначение веб-сервера.
3. Опишите формат HTML-документа.
4. Опишите различие между блочным и строчными элементами.
5. Опишите способ создания гиперссылки в HTML-документе.
6. Охарактеризуйте атрибуты id и class.
7. Приведите способы стилизации HTML-документов.
8. Опишите структуру CSS-правила.

9. Опишите и приведите примеры различным CSS-селекторам.
10. Дайте определение псевдоклассу. Приведите примеры
11. Опишите механизм приоретизации противоречивых CSS-правил.
12. Опишите механизм создания гибких макетов HTML-страниц

## **ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ**

На выполнение лабораторной работы отводится 5 часов: 4 часа на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Номер варианта студенту выдается преподавателем.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы (установки и настройки веб-сервера), исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.

## **ЛАБОРАТОРНАЯ РАБОТА №2**

### **ИСПОЛЬЗОВАНИЕ PHP**

**Цель работы:** получить навык разработки сайтов с использованием языка программирования PHP

**Задачи:**

1. Разработать сайт на PHP, отображающий информацию из файлов и позволяющий проводить запись в файл.
2. Разместить сайт на веб-сервере.

**Результатами работы являются:**

1. Разработанный сайт, содержащий PHP и CSS файлы.
3. Подготовленный отчет.



## СОЗДАНИЕ ДИНАМИЧЕСКИХ САЙТОВ С ПОМОЩЬЮ PHP

Статические сайты состоят из неизменяемых страниц. Это значит, что сайт имеет один и тот же внешний вид, а также одно и то же наполнение для всех посетителей. При запросе такого сайта в браузере сервер сразу предоставляет готовый HTML-документ, CSS и JS файлы.

Динамические сайты, в свою очередь, имеют изменяемые страницы, адаптирующиеся под конкретного пользователя. Такие страницы не размещены на сервере в готовом виде, а собираются заново по каждому новому запросу.

PHP (англ. PHP: Hypertext Preprocessor) — скриптовый язык программирования общего назначения, применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из самых распространенных среди языков программирования, применяющихся для создания динамических веб-сайтов.

PHP-скрипт в общем случае представляет из себя некий шаблон HTML-страницы, в которой с помощью специального тега `<?php>` добавляются вставки PHP-кода. Интерпретатор выполнит код на странице в порядке появления, в итоговой сгенерированной странице вместо каждого из блоков `<?php>` будет результат выполнения блока кода.

```
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <?php
      echo "Hello world!";
    ?>
  </body>
</html>
```

## Типы данных в PHP

Boolean – логический тип, который содержит значение true или false.

Integer – содержит значения целого числа.

String – содержит значение текста произвольной длины.

Float – вещественное число.

Object – объект.

Array – массив.

Resource – ресурс (например, файл).

NULL – значение NULL.

Для объявления переменных используется специальный знак «\$», команда echo возвращает значение из скрипта, например:

```
$text = "news"  
echo "This is {$text}paper";
```

Таблица 2.1. Логические операторы сравнения

<code>\$a == \$b</code>	Равно
<code>\$a === \$b</code>	Тождественно равно
<code>\$a != \$b</code>	Не равно
<code>\$a &lt;&gt; \$b</code>	Не равно
<code>\$a !== \$b</code>	Тождественно не равно

Для создания массивов можно использовать одну из следующих конструкций:

```
$arr1 = array("php", "html", "css");  
$arr2 = array(1 => "php", "html", "css");  
$arr3 = ["php", "laravel", "yii", "zend", "cakephp"];
```

Для удаления элемента из массива можно использовать метод unset:

```
unset($arr[1])
```

Для итерации по элементам массива можно использовать цикл foreach:

```
$array = array ("Apple", "Limon", "Chery", "Oranges");
foreach ($array as $value)
{
    echo "Вы выбрали фрукт - $value <br>";
}
```

## Работа с формами в PHP

Создание формы, где в атрибуте action указан PHP-скрипт обработчик формы:

```
<?php header("Content-Type: text/html;
    charset=utf-8");?>
<form action="app/check.php" method="post">
    <p>Имя: <input name="name" type="text"></p>
    <p>Фамилия: <input name="surname" type="text"></p>
    <p>E-mail: <input name="email" type="text"></p>
    <p>Сообщение: <br /><textarea name="message"
        cols="30" rows="5"></textarea></p>
    <p><input type='submit' value='Отправить'></p>
</form>
<?php>
```

Обработка полученной формы:

```
<?php
    if($_SERVER['REQUEST_METHOD'] == 'POST') {
        $name = $_POST['name'];
        $surname = $_POST['surname'];
        $email = $_POST['email'];
        $message = $_POST['message'];
        echo "name: $name <br/> surname: $surname
            <br/> email: $email
            <br/> message: $message";
    }
?>
```

## Работа с файлами в PHP

Для открытия файла используется функция `fopen`, принимающая в качестве параметра имя файла и режим доступа к файлу. Возможные значения режимов доступа:

- `r` – открытие файла только для чтения.
- `r+` – открытие файла одновременно на чтение и запись.
- `w` – создание нового пустого файла. Если на момент вызова уже существует такой файл, то он уничтожается.
- `w+` – аналогичен `r+`, только если на момент вызова файл такой существует, его содержимое удаляется.
- `a` – открывает существующий файл в режиме записи, при этом указатель сдвигается на последний байт файла (на конец файла).
- `a+` – открывает файл в режиме чтения и записи при этом указатель сдвигается на последний байт файла (на конец файла). Содержимое файла не удаляется.

Добавляя «`t`» работа с файлом будет происходить в текстовом режиме.

Примеры открытия файлов:

```
<?php
    $fp = fopen('counter.txt', 'r'); // Бинарный режим
    $fp = fopen('counter.txt', 'rt'); // Текстовый режим
    $fp = fopen("http://www.yandex.ru", "r");
    $fp = fopen("ftp://user:password@example.ru", 'w');
?>
```

Для чтения данных из файла можно воспользоваться одной из функций:

- `fgets($file)` – строчное чтение файла;
- `file_get_contents($file_name)` – загрузка всего содержимого файла;
- `fread($file, 100)` – чтение из файла указанного количества байт, переданного вторым параметром.

Для генерации исключения в случае невозможности открыть файл используется функция `die()`, принимающая на вход текст исключения. Примеры чтения данных из файлов:

```
<?php
    $fd = fopen("file.dat", 'r') or die("не удалось
        открыть файл");
    while(!feof($fd))
    {
        $str = fgets($fd);
        echo $str;
    }
    fclose($fd);

    $str = file_get_contents("file.dat");
    echo $str;

    $fd = fopen("file.dat", 'r') or die("не удалось
        открыть файл");
    while(!feof($fd))
    {
        $str = fread($fd, 600);
        echo $str;
    }
    fclose($fd);
?>
```

Запись в файл выполняется с помощью функции `fwrite`:

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

После того как работа с файлом завершена, его необходимо закрыть с помощью функции `fclose()`.

## **ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

Разработать сайт тематики, согласно полученному варианту.

Сайт должен быть стилизован с помощью CSS (возможно использование CSS-фреймворка Bootstrap).

На сайте должна быть отображена таблица, содержащая информацию, хранимую на веб-сервере в отдельном текстовом файле (возможно использование формата xml или json).

На сайте должна быть предусмотрена возможность добавления новых данных (с сохранением в файл).

### **ВАРИАНТЫ ЗАДАНИЙ**

1. Сайт школы
2. Сайт фитнес-центра
3. Новостной сайт
4. Метеорологический сайт
5. Сайт музыкального исполнителя
6. Сайт кинотеатра
7. Сайт отеля
8. Сайт аэропорта
9. Сайт телеканала
10. Сайт радиостанции
11. Сайт автосалона
12. Сайт ресторана
13. Сайт университета
14. Сайт библиотеки
15. Сайт театра
16. Сайт ветеринарной клиники
17. Сайт туристической фирмы
18. Сайт агентства по продаже недвижимости
19. Сайт букмекерской компании
20. Сайт биржи криптовалют

## **КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ**

1. Перечислите типы данных в PHP.
2. Приведите код для работы с массивами в PHP.
3. Приведите команду, с помощью которой можно вернуть значение из PHP-скрипта.
4. Приведите логические операторы сравнения в PHP.
5. Приведите функцию PHP для генерации исключения.
6. Опишите механизм обработки формы с помощью PHP.
7. Перечислите режимы доступа к файлам в PHP.
8. Опишите механизм работы с файлами с помощью PHP.

## **ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ**

На выполнение лабораторной работы отводится 5 часов: 4 часа на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы, исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.

## **ЛАБОРАТОРНАЯ РАБОТА №3**

### **JAVASCRIPT**

**Цель работы:** получить навык разработки с использованием языка программирования JavaScript

**Задачи:**

1. Разработать JS-скрипт, реализующий действие на странице в браузере согласно варианту
2. Добавить скрипт к странице.

**Результатами работы являются:**

1. Разработанный скрипт.
2. Подготовленный отчет.



## ИСПОЛЬЗОВАНИЕ JAVASCRIPT В HTML

JS – скриптовый язык. Скрипты могут встраиваться в [HTML](#) и выполняться браузером при загрузке. Также JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» (виртуальной машиной) JavaScript.

JavaScript может как встраиваться в HTML, так и быть вынесенным в отдельный .js файл (рекомендуемый подход). Пример использования JS:

```
<!DOCTYPE HTML>
<html>
  <head>
    <script src="/path//someScript.js"></script>
  </head>
  <body>
    <p>Перед скриптом...</p>
    <script>
      alert( 'Привет, мир!' );
    </script>
    <p>...После скрипта.</p>
  </body>
</html>
```

### Объявление переменных в JS

Область видимости переменных var ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока. Объявления (инициализация) переменных var производится в начале исполнения функции (или скрипта для глобальных переменных). Для более «традиционного» объявления переменных рекомендуется использовать ключевое слово let. В JS, как и в PHP динамическая типизация.

Типы данных:

- Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой.

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: Infinity, -Infinity и NaN. не может содержать числа больше, чем  $(2^{53}-1)$  (т. е. 9007199254740991), или меньше, чем  $-(2^{53}-1)$

- BigInt. Чтобы создать значение типа BigInt, необходимо добавить n в конец числового литерала:  
`const bigInt = 1234567890123456789012345678901234567890n;`
- Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь).
- Строка
- Специальное значение null не относится ни к одному из типов, описанных выше. Оно формирует отдельный тип, который содержит только значение null:
- Специальное значение undefined также стоит особняком. Оно формирует тип из самого себя так же, как и null. Оно означает, что «значение не было присвоено». Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined
- Объекты – используется для коллекций данных и для объявления более сложных сущностей. Объявляются объекты при помощи фигурных скобок {...}.
- Тип symbol (символ) используется для создания уникальных идентификаторов в объектах

Для определения типа переменной можно воспользоваться оператором typeof, у него есть две синтаксические формы:

- Синтаксис оператора: `typeof x`.
- Синтаксис функции: `typeof(x)`.

Однако, при использовании typeof есть несколько особенностей:

- При применении для функции будет возвращено значение function, несмотря на то что такого типа нет и функции представляют из себя объекты.

- При применении для значения `null` будет возвращено значение `object`, хотя `null` представляет из себя отдельный тип данных.

Также в JS есть ряд объектов-прототипов со своим набором методов для работы с данными: массив, словарь и др.

В JS используется прототипное наследование, работающее по следующему принципу: при обращении к полю объекта это поле ищется в самом объекте, если оно не находится, то поиск продолжается в объекте, указанном в поле `prototype` и т.д. по цепочке прототипов пока не будет найдено нужное поле или пока не найдется прототип, у которого не будет указано поле `prototype`.

## Функции

Функции в JS – это значения. Они могут быть присвоены, скопированы или объявлены в другом месте кода. Есть 2 способа задания функций: декларативный и функциональный:

```
//Function Declaration
function sayHi() {
    alert( "Привет" );
}

//Function Expression
let sayHi = function() {
    alert( "Привет" );
};
```

В отличие от других C-подобных языков ключевое слово `this` в JS вычисляется во время выполнения кода и зависит от контекста, т.е. объекту (а, значит, и функции) можно в любой момент выполнения задать необходимый контекст, к которому можно будет обратиться с помощью ключевого слова `this`. Задать контекст можно с помощью метода `bind(context, [params])`, либо можно вызвать функцию с определенным контекстом с помощью метода `call(context, [params])`.

В JS есть возможность создания анонимных функций, для этого можно использовать `arrow function` (стрелочные функции), которые

представляют из себя набор параметров знак `=>` и само тело функции. Важно: у стрелочных функций нет контекста! Кроме того, можно создать самовывзываемую функцию, для этого необходимо обернуть функцию в `()`, а затем указать ее входные параметры. Пример самовывываемой стрелочной функции:

```
((val) => console.log(val)) ("Привет")
```

## Работа с DOM в JS

После загрузки HTML/XML документы представляются в браузере в виде DOM-дерева (Document Object Model) – JS объекта, которым удобно управлять и с которым удобно производить изменения в отличие от текстового формата HTML/XML. Теги становятся узлами-элементами и формируют структуру документа. Текст становится текстовыми узлами. После построения DOM браузер будет работать именно с построенным объектом, а не с самим HTML.

В JS коде есть возможность получать доступ к элементам DOM для задания свойств и изменения самой структуры DOM. Для доступа к DOM используется глобальный объект `document`, например `document.body` – объект для тега `<body>`.

Между узлами DOM можно перемещаться, а также можно производить поиск элементов:

- `querySelector(selector)` возвращает первый элемент, который соответствует одному или более CSS селекторам. Если совпадения не будет, то он вернет `null`.
- `querySelectorAll()` возвращает все элементы, которые подходят под указанный CSS селектор. Подходящие элементы возвращаются в виде `NodeList` объекта, который будет пустым в случае того, если не будет найдено совпадений.
- `getElementById` – получение элемента по значению `id`. В качестве результата метод `getElementById` возвращает ссылку на объект типа `Element` или значение `null`, если элемент с указанным идентификатором не найден. Метод `getElementById` имеется только у объекта `document`/

- `getElementsByName` – получение элементов по значению атрибута `name`. Аналогично предыдущему, метод имеется только у объекта `document`.
- `getElementsByClassName` – получение списка элементов по именам классов. Этот метод позволяет найти все элементы с указанными классами во всём документе или в некотором элементе.
- `getElementsByName` – получение элементов по имени тега.

Найдя интересующие узлы DOM-дерева, с ними можно производить различные манипуляции:

#### 1. Получение и задание содержимого:

- `innerHTML` – внутреннее HTML-содержимое узла-элемента. Можно изменять.
- `outerHTML` – полный HTML узла-элемента. Запись в `elem.outerHTML` не меняет `elem`. Вместо этого она заменяет его во внешнем контексте.
- `textContent` – текст внутри элемента: HTML за вычетом всех тегов. Запись в него помещает текст в элемент, при этом все специальные символы и теги интерпретируются как текст.

#### 2. Получение и установка значений атрибутов:

- `elem.hasAttribute(name)` – проверяет наличие атрибута.
- `elem.getAttribute(name)` – получает значение атрибута.
- `elem.setAttribute(name, value)` – устанавливает значение атрибута.
- `elem.removeAttribute(name)` – удаляет атрибут.

#### 3. Создание элементов:

- `document.createElement(tag)` – создаёт новый элемент с заданным тегом
- `document.createTextNode(text)` – создаёт новый текстовый узел с заданным текстом

#### 4. Добавление информации к узлам:

- `node.append(...nodes or strings)` – добавляет узлы или строки в конец `node`.

- `node.prepend(...nodes or strings)` – вставляет узлы или строки в начало `node`.
- `node.before(...nodes or strings)` – вставляет узлы или строки до `node`.
- `node.after(...nodes or strings)` – вставляет узлы или строки после `node`.
- `node.replaceWith(...nodes or strings)` – заменяет `node` заданными узлами или строками.

#### 5. Удаление:

- `node.remove()`

#### 6. Клонирование:

- `elem.cloneNode(true)`

## События

Событие – это сигнал от браузера о том, что что-то произошло. Событию можно назначить обработчик, то есть функцию, которая работает, как только событие произошло. Примеры:

```
<input value="Нажми меня" onclick="alert('Клик!')"
      type="button">

<input id="elem" type="button" value="Нажми меня!">
<script>
    elem.onclick = function() {
        alert('Спасибо');
    };
</script>
element.addEventListener(event, handler[, options]);
```

По умолчанию для вложенных элементов применяется механизм всплытия: после обработки события целевым элементом событие передается вверх родительскому элементу и т.д. до верхнего элемента или до вызова метода `event.stopPropagation()`. Также можно включить механизм погружения (вначале вызываются обработчики родительских элементов, начиная с верхнего, потом целевого, а затем

повторяется всплытие. На каждом этапе можно получить фазу обработки события, обратившись к свойству `event.eventPhase`), для этого обработчику события необходимо указать свойство `{capture: True}`.

При обработке событий можно отключить поведение браузера по умолчанию (например, перезагрузку страницы после отправки формы) с помощью метода `event.preventDefault()`. Если обработчик назначен через `on<событие>` (не через `addEventListener`), то также можно вернуть `false` из обработчика.

### **Загрузка скриптов**

При указании JS-скрипта на странице, рендеринг страницы приостановится до того момента пока не загрузится указанный скрипт. Иногда это нерационально, поскольку у пользователя будет либо пустая, либо недорисованная страница, хотя на ее внешний вид скрипт может никак не влиять. Для асинхронной загрузки скрипта, т.е. загрузки «на фоне» без влияния на рендеринг страницы можно указать ключевое слово `defer`.

Для оптимизации загрузки страницы может потребоваться разбить скрипт на несколько составных частей. В таком случае все скрипты, загруженные с указанием ключевого слова `defer`, будут загружены последовательно. Если же скрипт абсолютно независим и его можно загружать асинхронно, то вместо `defer` нужно использовать ключевое слово `async`.

Скрипты можно загрузить и с помощью JS, такие скрипты по умолчанию будут загружены асинхронно:

```
let script = document.createElement('script');
script.src = "/long.js";
document.body.append(script); // (*)
```

### **Promises**

Для асинхронного выполнения кода можно использовать механизм промисов (от англ. `promise` – обещание). Промис представляет из себя объект, выполняющий некое асинхронное действие и имеющий

переданными параметрами 2 функции: функцию, которую необходимо выполнить при успешном завершении промиса и функцию, которую необходимо выполнить при ошибке во время выполнения. Самый простой пример промиса:

```
let promise = new Promise(function(resolve, reject) {
    setTimeout(() =>
        reject(new Error("Whoops!")), 1000);
});
```

Кроме того, из промисов можно выстраивать цепочки, используя потребитель промиса `then`, который также возвращает промис:

```
new Promise(function(resolve, reject) {
    setTimeout(() => resolve(1), 1000);
}).then(function(result) {
    alert(result);
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve(result * 2), 1000);
    });
});
```

#### Promise API:

- `let promise = Promise.all([...промисы...])` – ожидает пока выполнятся все промисы в массиве.
- `let promise = Promise.allSettled([...промисы...])` – ожидает успешного выполнения всех промисов в массиве
- `let promise = Promise.race([...промисы...])` – ожидает пока выполнится хоть один из промисов в массиве.
- `Promise.resolve(value)` – возвращает успешный промис из значения (может быть необходимо в цепочке промисов). Является устаревшим.
- `Promise.reject(value)` – возвращает неуспешный промис из значения (может быть необходимо в цепочке промисов). Является устаревшим.



Однако, вместо использования `.then()` рекомендуется использовать ключевые слова `async` и `await`. У слова `async` один простой смысл: эта функция всегда возвращает промис. Значения других типов оборачиваются в завершившийся успешно промис автоматически.

Ключевое слово `await` заставит интерпретатор JavaScript ждать до тех пор, пока промис справа от `await` не выполнится. После чего оно вернёт его результат, и выполнение кода продолжится. Например:

```
async function f() {
  let promise = Promise((resolve, reject) => {
    setTimeout(() => resolve("Ok!"), 1000);
  });
  let result = await promise;
}
```

## Работа с Canvas

`<canvas>` — это HTML5 элемент, использующийся для рисования графики средствами JavaScript. В скрипте, получив контекст `canvas` и задав стили рисования (например, цвет, толщину линий, разделитель и др.) можно рисовать примитивы (линия, прямоугольник, дуга и др.):

```
<head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
    function draw() {
      const canvas = document
        .getElementById('canvas');
      if (canvas.getContext) {
        const ctx = canvas.getContext('2d');
        ctx.fillStyle = 'rgb(200, 0, 0)';
        ctx.fillRect(5, 5, 20, 20);
        ctx.fillStyle = 'rgba(0, 0, 90, 0.5)';
        ctx.fillRect(30, 30, 50, 50);
      }
    }
  </script>
</head>
<body onload="draw();">
  <canvas id="canvas" width="150" height="150"></canvas>
</body>
```

Для рисования фигур необходимо вызвать метод `beginPath`, после чего можно указывать точки фигуры, с помощью метода `stroke` будет нарисована ломанная, с помощью метода `fill` можно нарисовать закрашенную фигуру:

```
ctx.beginPath();  
ctx.moveTo(5 + i * 14, 5);  
ctx.lineTo(5 + i * 14, 140);  
ctx.stroke();
```

Для рисования дуг применяется метод `arc`, например, для рисования окружности:

```
ctx.arc(100, 100, 75, 0, getRadians(360));
```

Перед рисованием к фигурам можно применить преобразования поворота и сдвига:

```
ctx.rotate(angle);  
ctx.translate(105, 0);
```

Для очистки холста используется метод `clearRect()`, для очистки всего экрана можно воспользоваться:

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

Для создания анимации необходимо с некоторой периодичности очищать экран и перерисовывать изображение, этого можно добиться, например, с помощью функции `window.requestAnimationFrame(draw)`, в качестве параметра передается функция отрисовки (частота обновления зависит от браузера, чаще всего 60 раз в секунду).

При отрисовке сложной сцены может потребоваться множество раз задавать одни и те же свойства рисования, чтобы этого избежать, свойства можно сохранять в стек с помощью метода `ctx.save()`, а затем восстанавливать с помощью метода `ctx.restore()`;

## **ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

Разработать скрипт согласно варианту, полученному у преподавателя.

### **ВАРИАНТЫ ЗАДАНИЙ**

1. Разработать браузерную игру «Крестики-нолики».
2. Разработать браузерное приложение для подсчета интегралов задаваемых полиномиальных функций в указанных промежутках методами прямоугольников и трапеций.
3. Разработать браузерную игру «Змейка»
4. Разработать браузерное приложение для решения СЛАУ методом Гаусса.
5. Разработать браузерное приложение для интерполяции значений функции по заданным точкам методом наименьших квадратов.
6. Разработать браузерное приложение для кластеризации введенных двумерных точек методом k-means.
7. Разработать браузерную игру «Космические захватчики»
8. Разработать браузерное приложение для подсчета изолированных комнат в заданном лабиринте.

### **КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ**

1. Перечислите типы данных в JS.
2. Опишите назначение и приведите синтаксис оператора typeof.
3. Приведите особенности оператора typeof.
4. Сравните использование ключевых слов var и let.
5. Опишите механизм прототипного наследования.
6. Приведите синтаксис создания функции.
7. Охарактеризуйте контекст в js.
8. Опишите механизм самовывзываемых функций.

9. Охарактеризуйте DOM.
10. Перечислите и охарактеризуйте методы для поиска элементов в DOM.
11. Перечислите действия, которые можно совершать с элементами DOM.
12. Приведите пример обработки событий в JS.
13. Опишите процедуру всплытия и погружения события
14. Приведите пример асинхронной загрузки JS-скриптов.
15. Опишите механизм работы промисов.
16. Приведите пример создания цепочки промисов.
17. Приведите пример рисования примитивов на canvas с помощью JS.
18. Опишите механизм создания анимации на canvas с помощью JS.

## **ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ**

На выполнение лабораторной работы отводится 7 часов: 6 часов на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы, исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.

## **ЛАБОРАТОРНАЯ РАБОТА №4**

### **SESSION. BOOTSTRAP**

**Цель работы:** получить навык разработки сайтов с использованием языка программирования PHP

**Задачи:**

1. Разработать приложение на PHP, использующий сессии для хранения информации о текущем пользователе.
2. Реализовать динамическую загрузку данных при помощи AJAX.
3. Реализовать веб-интерфейс приложения с помощью CSS-фреймворка Bootstrap.

**Результатами работы являются:**

1. Разработанный сайт, содержащий PHP, JS и CSS файлы.
2. Подготовленный отчет.

## СЕССИИ

Протокол HTTP(S) не поддерживает сессионность, т.е. каждый приходящий запрос на сервер рассматривается как изолированный, вне какого-то контекста. Иногда возникает необходимость связать ряд запросов в одну цепочку, например, запросы одного пользователя, для того чтобы разграничить права доступа и обеспечить отображение корректной информации. Для достижения такой цели необходимо использовать дополнительные механизмы.

Cookie (куки, печенки) — простые текстовые файлы на клиентском компьютере, по файлу для каждого домена (некоторые браузеры используют для хранения SQLite базу данных), при этом браузер накладывает ограничение на количество записей и размер хранимых данных (для большинства браузеров это 4096 байт). Для установки cookie с помощью PHP необходимо воспользоваться функцией:

```
int setcookie (string name [, string value [, int expire  
[, string path [, string domain [, int secure]]]])
```

Любые cookies, отправленные серверу браузером клиента, будут автоматически включены в суперглобальный массив `$_COOKIE` для обращения к ним со стороны сервера.

Чтобы удалить cookie достаточно в качестве срока действия указать какое-либо время в прошлом.

На основе cookies работает механизм поддержания сессий: при авторизации пользователя, сервер генерирует и запоминает уникальный ключ — идентификатор сессии, и сообщает его браузеру; браузер сохраняет этот ключ в cookie, и при каждом последующем запросе, его отправляет; проверяя по словарю значение полученной cookie, сервер может восстановить хранимое на сервере в отдельном файле содержимое сессии текущего пользователя.

В PHP уже реализован данный механизм: данные между запросами сохраняются в суперглобальном массиве `$_SESSION`. В тот момент, когда посетитель получает доступ к сайту, PHP проверяет автоматически (если `session.auto_start` установлено в 1) или по запросу

(явным образом через вызов `session_start()`), был ли определенный идентификатор сессии послан вместе с запросом. Если это так, восстанавливается сохраненное ранее окружение.

Пример работы с сессией в PHP:

```
//index.php
<?php header("Content-Type: text/html;
    charset=utf-8");
    session_start();

    if (!isset($_SESSION['count'])) {
        echo "Открытие сессии";
        $_SESSION['count'] = 0;
    } else {
        echo "Счетчик = " . $_SESSION['count'];
        $_SESSION['count']++;
    }
?>

<form action="destroy.php" method="post">
    <input type="submit" value="Убить сессию" />
</form>

//destroy.php
<?php header("Content-Type: text/html;
    charset=utf-8");
    session_start();
    session_destroy();
?>

<a href="test.php">Назад</a>
```

Также при работе с сессиями можно использовать дополнительные функции:

- `session_cache_expire` — вернуть текущее время жизни кеша
- `session_cache_limiter` — получить и/или установить текущий режим кеширования

- `session_create_id` — создаёт новый идентификатор сессии
- `session_decode` — декодирует данные сессии из закодированной строки сессии
- `session_destroy` — уничтожает все данные сессии
- `session_encode` — кодирует данные текущей сессии в формате строки сессии
- `session_gc` — выполняет сборку мусора данных сессии
- `session_get_cookie_params` — возвращает параметры cookie сессии
- `session_id` — получает и/или устанавливает идентификатор текущей сессии
- `session_is_registered` — определяет, зарегистрирована ли глобальная переменная в сессии
- `session_module_name` — возвращает и/или устанавливает модуль текущей сессии
- `session_name` — получить или установить имя текущей сессии
- `session_regenerate_id` — генерирует и обновляет идентификатор текущей сессии
- `session_register_shutdown` — функция завершения сессии

## **AJAX**

AJAX — это аббревиатура, которая означает Asynchronous Javascript and XML.

В классической модели веб-приложения пользователь заходит на веб-страницу, совершает какое-то действие, после чего браузер формирует и отправляет запрос серверу. В ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру и т. д., после чего браузер полностью перезагружает всю страницу.

При использовании AJAX пользователь заходит на веб-страницу и по совершению какого-то действия скрипт (на языке JavaScript) определяет, какая информация необходима для обновления страницы, браузер отправляет соответствующий запрос на сервер. Сервер



возвращает только ту часть документа, на которую пришёл запрос. Скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы), например, добавляет некоторые элементы в [DOM](#).

Пример AJAX запроса с помощью нативного интерфейса fetch (fetch возвращает промис):

```
const response = await fetch(url, {
  method: 'POST', // *GET, POST, PUT, DELETE, etc.
  mode: 'cors', // no-cors, *cors, same-origin
  cache: 'no-cache', // *default, no-cache, reload,
    // force-cache, only-if-cached
  credentials: 'same-origin', // include,
    // *same-origin, omit
  headers: {
    'Content-Type': 'application/json'
  },
  redirect: 'follow', // manual, *follow, error
  referrerPolicy: 'no-referrer', // no-referrer,
    // *client
  body: JSON.stringify(data)
})
  .catch((error) => {
    console.log(error)
  });

if (response.ok) {
  return response.json();
}
```

## BOOTSTRAP

Bootstrap — это открытый и бесплатный CSS фреймворк (конечно, не единственный в мире), который используется веб-разработчиками для быстрой верстки адаптивных дизайнов сайтов и веб-приложений.

В CSS Bootstrap содержит набор готовых правил для элементов различных классов: все классы имеют единую стилистику и приятную, легко изменяемую, цветовую схему (<https://getbootstrap.com/docs/5.2/components>).

Для ряда элементов, обладающих некоторым поведением, таких как выпадающие списки или radio-buttons, помимо css файлов необходимы соответствующие js-файлы.

Используя набор готовых правил можно существенно ускорить разработку, не отвлекаясь на дизайн, однако, нужно понимать, что на выходе получится шаблонный интерфейс, к тому, скорее всего с набором лишних зависимостей.

Кроме готового «дизайна» Bootstrap позволяет создавать адаптивную верстку с помощью специальной системы сеток.

Система сеток Bootstrap использует контейнеры, ряды и колонки, чтобы удобно располагать содержимое.

.container является корневым блоком сетки в Bootstrap, то есть располагается на внешнем уровне.

```
<div class="container">
  <div class="row">
    <div class="col">
      Этот текст внутри сетки Bootstrap
    </div>
  </div>
</div>
```

Контейнер может показаться избыточным элементом, но это не так. Он определяет ширину макета и выравнивает его по горизонтали в области просмотра. Кроме того, контейнер нужен, чтобы контролировать отрицательные внешние отступы рядов.

Название "ряд" (row) часто вводит в заблуждение и скрывает настоящее назначение рядов в сетке Bootstrap. Оно вызывает ошибочное представление о горизонтальной линии, однако колонки внутри ряда не всегда располагаются в одну строку. Их положение может меняться. Например, на маленьких экранах они могут выстраиваться друг под другом, а на больших – рядом. Лучше всего думать о ряде, как о родительском элементе для группы колонок.

Колонки нужны для деления области просмотра по горизонтали, при этом в одном ряду могут быть столбцы разной ширины. Их размер

может изменяться в зависимости от некоторых факторов. Пространство между колонками называется "желоб" (gutter). Родительский элемент делится на 12 колонок (Рис. 4.1).

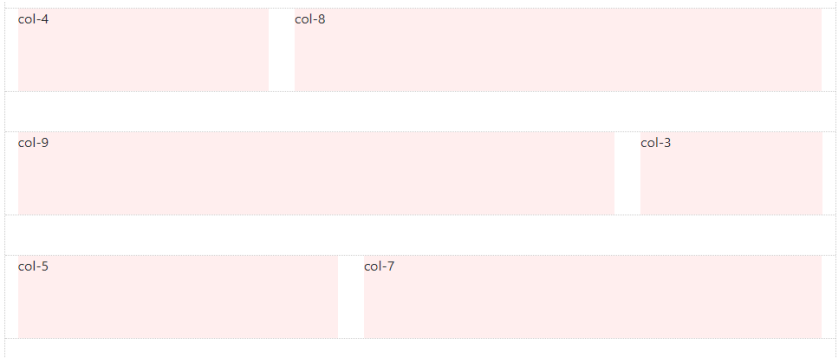







Рис. 4.1. Использование колонок в Bootstrap

Ширина одной и той же колонки, а также общая структура сетки может меняться для разных областей просмотра. Изменениями можно управлять с помощью специальных классов.

Фреймворк определяет 5 уровней адаптивности (брейкпоинтов), которые основаны на ширине области просмотра (Рис. 4.2).

					
	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Максимальная ширина контейнера	None (auto)	540px	720px	960px	1140px
Префикс класса	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
Число колонок	12				
Ширина отступа	30px (15px с каждой стороны столбца)				
Может быть вложенным	Да				
Упорядочивание колонок	Да				

#### Рис. 4.2. Классы адаптивных колонок

Большие брейкпоинты переопределяют меньшие: `xs(default) > sm > md > lg > xl`. Таким образом, класс `.col-sm-6` на самом деле означает, что ширина колонки будет составлять 50% на всех экранах размера `sm` и больше. Но класс `.col-lg-4` может переопределить это правило для больших и супербольших областей видимости, и на таких мониторах колонка будет занимать 33,3% ширины.

Если требуется установить одинаковую ширину столбцов на всех уровнях, достаточно явно указать ее для наименьшего.

Если не указывать размер колонок, то ряд равномерно поделится на все перечисленные столбцы. Для задания размера колонки по содержимому можно использовать классы `.col-{size}-auto`.

## ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать сайт тематики, согласно полученному варианту.

Сайт должен требовать авторизации пользователей. Информация о пользователях хранится в отдельном файле (возможно использование формата xml или json). Должна быть предусмотрена регистрация новых пользователей (с сохранением в файл). Вся другая информация должна быть доступна только после авторизации.

На сайте должна быть отображена таблица (элемент bootstrap-table), содержащая информацию, хранимую на веб-сервере в отдельном текстовом файле (возможно использование формата xml или json). На сайте должна быть предусмотрена возможность добавления новых данных (с сохранением в файл).

Загрузка и передача данных должна осуществляться с помощью AJAX.

Вся валидация данных должна проводиться на стороне клиента без использования дополнительных библиотек.

## ВАРИАНТЫ ЗАДАНИЙ

1. Сайт школы
2. Сайт фитнес-центра
3. Новостной сайт
4. Метеорологический сайт
5. Сайт музыкального исполнителя
6. Сайт кинотеатра
7. Сайт отеля
8. Сайт аэропорта
9. Сайт телеканала
10. Сайт радиостанции
11. Сайт автосалона
12. Сайт ресторана
13. Сайт университета
14. Сайт библиотеки

15. Сайт театра
16. Сайт ветеринарной клиники
17. Сайт туристической фирмы
18. Сайт агентства по продаже недвижимости
19. Сайт букмекерской компании
20. Сайт биржи криптовалют

## **КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ**

1. Опишите механизм работы cookies.
2. Опишите механизм работы сессии.
3. Приведите способ удаления cookie.
4. Дайте определение AJAX.
5. Раскройте, в чем преимущество использования AJAX.
6. Приведите пример использования нативного интерфейса fetch.
7. Раскройте, в чем преимущества и недостатки использования Bootstrap.
8. Опишите механизм создания адаптивной верстки с помощью Bootstrap.

## **ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ**

На выполнение лабораторной работы отводится 5 часов: 4 часа на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы, исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.

## **ЛАБОРАТОРНАЯ РАБОТА №5**

### **MVC. FLASK. JINJA2**

**Цель работы:** получить навык разработки веб-приложений с помощью микрофреймворка Flask

**Задачи:**

1. Разработать серверную часть веб-приложения с помощью микрофреймворка Flask.
2. Разработать шаблоны страниц с помощью шаблонизатора Jinja2.
3. Реализовать динамическую загрузку данных при помощи AJAX

**Результатами работы являются:**

1. Разработанный сайт, содержащий python, JS и CSS файлы.
2. Подготовленный отчет.

## MVC

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо:

1. Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаёте To-Do приложение, код компонента `model` будет определять список задач и отдельные задачи.

2. Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента `view` определяет внешний вид приложения и способы его использования.

3. Контроллер — этот компонент отвечает за связь между `model` и `view`. Код компонента `controller` определяет, как сайт реагирует на действия пользователя.

Существует множество MVC-фреймворков (и микрофреймворков), одним из которых является Python-микрофреймворк Flask.

## FLASK

В самом простом виде (без модели) пример Flask-приложения имеет следующий вид:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'index'

app.run(debug = True, host='127.0.0.1', port='5050')
```



После импорта необходимых модулей необходимо создать экземпляр Flask-приложения. После этого в приложении можно регистрировать контроллеры, т.е. те функции, которые необходимо будет выполнить приложению по соответствующим URL запросов клиента. Контроллеры могут возвращать как конкретные значения, так и HTML-разметку страниц. После ригистрации всех контроллеров необходимо запустить приложение, указав параметрами диапазон IP-адресов, с которых приложение будет прослушивать запросы и номер порта приложения (по умолчанию 5000). Хотя, разумеется, контроллеры нужно выносить в отдельные файлы и, как будет рассмотрено далее, разбивать приложение на `blueprint`'ы.

В качестве URL контроллерам можно указывать изменяемые значения:

```
@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % escape(username)

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    return 'Subpath %s' % escape(subpath)
```

Значения будут подставлены «на лету» по запросу клиента, к полученным от клиента данным можно обращаться как обычным переменным внутри контроллера. Для более гибкой обработки запросов клиентов можно перегрузить контроллеры, указав ожидаемый тип входного значения окончания URL. Применяемые типы: `string`, `int`, `float`, `path`, `uuid`.

Контроллер может вернуть до 3 значений: данные, код HTTP ответа, HTTP-сообщение с заголовками (2 последних можно опустить и Flask вернет значения по умолчанию). Для возвращения данных в формате `json` (например, для AJAX-запросов) нужно воспользоваться функцией

`jsonify` – эта функция создаст `json` из данных переданных параметром (данные должны быть сериализуемы!), кроме того, при формировании HTTP-ответа, добавит заголовок `mimetype='application/json'`. Для управления заголовками также можно вручную создать экземпляр HTTP-ответа с помощью метода `make_response`, установить необходимые заголовки, задать возвращаемое значение, код ответа и вернуть этот объект контроллером.

Если необходимо вернуть HTTP-ошибку, то в любом месте контроллера можно вызвать метод `abort` с параметром – кодом ошибки.

Контроллер может обрабатывать несколько типов HTTP-запросов (по умолчанию GET-запрос), для этого при создании контроллера необходимо указать список доступных HTTP-методов, в теле контроллера можно получить тип текущего запроса через объект `request`:

```
from flask import request

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```

Помимо HTTP-метода, используя объект `request`, можно получить и другую полезную информацию:

- `method` – HTTP-метод
- `form` – словарь значений, переданных в теле POST-запроса
- `args` – словарь параметров GET-запроса
- `files` – список вложенных файлов
- `url` – полный URL запроса
- `cookies` – список cookies, переданных вместе с запросом

По умолчанию Flask хранит значения сессий на клиенте в зашифрованном виде (для альтернативных решений необходимо использовать модуль `Flask-Session`). Для шифрования данных

приложению до запуска необходимо указать секретный ключ шифрования: `app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'`

Для работы с сессиями во Flask нужно использовать словать session:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] =
            request.form['username']
        return redirect(url_for('index'))
    return '''<form method="post" action="login">
        <input type="text" name="username">
        <input type="submit" value="Submit">
    </form>'''

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))
```

В данном примере на GET-запрос по URL 'login' будет возвращен HTML-содержащий форму с полем для ввода имени и кнопкой отправки формы. После отправки формы по URL 'login' придет POST-запрос, обработчик которого установит значению username в сессии полученное значение и с помощью функции `redirect` перенаправит запрос контроллеру, обрабатывающему URL 'index'. Для удаления сессии достаточно удалить значения из словаря session с помощью метода `pop`. Для работы с пользователями можно использовать дополнительный модуль Flask-Login.

В данном примере использовалась функция `url_for`, позволяющая построить полный URL для текущего сервера, что часто используется для обработки статических файлов. Для этого первым параметром указывается ключевое значение 'static', а затем относительный путь к файлу в папке 'static' (по умолчанию в корне проекта, но при запуске приложения можно указать любую доступную директорию парметром static).

## Jinja2 во Flask

Jinja2 – это шаблонизатор, позволяющий на основе подготовленных шаблонов формировать текстовые данные. Данный шаблонизатор широко применяется в микрофреймворке Flask для генерации HTML-страниц контроллерами на основе заранее подготовленных шаблонов, для этого нужно воспользоваться функцией `render_template`. В качестве параметров указав имя шаблона и значения для переменных, применяемых в шаблоне. Все шаблоны будут искаться в директории `templates` (по умолчанию в корне проекта, но при запуске приложения можно изменить на произвольную доступную директорию параметром `templates`).

Возможные конструкции в шаблоне:

- `{{ }}` – переменные, выражения и вызовы функций
- `{# комментарий #}` – комментарии
- `{% set fruit = 'apple' %}` – объявление переменных
- `{% if user.newbie %}`  
    `<p>Display newbie stages</p>`  
    `{% elif user.pro %}`  
    `<p>Display pro stages</p>`  
    `{% elif user.ninja %}`  
    `<p>Display ninja stages</p>`  
    `{% else %}`  
    `<p>You have completed all stages</p>`  
    `{% endif %}` – условный оператор
- `{% for user in user_list %}`  
    `<li>{{ user }}</li>`  
    `{% else %}`  
    `<li>user_list is empty</li>`  
    `{% endfor %}` – цикл. Цикл `for` предоставляет специальную переменную `loop` для отслеживания прогресса цикла

Пример:

```
@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

```
//hello.html
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello, World!</h1>
{% endif %}
```

В примере контроллер будет возвращать HTML, сформированный на основе шаблона 'hello.html'. В случае, если name не передан параметром (None по умолчанию) по шаблону сформируется страница, содержащая заголовок 'Hello, World!', иначе контроллер передаст шаблонизатору значение name и сформируется страница, содержащая заголовок с приветствие конкретного пользователя.

В шаблонах можно применять фильтры, изменяющие переменные до процесса рендеринга. Синтаксис использования фильтров следующий: `variable_or_value|filter_name`.

- `upper` – делает все символы заглавными
- `lower` – приводит все символы к нижнему регистру
- `capitalize` – делает заглавной первую букву и приводит остальные к нижнему регистру
- `escape` – экранирует значение
- `safe` – предотвращает экранирование
- `length` – возвращает количество элементов в последовательности
- `trim` – удаляет пустые символы в начале и в конце
- `random` – возвращает случайный элемент последовательности

Для повышения повторного использования кода, можно использовать вложенные шаблоны:

```
<nav>
    <a href="/home">Home</a>
    <a href="/blog">Blog</a>
    <a href="/contact">Contact</a>
</nav>
```

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    {% include 'nav.html' %}
</body>
</html>

```

Или наследовать шаблоны, для этого в родительском шаблоне определяются блоки, которые могут быть переопределены в шаблонах-наследниках:

```

<body>
    {% block content %}
    {% endblock %}
</body>

{% extends 'base.html' %}
{% block content %}
    {% for bookmark in bookmarks %}
        <p>{{ bookmark.title }}</p>
    {% endfor %}
{% endblock %}

```

Если необходимо дописать что-либо в блок, определенный родителем, можно переопределить блок и вызвать содержимое родителя с помощью метода `{{ super() }}`.

## Blueprints

Blueprints — способ организации Flask-приложений. Эскиз может иметь собственные функции представления, шаблоны и статические файлы, для них можно выбрать собственные URI.

Основная концепция blueprint'ов заключается в том, что они записывают операции для выполнения при регистрации в приложении. Flask связывает функции представлений с blueprint'ами при обработке запросов и генерировании URL'ов от одной конечной точки к другой.

Пример использования blueprint'ов:

```
//создание blueprint'а со своими шаблонами
//и контроллерами
from flask import Blueprint, render_template, abort
from jinja2 import TemplateNotFound

auth_bp = Blueprint('auth_bp', __name__,
                    template_folder='templates')

@auth_bp.route('/', defaults={'page': 'index'})
@auth_bp.route('/<page>')
def show(page):
    try:
        return render_template('pages/%s.html' % page)
    except TemplateNotFound:
        abort(404)

//регистрация blueprint'а в приложении
from flask import Flask, render_template
from flask import Blueprint, render_template, abort
from auth.auth import auth_bp

app = Flask(__name__)
app.register_blueprint(auth_bp)
app.run(debug = True, host='127.0.0.1', port='5050')
```

## Фабрика приложений

В ситуациях, требующих создания различных экземпляров приложения с различными конфигурациями (например, при тестировании), полезным будет создание фабрики приложений – функции, создающей и возвращающей экземпляр приложения нужной конфигурации. Пример:

```

def create_app(config):
    app = Flask(__name__)
    app.config.from_object(config)
    mail.init_app(app)

    from .main import main as main_blueprint
    app.register_blueprint(main_blueprint)

    from .admin import main as admin_blueprint
    app.register_blueprint(admin_blueprint)

    return app

```

Таким образом, несколько файлов (например, run.py или test.py) могут создавать свой конфигурационный объект каждый и, вызывая фабрику приложений, получать свой уникальный экземпляр приложения. Обобщая все вышеуказанное, получим следующую структуру приложения:

```

/
  app/
    blueprint1/
      templates/
      static/
      __init__.py
      blueprint1.py
    static/
    templates/
    model.py
    __init__.py
  run.py
  test.py

```



## ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать сайт тематики, согласно полученному варианту, используя паттерн MVC и микрофреймворк Flask.

Python-приложение должно быть запущенно в индивидуальном виртуальном окружении.

При разработке шаблонов необходимо использовать наследование: на страницах должны быть header, footer и панель навигации.

Как минимум на одной из страниц должна быть отображена таблица, содержащая информацию, хранимую на сервере в отдельном текстовом файле (возможно использование формата xml или json). На сайте должна быть предусмотрена возможность добавления новых данных (с сохранением в файл).

Загрузка и передача данных должна осуществляться с помощью AJAX.

Возможно применение CSS-фреймворка Bootstrap

## ВАРИАНТЫ ЗАДАНИЙ

1. Сайт школы
2. Сайт фитнес-центра
3. Новостной сайт
4. Метеорологический сайт
5. Сайт музыкального исполнителя
6. Сайт кинотеатра
7. Сайт отеля
8. Сайт аэропорта
9. Сайт телеканала
10. Сайт радиостанции
11. Сайт автосалона
12. Сайт ресторана
13. Сайт университета
14. Сайт библиотеки
15. Сайт театра

16. Сайт ветеринарной клиники
17. Сайт туристической фирмы
18. Сайт агентства по продаже недвижимости
19. Сайт букмекерской компании
20. Сайт биржи криптовалют

## **КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ**

1. Раскройте суть паттерна MVC.
2. Раскройте назначение метода `make_response`.
3. Перечислите информацию, содержащуюся в объекте `request`.
4. Опишите механизм использования шаблонов Jinja2
5. Приведите способы генерирования HTTP-ответа с кодом ошибки.
6. Раскройте суть использования фильтров в Jinja2.
7. Опишите механизм наследования шаблонов.
8. Опишите механизм применения вложенных шаблонов.
9. Раскройте суть применения `blueprint`'ов.
10. Раскройте суть фабрики приложений.

## **ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ**

На выполнение лабораторной работы отводится 5 часов: 4 часа на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы, исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.

## **ЛАБОРАТОРНАЯ РАБОТА №6**

### **REACTJS**

**Цель работы:** получить навык разработки веб-приложений с помощью библиотеки ReactJS.

**Задачи:**

1. Изучить принципы работы с виртуальным DOM
2. Изучить основы JSX
3. Разработать 2 SPA (Single Page Application) с помощью библиотеки ReactJS: используя классовые и функциональные компоненты.

**Результатами работы являются:**

1. Разработанные сайты.
2. Подготовленный отчет.

## REACTJS

ReactJS — JS-библиотека для создания пользовательских интерфейсов. Ключевые возможности библиотеки — работа с виртуальным DOM и использование декларативного HTML-подобного JSX-синтаксиса создания элементов интерфейса, преобразующегося в JS-скрипты, рисующие элементы в DOM (определена большая часть тегов HTML, также возможности JSX можно расширять собственными компонентами).

В самом простом варианте использования, достаточно добавить библиотеку в зависимости проекта, а после в JS создать React-элементы и дорисовать их в виртуальный DOM к конкретному узлу с помощью метода `ReactDOM.render(element, parentElement)`:

```
const myelement = <h1>I Love JSX!</h1>;
ReactDOM.render(myelement,
  document.getElementById('root'));
```

Однако, при работе над сложным проектом, состоящим из множества компонентов и большого числа внешних зависимостей проще использовать утилиту `create-react-app` (можно установить, используя менеджер пакетов `npm`, и запустить с помощью `prx`), создающую проект по умолчанию с удобным распределением файлов по директориям: в проекте содержится файл `index.html`, содержащий контент базовой страницы, содержащий корневой элемент; различные файлы ресурсов; JS-файлы с описанием React-элементов; файл с перечнем необходимых зависимостей; а также файл описания действий жизненного цикла приложения и некоторых других дополнительных файлов. При разработке удобно использовать встроенный в `node.js` веб-сервер `express`, автоматически обновляющий файлы проекта при изменении.

Есть 2 способа создания react-компонентов: на основе классов и функций. Рассмотрим их подробнее.

## СОЗДАНИЕ КОМПОНЕНТОВ НА ОСНОВЕ КЛАССОВ

React-компоненты на основе классов представляют собой обычные JS-классы, наследуемые от класса `React.Component` и обязательно имеющие метод `render()`, возвращающий JSX-представление компонента (обязательно должен быть только один JSX-элемент верхнего уровня. При необходимости можно использовать родительские `<div>` или `<React.Fragment>` если не желательно создавать новые элементы в DOM. Для подстановки результатов вычислений необходимо использовать `{}`). В данном примере создается компонент `Car`, затем используемый в другом компоненте `Garage`:

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    return (
      <div>
        <h1>Who lives in my Garage?</h1>
        <Car />
      </div>
    );
  }
}
```

Часто для корректного поведения компонента бывает необходимо передать ему какие-то входные параметры (например, имя пользователя, или его роль). Для этого необходимо использовать специальный объект `props`: `prop`'ы передаются в JSX аналогично атрибутам в HTML – указывается имя `prop`'а и его значение – в самом компоненте к данным `props` можно обратиться с помощью словаря `this.props` (обратите внимание! В методах `React.Component`, таких как `render`, контекст устанавливается в конструкторе `React.Component`. Для

этого в собственном компоненте необходимо создать конструктор, вызывающий конструктор базового класса, для каких-либо других методов контекст необходимо задавать). Изменение props приведет к созданию нового элемента (см. методы жизненного цикла ниже). Важно: при создании списков для идентификации элементов каждому элементу списка необходимо передать prop key, представляющий собой уникальный идентификатор.

```
class Car extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h2>I am a {this.props.brand}!</h1>;
  }
}

//использование компонента
const myelement = <Car brand="Ford" />;
```

Помимо входных параметров, компонент может обладать своим внутренним состоянием, не зависящим (по крайней мере, напрямую) от других компонентов (например, компонент авторизации может хранить имя пользователя, его возраст и т.д.). Для хранения этих данных используется объект state. Объект state инициализируется в конструкторе. Обратиться к объекту state в любом месте компонента можно используя синтаксис: this.state.propertyname. Чтобы изменить значение в объекте состояния, используйте метод this.setState() — никогда не мутите this.state напрямую, так как более поздний вызов setState() может перезаписать эту мутацию! Когда объект state изменяется, компонент ререндерится.

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    }
  }
}
```

```

render() {
  return <React.Fragment>
    <h2>Counter =
      {this.state.counter}</h1>;
    <button onClick={() =>
      this.setState({count:
        this.state.count++})}>
      Increment
    </button>
  </React.Fragment>
}
}

```

Состояние используется также для создания управляемых компонентов – компонентов, хранящих свои данные не в себе, а в родительском react-компоненте (например, значение input’а можно хранить не в самом input’е, а в компоненте-форме, что позволит централизованно обрабатывать данные формы, например, проводить валидацию формы).

Каждый элемент в приложении «проживает» определенные стадии жизни, в reactJS возможно использовать специальные методы, так называемые методы жизненного цикла, для того чтобы в нужные моменты жизни элемента выполнять определенные действия (например, выполнить AJAX-запрос для получения данных, которые потом будут отображены в элементе). В начале элемента не существует в приложении, затем он создается (вызывается его конструктор), вызывается метод `render`, элемент монтируется в DOM (после чего вызывается метод `componentDidMount`), в какой-то момент элемент может изменить свое состояние (вызывается метод `componentDidUpdate`), а когда-то он может быть демонтирован из DOM (перед этим вызовется `componentWillUnmount`). Схема представлена на рис. 5.1. Это наиболее используемые методы жизненного цикла, но они представляют не полный перечень методов жизненного цикла.

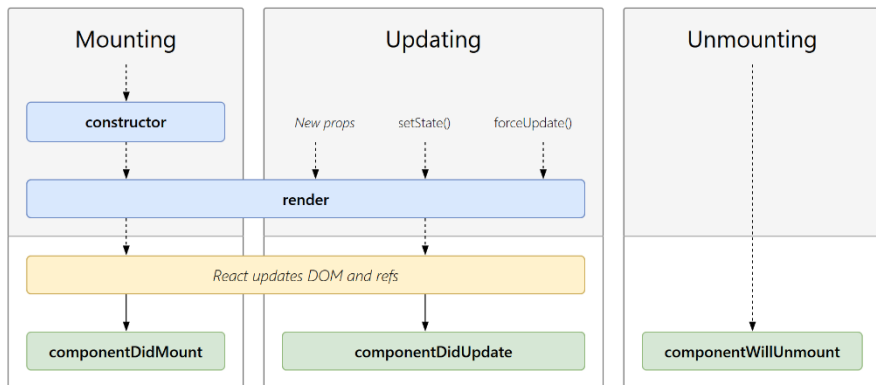


Рис. 5.1

Еще одной интересной возможностью reactJS являются рефы (ref). Рефы дают возможность получить доступ к DOM-узлам или React-элементам, созданным в рендер-методе. По сути реф – это переменная, которая хранит ссылку на react-элемент. Ситуации, в которых использование рефов является оправданным:

- Управление фокусом, выделение текста или воспроизведение медиа.
- Императивный вызов анимаций.
- Интеграция со сторонними DOM-библиотеками.

Пример использования рефов:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

```
//установка фокуса в произвольном месте кода
this.myRef.current.focus();
```



## СОЗДАНИЕ ФУНКЦИОНАЛЬНЫХ КОМПОНЕНТОВ

Изначально функциональные компоненты были облегченной версией создания компонентов — функциональные компоненты обладают более простым синтаксисом, при этом не обладают состоянием и не имеют методов жизненного цикла — однако, в настоящий момент с расширением функционала с помощью различных хук, являются рекомендуемым способом создания react-компонентов, т.к. использование собственных хук позволяет существенно понизить дублирование кода по сравнению с компонентами на основе классов.

Функциональный компонент представляет собой обычную JS-функцию, возвращающую JSX, в качестве входного параметра передаются props:

```
function Car(props) {  
  return <h2>I am a {props.brand}!</h1>;  
}
```

Хуки — это функции, с помощью которых можно использовать состояние и методы жизненного цикла React из функциональных компонентов. Хуки не работают внутри классов — они дают возможность использовать React без классов.

Правила использования:

- Хуки следует вызывать только на верхнем уровне. Не вызывайте хуки внутри циклов, условий или вложенных функций (т.к. может нарушиться порядок вызова хук, что приведет к печальным последствиям).
- Хуки следует вызывать только из функциональных компонентов React. Не вызывайте хуки из обычных JavaScript-функций (опять же, порядок вызова хук станет непредсказуемым). Есть только одно исключение, откуда можно вызывать хуки — это пользовательские хуки.

Рассмотрим базовые хуки.

1. Хука состояния.

Вызов `useState` возвращает две вещи: текущее значение состояния и функцию для его обновления. Единственный аргумент `useState` — это начальное состояние: `const [count, setCount] = useState(0);`

Обращение к состоянию: `<div>{count}</div>`

2. Хука состояния часто используется вместе с хукой эффекта (загрузка и сохранение данных).

Хук эффекта даёт возможность выполнять побочные эффекты в функциональном компоненте. Побочными эффектами в React-компонентах могут быть: загрузка данных, оформление подписки и изменение DOM вручную (хук `useEffect` представляет собой совокупность методов `componentDidMount`, `componentDidUpdate`, и `componentWillUnmount`):

```
useEffect(() => { document.title =  
  `Вы нажали ${count} раз`; },  
  [count]  
);
```

Вторым необязательным параметром передается массив переменных, на изменении которых должен срабатывать эффект. Если эффект возвращает функцию, React выполнит её только тогда, когда наступит время сбросить эффект. Пример использования хук:

```
function Garage(props) => {  
  const [cars, setCars] = useState([])  
  useEffect(() => {  
    let result = await fetch(...)  
    if (result.ok){  
      setCars(result.json())  
    }  
  })  
  
  return <React.Fragment>  
    {  
      cars.map(item => <h2>{item.name}</h2>)  
    }  
  </React.Fragment>  
}
```

3. Для работы с рефами в функциональных компонентах используется хук `useRef`. `useRef` возвращает изменяемый `ref`-объект, свойство `.current` которого инициализируется переданным аргументом (`initialValue`). Возвращённый объект будет сохраняться в течение всего времени жизни компонента. Пример использования:

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onClick = () => {
    // `current` указывает на элемент `input`
    inputEl.current.focus();
  };
  return (
    <React.Fragment>
      <input ref={inputEl} type="text" />
      <button onClick={onClick}>
        Установить фокус на поле ввода
      </button>
    </React.Fragment>
  );
}
```

Но хук `useRef()` полезен не только установкой атрибута с рефом. Он удобен для сохранения любого мутируемого значения, по аналогии с тем, как используются поля экземпляра в классах.

Это возможно, поскольку `useRef()` создаёт обычный JavaScript-объект. Единственная разница между `useRef()` и просто созданием самого объекта `{current: ...}` — это то, что хук `useRef` даст один и тот же объект с рефом при каждом рендере. Мутирование свойства `.current` не вызывает повторный рендер.

Перечисленные хуки дают базовые представления о работе с функциональными хуками, другие полезные хуки рассмотрены в документации <https://ru.reactjs.org/docs/hooks-reference.html>.

Кроме базовых хук у прикладного программиста есть возможность создавать свои собственные кастомные хуки, комбинируя существующие. Созданные хуки можно многократно использовать в различных компонентах, что позволяет сократить дублирование кода

(в компонентах на основе классов пришлось бы писать одну и ту же логику для каждого из компонента). Пример пользовательской хуки:

```
function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(friendID,
      handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus
        (friendID, handleStatusChange);
    };
  });
  return isOnline;
}
```

Данная хука при монтировании компонента подписывается на некое API чата, через которое будет получен статус друга (онлайн или оффлайн) с указанным идентификатором, по демонтировании компонента произойдет отписка от API чата и возвращает переменную, содержащую значение статуса друга. Т.е. данная хука комбинирует хуку состояния и хуку эффекта. В дальнейшем может многократное использование данной хуки в разных компонентах, например, в компоненте, отображающем список друзей (классические зеленые и красные «точки» на иконках друзей) и в компоненте диалога с другом (надпись Online или Offline). Используя компоненты на основе классов, пришлось реализовывать одну и ту же логику в обоих компонентах.

Благодаря тому, что есть возможность создавать пользовательские хуки, помимо стандартных хук react'а, существует большое количество хук, реализованных в различных библиотеках, таких, как, например, Formik.

## **ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

Разработать 2 сайта тематики, согласно полученному варианту (используя react-компоненты на основе классов и функций).

Сайт должен состоять минимум из 2-х частей с возможностью перехода между ними (заданием переменной в state элемента).

Обязательно использование вложенных React-элементов (с передачей props, например, введенных в родительском элементе данных), условного рендеринга.

Использование готовых компонентов и сторонних библиотек запрещено.

### **ВАРИАНТЫ ЗАДАНИЙ**

1. Сайт школы
2. Сайт фитнес-центра
3. Новостной сайт
4. Метеорологический сайт
5. Сайт музыкального исполнителя
6. Сайт кинотеатра
7. Сайт отеля
8. Сайт аэропорта
9. Сайт телеканала
10. Сайт радиостанции
11. Сайт автосалона
12. Сайт ресторана
13. Сайт университета
14. Сайт библиотеки
15. Сайт театра
16. Сайт ветеринарной клиники
17. Сайт туристической фирмы
18. Сайт агентства по продаже недвижимости
19. Сайт букмекерской компании
20. Сайт биржи криптовалют

## **КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ**

1. Охарактеризуйте JSX.
2. Приведите пример кода рендера react-компонента к какому-либо узлу DOM.
3. Охарактеризуйте props.
4. Охарактеризуйте state.
5. Приведите методы жизненного цикла react-компонента.
6. Опишите маханизм использования рефов.
7. Сравните react-компоненты на основе классов и на основе функций.
8. Перечислите ограничения, накладываемые на использование хук.
9. Опишите useState.
10. Опишите useEffect.
11. Опишите useRef.
12. Приведите пример создания собственной хуки.

## **ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ**

На выполнение лабораторной работы отводится 7 часов: 6 часов на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы, исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.

## СПИСОК ЛИТЕРАТУРЫ

1 Беликова, С. А. Основы HTML и CSS: проектирование и дизайн веб-сайтов: учебное пособие по курсу «Web-разработка» / С. А. Беликова, А. Н. Беликов. — Ростов-на-Дону, Таганрог : Издательство Южного федерального университета, 2020 — 174 с. — ISBN 978-5-9275-3435-7. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <http://www.iprbooks.ru/100186.html>

2 Вагин, Д. В. Современные технологии разработки веб-приложений : учебное пособие / Д. В. Вагин, Р. В. Петров. — Новосибирск : Новосибирский государственный технический университет, 2019 — 52 с. — ISBN 978-5-7782-3939-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <http://www.iprbookshop.ru/98738.html>

3 Кулькова, Л. И. Задачи и упражнения по JavaScript : учебное пособие / Л. И. Кулькова, С. И. Салпагаров. — Москва : Российский университет дружбы народов, 2018 — 102 с. — ISBN 978-5-209-08646-8. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <http://www.iprbookshop.ru/104199.html>

4 Основы работы с HTML : учебное пособие / . — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021 — 208 с. — ISBN 978-5-4497-0903-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <http://www.iprbookshop.ru/102036.html>

5 Сузи, Р. А. Язык программирования Python: учебное пособие / Р. А. Сузи. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020 — 350 с. — ISBN 978-5-4497-0705-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: <https://www.iprbookshop.ru/97589.html>

6 Сычев, А. В. Перспективные технологии и языки веб-разработки : практикум / А. В. Сычев. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2019 —

493 с. — ISBN 978-5-4486-0507-9. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: [http://www.iprbookshop.ru/79730 . html](http://www.iprbookshop.ru/79730.html)