## Теория по веб-программированию

1. **Протокол HTTP. Структура запросов и ответов, методы запросов, коды ответов сервера, заголовки запросов и ответов.**

HTTP (HyperText Transfer Protocol) — протокол прикладного уровня; аналогичными ему являются FTP и SMTP. Обмен сообщениями идёт по схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера).

API многих сервисов может использовать HTTP для передачи данных — сами могут иметь любой формат, например, XML или JSON.

**Протокол HTTP**

HTTP (англ. HyperText Transfer Protocol – «протокол передачи гипертекста») – протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных). Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

* Потребителей (клиентов), которые инициируют соединение и посылают запрос;
* Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

**Структура запросов**

HTTP запрос состоит из трех основных частей, которые идут в нем именно в том порядке, который указан ниже. Между заголовками и телом сообщения находится пустая строка (в качестве разделителя), она представляет собой символ перевода строки.

1. строка запроса (Request Line)
2. заголовки (Message Headers)
3. пустая строка (разделитель)
4. тело сообщения (Entity Body) – необязательный параметр

Строка запроса – указывает метод передачи, URL-адрес, к которому нужно обратиться и версию протокола HTTP.  
Заголовки – описывают тело сообщений, передают различные параметры и др. сведения и информацию.  
Тело сообщения – это сами данные, которые передаются в запросе. Тело сообщения – это необязательный параметр и может отсутствовать.

Пример:

GET /iaps/labs HTTP/1.1

Host: сs.ifmo.ru

User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.6.14

Accept: text/html

Connection: close

**Структура ответов**

HTTP ответ состоит из трех основных частей как и HTTP запрос.

1. строка ответа (Response Line)
2. заголовки (Message Headers)
3. пустая строка (разделитель)
4. тело сообщения (Entity Body) – необязательный параметр

Строка ответа – указывает версию протокола HTTP, код состояния и пояснение к коду состояния.  
Заголовки – описывают тело сообщений, передают различные параметры и др. сведения и информацию.  
Тело сообщения – чаще всего представляет собой содержимое web-страницы, не обязательный параметр

Пример:

HTTP/1.0 200 OK

Date: Wed, 02 Mar 2011 11:11:11 GMT

Server: Apache

X-Powered-By: PHP/5.2.4-2ubuntu5wm1

Last-Modified: Wed, 02 Mar 2011 11:11:11 GMT

Content-Language: ru

Content-Type: text/html; charset=utf-8

Content-Length: 1234

Connection: close

...HTML-код запрашиваемой страницы...

**Методы запросов**

* GET – запрашивает представление ресурса, может только извлекать данные.
* HEAD – как GET, только без тела ответа.
* POST – используется для отправка сущностей определенному ресурсу. Может изменять данные.
* PUT – создает новый ресурс или заменят представление целевого ресурса (в отличее от POST для идентичных наборов данных будет иметь одинаковый результат).
* DELETE – удаляет ресурс.
* CONNECT – устанавливает "туннель" к серверу, определенному по ресурсу.
* OPTIONS – для описания параметров соединения с ресурсом
* TRACE – вызов возвращаемого текстового сообщения
* PATCH – частичное изменение ресурса

**Коды ответов сервера**

1. 1хх: Informational
2. 2xx: Success
3. 3xx: Redirection (перенаправление)
4. 4xx: Client Error
5. 5xx: Server Error

**Заголовки запросов и ответов**

Заголовки представляются в формате ключ: значение Выделяют 4 группы заголовков:

* General Headers – могут включаться в любое сообщение клиента и сервера.  
  Пример – CacheControl.
* Request Headers – используются только в запросах клиента.  
  Пример – Referer.
* Response Headers – используются только в запросах сервера.  
  Пример – Allow.
* Entity Headers – сопровождают любую сущность сообщения.  
  Пример – Content-Language.

**2. Язык разметки HTML. Особенности, основные теги и атрибуты тегов**

**Язык разметки HTML**

HTML – Hypertext Markup Language, язык разметки гипертекста. Является стандартным языком разметки документов в интернете. Браузеры интерпретируют его и отображают в виде документа.

Пример:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Пример веб-страницы</title>

</head>

<body>

<h1>Заголовок</h1>

<!-- Комментарий -->

<p>Первый абзац.</p>

<p>Второй абзац.</p>

</body>

</html>

**Особенности**

Документ должен начинаться со строки объявления версии HTML:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

Документ состоит из элементов, начало и конец которых обозначаются *тегами*

<b>some text here</b>

Некоторые теги могут не содержать текст (<br> – перенос строки, <img> – картинка, <input> – элемент ввода в форме). Их не нужно закрывать:

<!-- плохо -->

<input type="..."></input>

<!-- хорошо -->

<input type="..."/>

В HTML5 введены *семантические* теги <header>, <footer>, <section>, которые аналогичны <div>, но указывают на логическую структуру.

**Основные теги**

Начало и конец документа обозначаются тегами <html> и </html>. Внутри этих тегов должны находиться заголовок <head>...</head> и тело документа <body>...</body>.

**Атрибуты тегов**

Чтобы расширить возможности отдельных тегов и более гибко управлять содержимым контейнеров применяются атрибуты тегов.

Когда для тега не добавлен какой-либо допустимый атрибут, это означает, что браузер в этом случае будет подставлять значение, заданное по умолчанию. Если вы ожидали получить иной результат на web-странице, проверьте, возможно, следует явно указать значения некоторых атрибутов.

Допустимо использовать некоторые атрибуты у тегов, не присваивая им никакого значения, как показано в примере.

Пример:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Добавление формы</title>

</head>

<body>

<form action="self.php">

<p><input type="text"></p>

<p><input type="submit" disabled></p>

</form>

</body>

</html>

В данном примере используются атрибуты action, type, disabled и некоторые другие в теге <meta>. У атрибута disabled явно не задано значение. Подобная запись называется «сокращенный атрибут тега».

Порядок атрибутов в любом теге не имеет значения и на результат отображения элемента не влияет.

Каждый атрибут тега относится к определенному типу (например: текст, число, путь к файлу и др.), который обязательно должен учитываться при написании атрибута. Если тип ожидаемых данных конкретного атрибута не совпадёт с типом переданных данных, то значение будет проигнорировано и возникнет ошибка при валидации документа.

**3. Структура HTML-страницы. Объектная модель документа (DOM)**

**Структура HTML-страницы**

Документ должен начинаться со строки объявления версии HTML:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

Документ состоит из элементов, начало и конец которых обозначаются *тегами*

<b>some text here</b>

Элементы могут быть вложенными:

<b>

Этот текст будет полужирным,

<i>а этот - ещё и курсивным</i>

</b>

Начало и конец документа обозначаются тегами <html> и </html>. Внутри этих тегов должны находиться заголовок <head>...</head> и тело документа <body>...</body>.

**Объектная модель документа (DOM)**

DOM – программный интерфейс для HTML и XML документов, описывающий структурированное представление документа и определяющий, как это структура может быть доступна из программ, которые могут изменять ее содержимое. (Другими словами, DOM соединяет web-страницу с языками описания сценариев).

Согласно DOM документ может быть представлен в виде объектов, с которыми можно производить манипуляции:

* генерация и добавление узлов
* получение узлов
* изменение узлов
* изменение связей между ними
* удаление узлов

Обращение к узлам происходит с помощью элементов document или window.

**Дополнительно (BOM)**

Есть еще BOM – объектная модель браузера. Основное предназначение — управление окнами браузера и обеспечение их взаимодействия. BOM специфична для каждого браузера. Может в создание системных диалогов, управление информацией о параметрах монитора и браузера и всякое такое.

**4. HTML-формы. Задание метода HTTP-запроса. Правила размещения форм на страницах, виды полей ввода**

**HTML-формы**

Форма предназначена для обмена данными между пользователем и сервером.

**Правила размещения форм на страницах**

Документ может содержать любое число форм, но одновременно на сервер может быть отправлена только одна из них.  
Вложенные формы запрещены.  
Задается с помощью тега <form> и могут содержать в себе атрибуты:

* action – содержит URI обработчика формы (обязательный атрибут)
* method – по умолчанию GET
* enctype – тип кодирования
* accept – MIME-типы для загрузки файлов
* name

**Задание метода HTTP-запроса**

Метод HTTP задаётся атрибутом method тега <form>:

<form method="GET" action="URL">

...

</form>

**Виды полей ввода**

Виды полей <input>:

* Кнопки. Типы кнопок:
  + submit – кнопка для отправки данных формы на сервер
  + image – поле с изображением, при нажатии на рисунок данные формы отправляются на сервер
  + reset – кнопка для возвращения данных формы в первоначальное значение
  + button – обычная кнопка
* checkbox
* radio
* select
* text и многострочный textarea
* password
* hidden (скрытое поле)
* file

Пример:

<form method="POST" action="handler.php">

<p>

<b>Как по вашему мнению расшифровывается аббревиатура &quot;ОС&quot;?</b>

</p>

<p>

<input type="radio" name="answer" value="a1">Офицерский состав<br>

<input type="radio" name="answer" value="a2">Операционная система<br>

<input type="radio" name="answer" value="a3">Большой полосатый мух

</p>

<p>

<input type="submit">

</p>

</form>

**5. Каскадные таблицы стилей (CSS). Структура - правила, селекторы. Виды селекторов, особенности их применения. Приоритеты правил. Преимущества CSS перед непосредственным заданием стилей через атрибуты тегов**

**CSS**

CSS (*Cascading Style Sheet*, каскадные таблицы стилей) – формальный язык описания внешнего вида документа с помощью языка разметки. Используется для задания цветов, шрифтов и других аспектов представления документа.

**Структура**

Таблица стилей состоит из набора правил.

CSS-правило – блок, состоящий из селектора и блока объявления стилей.

Селекторы – имя тега, к которому применяется правило.

селектор, селектор {

свойство: значение;

свойство: значение;

свойство: значение;

}

Пример:

div, td {

background-color: red;

}

**Основные виды селекторов**

* \* – любые элементы
* div – элементы с тегом div
* #id – элемент по id
* .class – элементы с классом class
* [name="value"] – селекторы по атрибуту
* :visited – псевдоклассы
* div p – элементы p, являющиеся потомками div
* div > p – только непосредственные потомки
* div ~ p – правые соседи: все p на том же уровне вложенности, которые идут после div
* div + p – первый правый сосед: p на том же уровне вложенности, который идёт сразу после div

**Приоритеты правил**

1. Самый высокий приоритет имеет атрибут style
2. Второе по приоритету – присутствие ID в селекторе
3. Все атрибуты (включая class и псевдоклассы)
4. Самый низкий – селекторы с именами элементов и псевдоэлементами

!important позволяет повысить приоритет стиля

**Преимущества CSS перед атрибутом style**

Основная цель — разделение содержимого документа и его представления. Позволяет представлять один и тот же документ в различных методах вывода (например, обычная версия и версия для печати)

Главное преимущество CSS заключается в том, что стиль может применяться не только для одного элемента, а для всех элементов подходящих под селектор. Таким образом можно оптимизировать код например чтобы все кнопки выглядели одинаково надо будет всего один раз прописать стиль с правильным селектором, а не на каждой кнопке вставлять один и тот-же код.

**6. LESS, Sass, SCSS. Ключевые особенности, сравнительные характеристики. Совместимость с браузерами, трансляция в "обычный" CSS**

**LESS**

LESS - это динамический язык стилей, который является надстройкой над CSS (Поэтому любой CSS код будет валидный LESS).

Преимущества LESS:

* Переменные (и области видимости переменных).
* Операции (в том числе и для управления цветом, т.е можно смешивать цвета: #941f1f + #222222).
* И другие функции для работы с цветом (осветление, затемнение и т.п.)
* Вложенность (можно вложить одно правило в другое, article.post p {} <=> article.post { p{ }}).
* Объединение аргументов.

LESS-файл конвертируется в CSS при помощи js (для этого необходимо скачать less.js с сайта LESS).

<script src="less.js" type="text/javascript"></script>

Затем можно привязывать файлы с расширением .less.

<link rel="stylesheet/less" type="text/css" href="style.less">

**Sass**

Sass - это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей.

Преимущества Sass:

* Вложенные правила.
* Переменные.
* Возможность создавать миксины, позволяющие создавать многоразовые CSS-правила - группы деклараций, для многократного использования. *(LESS в это не может)*
* Расширения. Одиночный селектор может быть расширен больше, чес одним селектором с помощью @extend.
* Есть логика. (if/then/for). *(Этого в LESS тоже нет)*

*Не может компилироваться на сервере в CSS (LESS использует js).*

Браузер не распознает файлы Sass, так что сначала их нужно скомпилировать в обычный CSS.

**SCSS**

SCSS — "диалект" языка SASS. Отличие SCSS от SASS заключается в том, что SCSS больше похож на обычный CSS код.

* @import - @import "template" подключит template.scss.
* Вложенность.
* $переменные.
* Математика чисел и цветов.
* Строки (умеет складывать строки, поддерживает конструкцию #{$var})

**Совместимость с браузерами, трансляция в "обычный" CSS**

Браузеры могут не поддерживать LESS, Sass, SCSS-таблицы стилей — нужен специальный транслятор, который преобразует эти правила в «обычный» CSS.

Less может быть сконвертирован в CSS прямо в браузере. А Sass и SCSS должны быть заранее скомпилированы в обычный CSS.

<!-- Пример для Maven -->

<!-- Sass compiler -->

<plugin>

<groupId>org.jasig.maven</groupId>

<artifactId>sass-maven-plugin</artifactId>

<version>2.25</version>

<executions>

<execution>

<phase>prepare-package</phase>

<goals>

<goal>update-stylesheets</goal>

</goals>

</execution>

</executions>

<configuration>

<resources>

<resource>

<!-- Set source and destination dirs -->

<source>

<directory>${project.basedir}/src/main/webapp/sass</directory>

</source>

<destination>${project.basedir}/src/main/webapp/sass\_compiled</destination>

</resource>

</resources>

</configuration>

</plugin>

**7. Клиентские сценарии. Особенности, сферы применения. Язык JavaScript**

**Клиентские сценарии**

Сценарий - код, включенный в состав web-страницы. Клиентский сценарий выполняется на компьютере-клиенте, для этого необходим встроенный интерпретатор. Вставка сценария в web-страницу происходит при помощи тега <script>. Клиентский сценарии в основном используются для придания интерактивности веб-страницам.

**Язык JavaScript**

JavaScript является объектно-ориентированным языком. JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания. Js имеет автоматическое приведение типов, автоматическая сборка мусора, анонимные функции, функции как объекты первого класса (т.е. могут быть сохранены в переменную, переданны в функцию как аргумент, созданы во время выполнения программы и т.п.)

Основные архитектурные черты:

* динамическая типизация;
* слабая типизация;
* автоматическое управление памятью;
* прототипное программирование;
* функции как объекты первого класса.

**Типы данных js**

* number – целые, дробные числа, Infinity, NaN
* String
* boolean
* null
* undefined – «значение не присвоено»
* object – для коллекций и более сложных сущностей

**8. Версии ECMAScript, новые возможности ES6 и ES7**

ECMAScript — это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков. Стандартизирован международной организацией ECMA в спецификации. (ECMAScript это стандарт, а JavaScript его реализация).

Имеет 5 примитивных типов данных:

* Number
* String
* Boolean
* Null
* Undefined Объектный тип данных — Object и 15 различных видов инструкций.

В особенности можно добавить то, что блок не ограничивает область видимости функции. Если переменная объявляется вне функции, то она попадает в глобальную область видимости. Функция — это тоже объект.

**Версии ECMAScript**

**ES6**

New features:

* новый синтаксис для написания классов и модулей
* итераторы и циклы for/of
* Python-style генераторы
* двоичные данные
* лямбда-выражения
* типизированные массивы
* коллекции
* обещания (promises)
* рефлексию и прокси
* усовершенствовали числа и математику
* ключевое слово let (которое помогает объявить переменной область видимости - блок) и const.

**ES7**

New features:

* операция возведения в степень (\*\*)
* Array.prototype.includes().

**TODO**

**9. Синхронная и асинхронная обработка HTTP-запросов. AJAX**

Синхронный запрос - запрос с ожиданием ответа. (скрипт послал запрос (объект) на сервер и ждет ответ).

Асинхронный запрос - запрос без ожидания ответа от сервера. (скрипт послал запрос (объект) на сервер, но продолжает выполняться, когда данные вернутся вступает в дело событие onreadystatechenge. Сам объект меняет это событие, когда у него меняется свойство readyState. Для события создается собственная функция, в которой проверяется свойство readyState. И как только оно становится равным "4" - это значит, что данные с сервера пришли. Теперь можно полученные данные обрабатывать).

var request = getXmlHttpRequest(); // создание объекта

request.onreadystatechenge = function(){ // установка обработчика onreadystatechenge и проверка свойства readyState

if(request.readyState == 4)

...

}

request.open('GET', url, true); // готовим запрос

request.send(null); // посылаем запрос

**AJAX**

Asynchronous JavaScript and XML - подход к построению интерактивных пользовательских интерфейсов web-приложений, заключающийся в «фоновом» обмене данными браузера с web-сервером.

Пользователь что-то делает -> скрипт определяет с чем там надо работать -> браузер отправляет запрос на сервер -> сервер возвращает только то, от чего ожидаются изменения -> скрипт вносит изменения обратно (без перезагрузки страницы).

**10. Библиотека jQuery. Назначение, основные API. Использование для реализации AJAX и работы с DOM**

jQuery - библиотека js, помогающая легко получить доступ к любому элементу DOM и манипулировать ими, предоставляет API для работы с AJAX.

jQuery включается в страницу как внешний файл.

<script src="jquery-2.2.2.min.js">

Вся работа с jQuery ведется с помощью функции $. Работу с jQuery можно разделить на 2 типа:

* Получение jQuery-объекта с помощью функции $().
* Вызов глобальных методов у объекта $.

Типичный пример манипуляции сразу несколькими узлами DOM заключается в вызове $ функции со строкой селектора CSS, что возвращает объект jQuery, содержащий некоторое количество элементов HTML-страницы. Эти элементы затем обрабатываются методами jQuery.

$("div.test").add("p.quote").addClass("blue").slideDown("slow");

//находит элементы div с классом test, все элементы p с классом quote, добавляет им класс blue, ...

$.ajax и соответствующие функции позволяют использовать методы AJAX

$.ajax({

type: "POST",

url: "some.php", // обращение к some.php

data: {name: 'John', location: 'Boston'}, //с какими-то параметрами

success: function(msg){

alert( "Data Saved: " + msg ); // полученный результат выводится в alert

}

});

**11. Реализация AJAX с помощью SuperAgent**

SuperAgent - API для реализации AJAX:

request

.post('/api/pet')

.send({ name: 'Manny', species: 'cat' })

.set('X-API-Key', 'foobar')

.set('Accept', 'application/json')

.end(function(err, res){

if (err || !res.ok) {

alert('Oh no! Error');

} else {

alert('yay got ' + JSON.stringify(res.body));

}

});

**12. Серверные сценарии. CGI - определение, назначение, ключевые особенности**

CGI (Common Gateway Interface — общий интерфейс шлюза) — стандарт интерфейса, используемого для связи внешней программы с web-сервером. Программу, которая работает по такому интерфейсу совместно с web-сервером, принято называть шлюзом (оно же скрипт или CGI-программа). По сути позволяет использовать консоль ввода и вывода для взаимодействия с клиентом.

CGI - сценарии

* CGI — механизм вызова пользователем программ на стороне сервера.
* Данные отправляются программе посредством HTTP-запроса, формируемого web-браузером.
* То, какая именно программа будет вызвана, обычно определяется URL запроса.
* Каждый запрос обрабатывается отдельным процессом CGI-программы.
* Взаимодействие программы с web-сервером осуществляется через stdin и stdout.

**13. FastCGI и CGI**

CGI — устаревшая технология, позволяющая взаимодействовать web-серверу с сервером приложений. Для каждого запроса запускается процесс с интерпретатором PHP, после возвращения ответа он завершается. Поскольку это очень неэффективно, был создан FastCGI, в котором процесс интерпретатора не завершается, а используется для последующих запросов.

В PHP возможна многопоточность через расширение pthreads, но оно недоступно при использовании PHP как сервера приложений, только при написании скриптов для командной строки.

**14. Язык PHP - синтаксис, типы данных, встраивание в web-страницы, правила обработки HTTP-запросов. Особенности реализации принципов ООП в PHP**

PHP - скриптовый язык общего назначения, интенсивно применяемый для разработки web-приложений.

**Типы данных php**

PHP - язык программирования с динамической типизацией. Преобразования между скалярными типами происходят неявно.

Скалярные типы: integer, float, double, boolean, string.

Не скалярные: array, object, resource, null

Псевдотипы: mixed (любой тип), number, callback (string или анонимная функция), void.

**Запрос**

$url = 'https://translate.yandex.ru';

$context = stream\_context\_create([

'http' => [

'method' => 'POST',

'content' => http\_build\_query([

'lang' => 'ru-en',

'text' => 'Все получилось',

])

]

]);

stream\_context\_create() отвечает за создание контекста запросов различных протоколов, в том числе ftp, ssl, tcp

**ООП в PHP**

class ClassName {

public $publicName;

private $privateName;

protected $protectedName;

const CONST\_VAL = 'val';

public function getPrivateName() {

return $this->$privateName; // $this – ссылка на сам объект, $parent - на родительский.

}

};

echo ClassName::CONST\_VAL; // для обращения к константам

$classname = new ClassName();

$className->publicName; // доступ к переменной

PHP поддерживает все три основных механизма ООП — инкапсуляцию, полиморфизм подтипов и наследование (с помощью extend). Поддерживаются интерфейсы (с помощью implements). Есть абстрактные и final методы и классы. Множественное наследование не поддерживается, но класс может реализовывать несколько интерфейсов или с помощью механизма особенностей (trait), который имеет средства для разрешения конфликтов.

Методы: \_\_construct() – конструктор \_\_destruct() – для деинициализации объекта \_\_get(), \_\_set() \_\_sleep(), \_\_wakeup() \_\_clone()