1. Что такое сервер с позиции программного обеспечения?
Сервер (программное обеспечение) - программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

2. Что такое сервер с позиции аппаратного обеспечения?
Сервер (аппаратное обеспечение) - выделенный или специализированный компьютер для выполнения сервисного программного обеспечения без непосредственного участия человека.

3. Что такое клиент (в рамках клиент-серверной архитектуры)?
Клиент - это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

4. Что такое база данных?
База данных - это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно категорировать. Базы данных функционируют под управлением систем управления базами данных (сокращенно СУБД).

5. Что такое API в общем случае и с позиции клиент-сервера?
API (Application Programming Interface - прикладной программный интерфейс) - набор функций и подпрограмм, обеспечивающий взаимодействие клиентов и серверов.
API (в клиент-сервере) - описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

6. Что такое сервис и чем он отличается от сервера?
Сервис - легко заменяемый компонент сервисноориентированной архитектуры со стандартизированными интерфейсами.

7. Что такое веб-служба?
Веб-сервис(веб-служба) - идентифицируемая уникальным вебадресом (URL-адресом) программная система со стандартизированными интерфейсами.

8. Что такое клиент-сервер как вычислительная модель?
Вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер - это программное обеспечение.

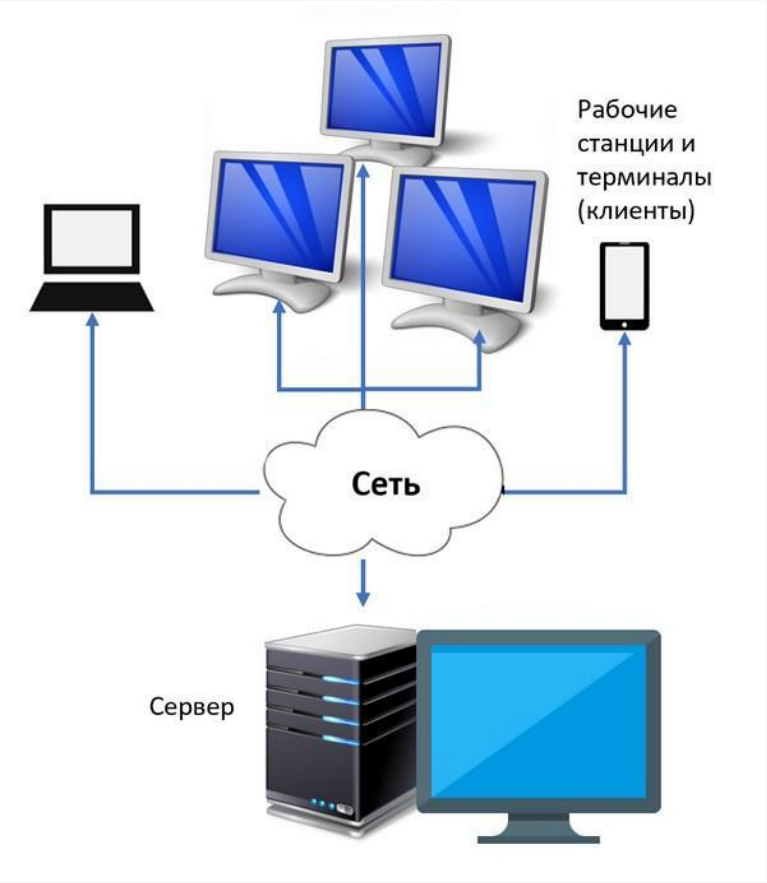
9. Из чего состоит модель клиент-сервера? Опишите ее компоненты.

Клиент Сервер БД

10.Что такое архитектура клиент-сервер как модель?

Архитектура «Клиент-Сервер» (также используются термины «сеть Клиент-Сервер» или «модель Клиент-Сервер») предусматривает разделение процессов предоставление услуг и отправки запросов на них на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других.

В архитектуре «Клиент-Сервер» несколько компьютеров-клиентов (удалённые системы) посылают запросы и получают услуги от централизованной служебной машины – сервера (server – англ. «официант, обслуга»), которая также может называться хост-системой (host system, от host – англ. «хозяин», обычно гостиницы).
Клиентская машина предоставляет пользователю т.н. «дружественный интерфейс» (user-friendly interface), чтобы облегчить его взаимодействие с сервером.



11.Перечислите достоинства клиент-серверной архитектуры.

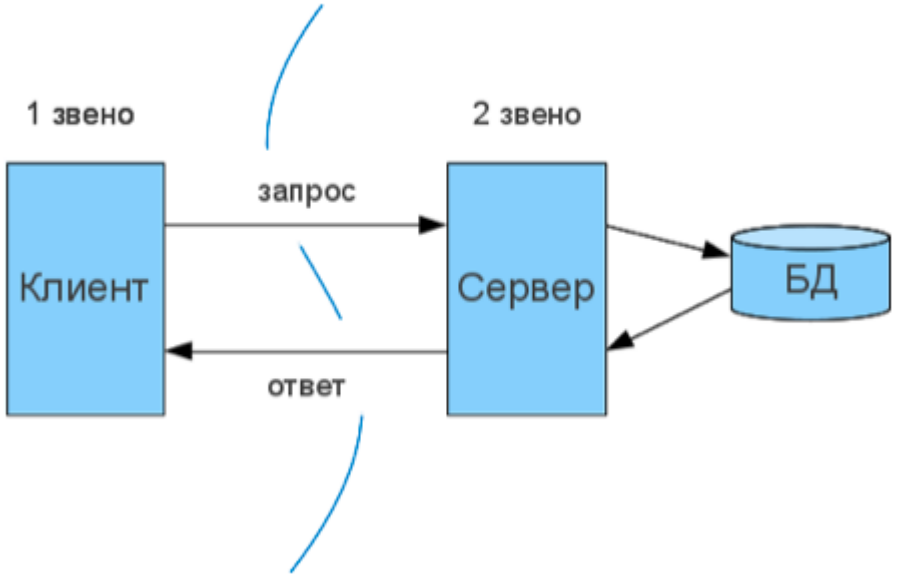
основная нагрузка ложится на сервер

разграничение доступа к данным разграничение полномочий между клиентом и сервером
кроссплатформенность реализаций клиента

12.Перечислите недостатки клиент-серверной архитектуры.
неработоспособность сервера может сделать неработоспособной всю информационную систему
поддержка работы данной системы требует отдельного специалиста
высокая стоимость серверного оборудования

13.Что такое веб-приложение и из каких основных компонентов состоит?
Веб-приложение — клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика **веб-приложения** распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети.

14.Опишите существующие клиент-серверные конфигурации архитектуры.
Двухзвенная архитектура - распределение трех базовых компонентов между двумя узлами (клиентом и сервером). Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме.



Расположение компонентов на стороне клиента или сервера определяет следующие основные модели их взаимодействия в рамках двухзвенной архитектуры:

- **Сервер терминалов** — распределенное представление данных.

- **Файл-сервер** — доступ к удаленной базе данных и файловым ресурсам.
- **Сервер БД** — удаленное представление данных.
- **Сервер приложений** — удаленное приложение.

Клиент – это браузер, но встречаются и исключения (в тех случаях, когда один веб-сервер (ВС1) выполняет запрос к другому (ВС2), роль клиента играет веб-сервер ВС1). В классической ситуации (когда роль клиента выполняет браузер) для того, чтобы пользователь увидел графический интерфейс приложения в окне браузера, последний должен обработать полученный ответ веб-сервера, в котором будет содержаться информация, реализованная с применением HTML, CSS, JS (самые используемые технологии). Именно эти технологии «дают понять» браузеру, как именно необходимо «отрисовать» все, что он получил в ответе.

Веб-сервер – это сервер, принимающий HTTP-запросы от клиентов и выдающий им HTTP-ответы. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает. Наиболее распространенными видами ПО веб-серверов являются Apache, IIS и NGINX. На веб-сервере функционирует тестируемое приложение, которое может быть реализовано с применением самых разнообразных языков программирования: PHP, Python, Ruby, Java, Perl и пр.

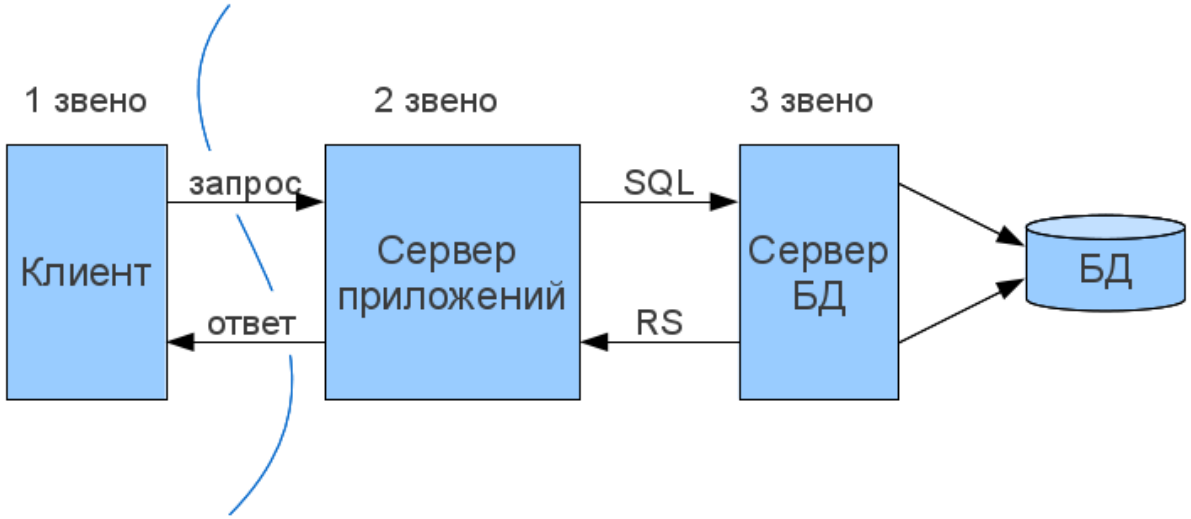
База данных фактически не является частью веб-сервера, но большинство приложений просто не могут выполнять все возложенные на них функции без нее, так как именно в базе данных хранится вся динамическая информация приложения (учетные, пользовательские данные и пр).

База данных - это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно категоризировать. Базы данных функционируют под управлением так называемых систем управления базами данных (далее – СУБД). Самыми популярными СУБД являются MySQL, MS SQL Server, PostgreSQL, Oracle (все – клиент-серверные).

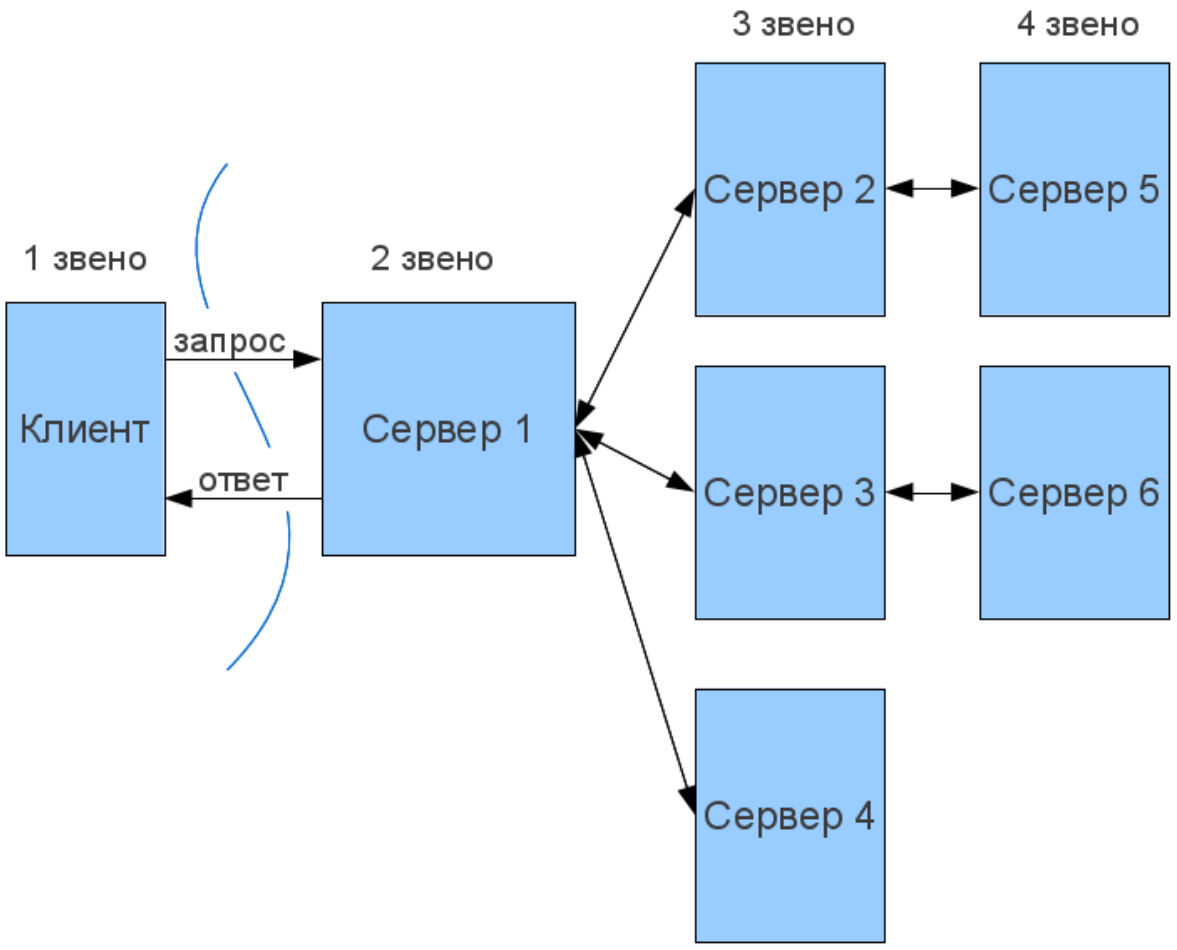
Трехзвенная архитектура - сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате.

Третьим звеном в трехзвенной архитектуре становится сервер приложений, т.е. компоненты распределяются следующим образом:

1. Представление данных — на стороне клиента.
2. Прикладной компонент — на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО).
3. Управление ресурсами — на сервере БД, который и представляет запрашиваемые данные.



Трехзвенная архитектура может быть расширена до **многозвенной (N-tier, Multi-tier)** путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня.



Двухзвенная архитектура проще, так как все запросы обслуживаются одним сервером, но именно из-за этого она менее надежна и предъявляет повышенные требования к производительности сервера.

Трехзвенная архитектура сложнее, но, благодаря тому, что функции распределены между серверами второго и третьего уровня, эта архитектура предоставляет:

1. Высокую степень гибкости и масштабируемости.
2. Высокую безопасность (т.к. защиту можно определить для каждого сервиса или уровня).
3. Высокую производительность (т.к. задачи распределены между серверами)

Многоуровневая архитектура (N-Tier)

В отдельный класс архитектуры «клиент-сервер» можно вынести многоуровневую архитектуру, в которой несколько серверов приложений используют результаты работы друг друга, а также данные от различных серверов баз данных, файловых серверов и других видов серверов.

По сути, предыдущий вариант, трёхуровневая архитектура – не более, чем частный случай многоуровневой архитектуры.

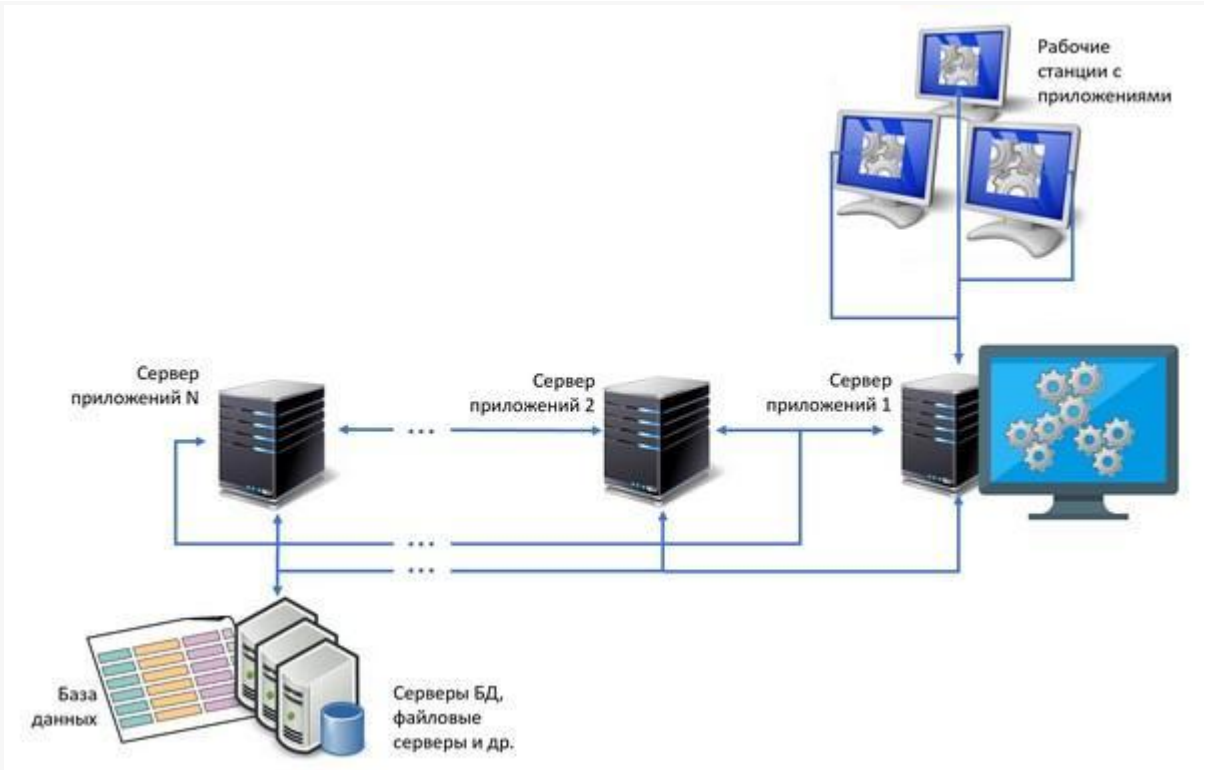


Рис. 5. Многоуровневая архитектура «клиент-сервер» (N-Tier).

Преимуществом многоуровневой архитектуры является гибкость предоставления услуг, которые могут являться комбинацией работы различных приложений серверов разных уровней и элементов этих приложений.

Очевидным недостатком является сложность, многокомпонентность такой архитектуры.

15.Опишите отличия двухзвенной архитектуры от н-звенной.
16.Опишите достоинства н-звенной архитектуры. (для 15 и 16)

Двухзвенная архитектура проще, так как все запросы обслуживаются одним сервером, но именно из-за этого она менее надежна и предъявляет повышенные требования к производительности сервера.

(Трех-)n-звенная архитектура сложнее, но благодаря тому, что функции распределены между серверами второго и n уровня, эта архитектура представляет:

1. Высокую степень гибкости и масштабируемости.
2. Высокую безопасность (т.к. защиту можно определить для каждого сервиса или уровня).
3. Высокую производительность (т.к. задачи распределены между серверами).

17.Опишите несколько различных сервисов.
World Wide Web, E-mail, Usenet, News, FTP, ICQ, Gopher

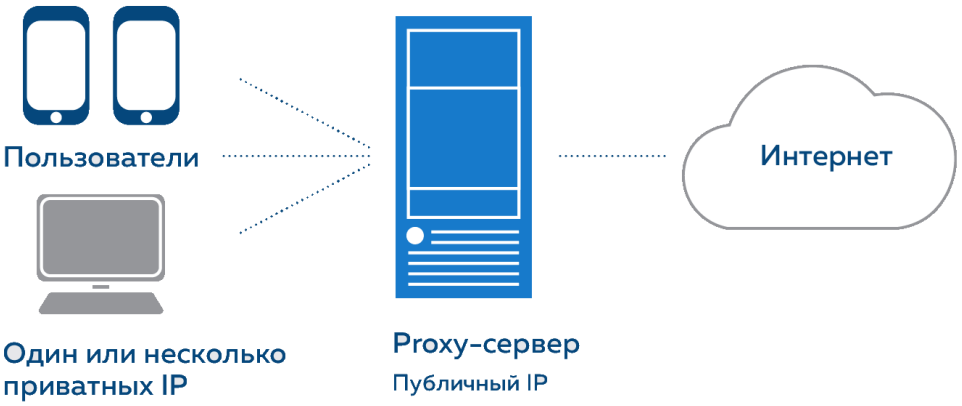
18.Что такое прокси-сервер и как он классифицируется?
Прокси сервер - это элемент сетевой инфраструктуры, который выполняет роль посредника между клиентским компьютером (терминал, браузер, приложение), находящимся во внутренней сети и другим сервером, который живёт во внешней сети или наоборот.
Прокси сервер может применяться для решения следующих задач:

- усиление безопасности
- защита приватности
- балансировка нагрузки на посещаемый ресурс

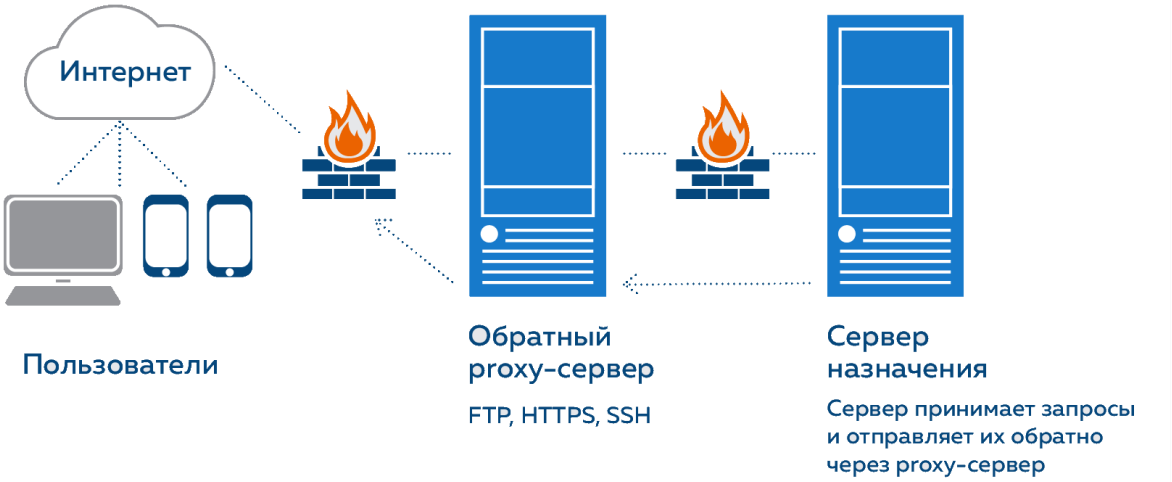
Прокси сервера бывают двух видов:

- **Прямой (Forward)** - прямой прокси - это такой промежуточный сервер, которых находится между клиентом и сервером назначения, которому обращается

клиент. Чтобы получить контент с сервера назначения, клиент отправляет запрос прокси-серверу с указанием сервера назначения в качестве цели, а прокси запрашивает контент и возвращает его клиенту. Клиент должен быть специально настроен (например, можно указать прокси в браузере) для использования такого прокси для доступа к другим сайтам.



- **Обратный (Reverse)** - обратный прокси, напротив, выглядит для клиента, как обычный веб-сервер. Никаких специальных настроек на клиенте не требуется. Клиент делает обычные запросы на получение контента, которые отправляются в пространство имен обратного прокси. Затем прокси решает, куда отправить эти запросы, и возвращает контент так, как если бы он и был сервером назначения.



Типичным примером использования обратного прокси-сервера является предоставление пользователям в Интернете доступа к серверу, который находится за межсетевым экраном. Обратные прокси-серверы также можно использовать для балансировки нагрузки между несколькими внутренними серверами или для обеспечения кэширования для более медленного внутреннего сервера. Кроме того, обратные прокси-серверы можно использовать просто для переноса нескольких серверов в одно и то же пространство URL.

19.Раскройте и классифицируйте понятие масштабируемость.

Масштабируемость — способность устройства увеличивать свои возможности путем наращивания числа функциональных блоков, выполняющих одни и те же задачи

Горизонтальное и вертикальное масштабирование веб приложений. **Вертикальное масштабирование** — scaling up — увеличение количества доступных для ПО ресурсов за счет увеличения мощности применяемых с серверов. **Горизонтальное масштабирование** — scaling out — увеличение количества нод, объединенных в кластер серверов при нехватке CPU, памяти или дискового пространства. И то и другое является инфраструктурными решениями. Они требуются в разных ситуациях когда веб проект растет.

20.Что такое протокол передачи данных и сетевой протокол?

Протокол передачи данных — набор определённых правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами.

Сетевой протокол — набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть системами.
Стандартизированный протокол позволяет разрабатывать интерфейсы (на физическом или прикладном уровне), не привязанные к конкретной аппаратной или программной платформе. OSI & TCP/IP.

21.Опишите классификацию протоколов. (для 20 и 21)

Протоколы передачи данных

Протокол передачи данных - набор определенных правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

Сетевой протокол - набор правил и действий (и их очередности), позволяющий осуществлять соединение и обмен данными между двумя и более включенными в сеть устройствами.

	TCP/IP	OSI	
7	Прикладной	Прикладной	HTTP, SMTP, SNMP, FTP, Telnet, SSH, SCP, SMB, NFS, RTSP, BGP
6		Представления	TLS, SSL
5		Сеансовый	RPC, NetBIOS, PPTP, L2TP
4	Транспортный	Транспортный	TCP, UDP, GRE
3	Сетевой	Сетевой	IP, ICMP, IGMP, OSPF, RIP, IPX
2	Канальный	Канальный	Ethernet, Token ring, HDLC, PPP, X.25, Frame relay, ISDN
1		Физический	электрические провода, радиосвязь, волоконно-оптические провода, инфракрасное излучение

22.Опишите кратко две основные, в настоящее время, сетевые модели.

Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью предыдущего («нижележащего») уровня передачи данных и, в свою очередь, предоставляет нужную функциональность следующему («вышележащему») уровню.
TCP/IP ссылается на огромное количество протоколов, описание которых находится в документах под названием RFC. Они находятся в свободном доступе в Интернете. Протоколы и правила были разделены на категории – уровни. Каждый уровень обладает своим набором функций, или сервисов, реализующихся за счет протоколов этого уровня.

23.Назовите основные прикладные протоколы, используемые в сети Интернет.

Протокол передачи файлов (File Transfer Protocol, FTP), протокол эмуляции терминала telnet, простой протокол передачи почты (Simple Mail Transfer Protocol, SMTP), протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP)

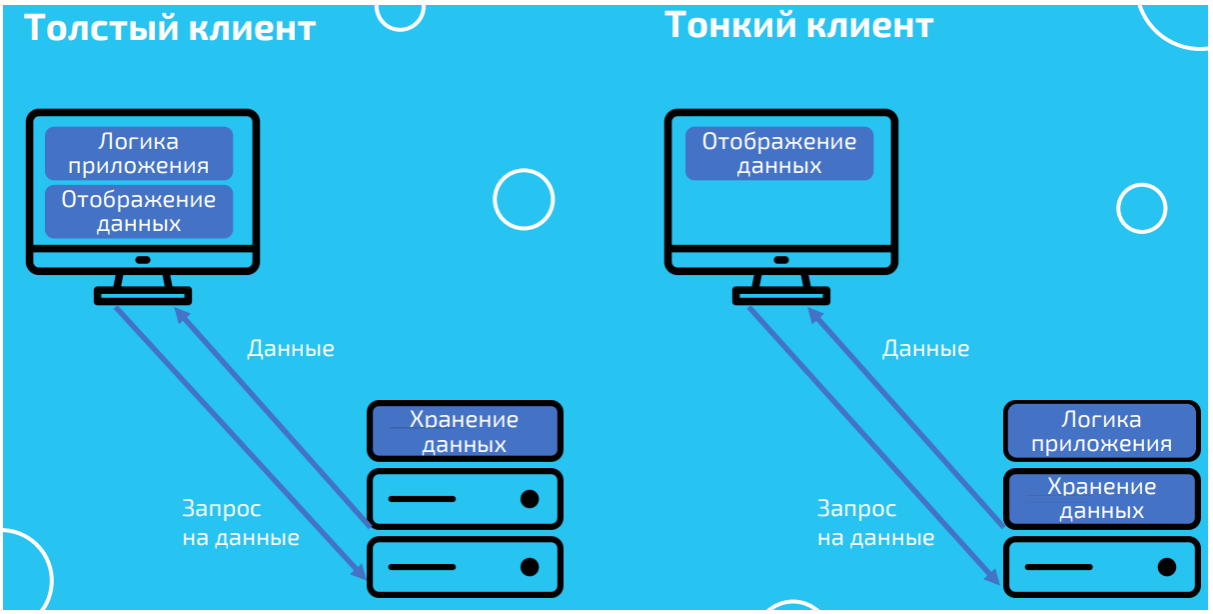
24.Опишите и определите понятия "толстый" и "тонкий" клиент.

Толстый клиент — **клиент**, выполняющий запрашиваемые со стороны пользователя манипуляции независимо от ведущего **сервера**. Основной **сервер** в такой вариации системной **архитектуры** может применяться как особое хранилище информации, обработка и конечное предоставление которых просто переносится на локальную машину пользователя.

Тонкий клиент (англ. thin client) в компьютерных технологиях — бездисковый компьютер-**клиент** в сетях с **клиент**-серверной или терминальной **архитектурой**, который переносит все или большую часть задач по обработке информации на **сервер** (Wikipedia). Если проще, то **тонкий клиент** – это недокомьютер, который загружает легкую операционную систему (обычно используется Linux, в обзоре возьмем это за априори) и соединяется с терминальным **сервером**.

Обычно **тонкие клиенты** создаются для экономии на железе и ПО, в редких случаях по иным соображениям

25.Опишите сходства и различия "толстого" и "тонкого" клиентов.



26.Что такое архитектура программного обеспечения?

Архитектура программного обеспечения (англ. *software architecture*) — совокупность важнейших решений об организации программной системы. Архитектура включает:

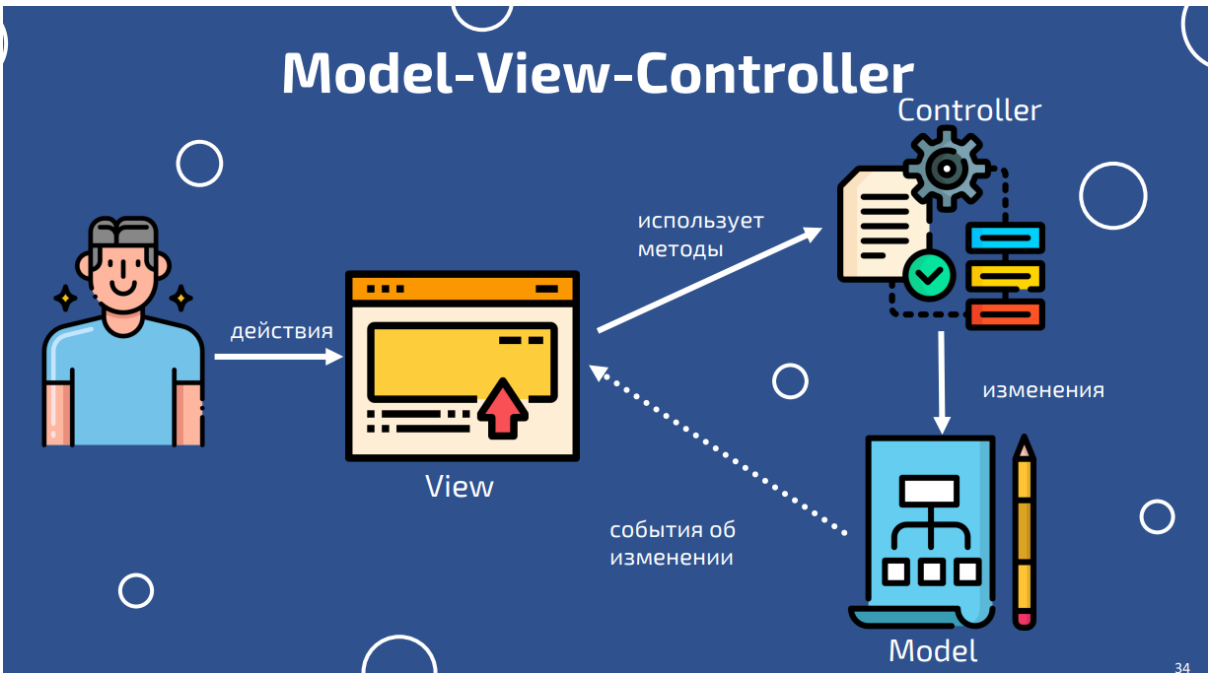
27.Что включает в себя понятие архитектуры программного обеспечения?

- выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
- соединение выбранных элементов структуры и поведения во всё более крупные системы;
- архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение

28.Что такое паттерн проектирования?

Шаблон **проектирования** или **паттерн** (англ. design **pattern**) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы **проектирования в** рамках некоторого часто возникающего контекста. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях.

29.Опишите кратко паттерн проектирования MVC.

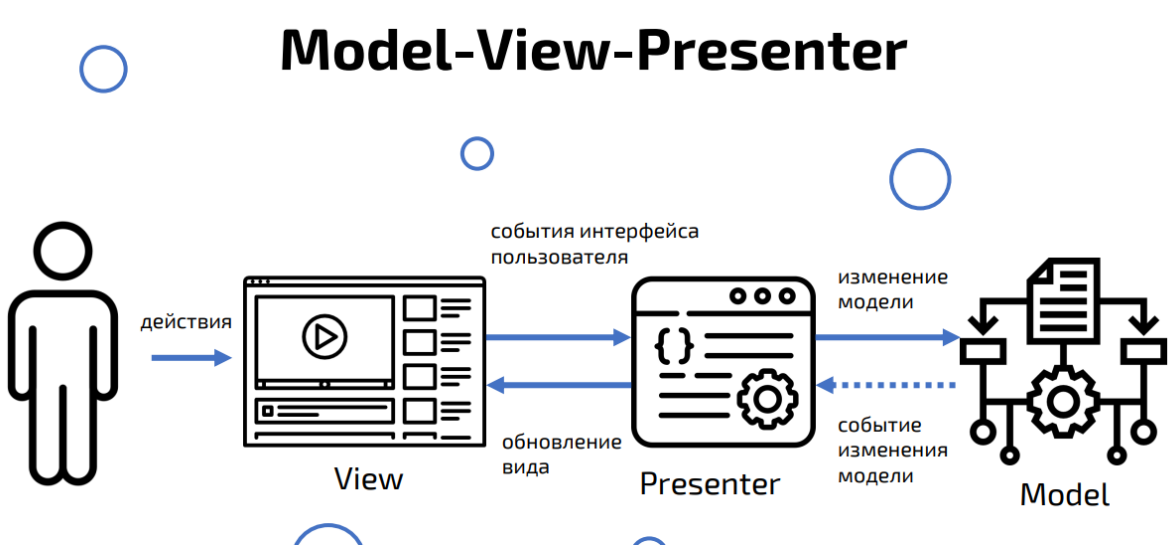


Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние. Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели. Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

30.Опишите признаки использования подхода MVC.

- Контроллер определяет, какие представление должно быть отображено в данный момент;
- События представления могут повлиять только на контроллер.контроллер может повлиять на модель и определить другое представление.
- Возможно несколько представлений только для одного контроллера;

31.Опишите кратко паттерн проектирования MVP.

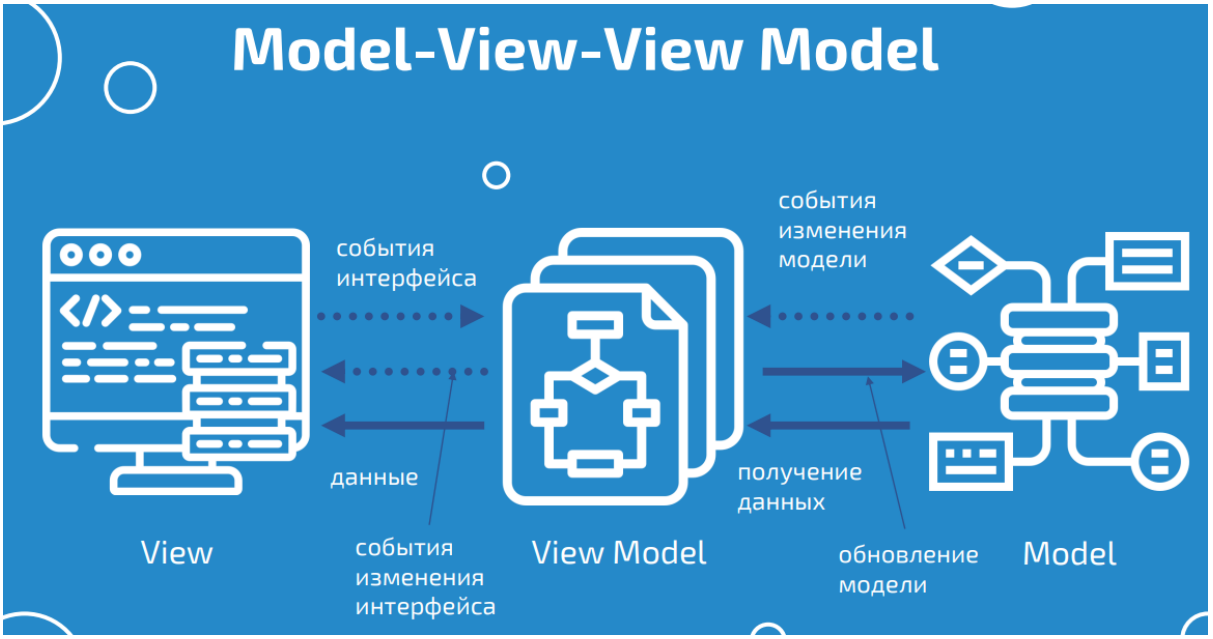


Model-View-Presenter — шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса. Элемент Presenter в данном шаблоне берёт на себя функциональность посредника (аналогично контроллеру в MVC) и отвечает за управление событиями пользовательского интерфейса (например, использование мыши) так же, как в других шаблонах обычно отвечает представление

32.Опишите признаки использования подхода MVP.

- Двухсторонняя коммуникация с представлением;
- Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
- Презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;
- *Один экземпляр презентера связан с одним отображением.*

33.Опишите кратко паттерн проектирования MVVM.



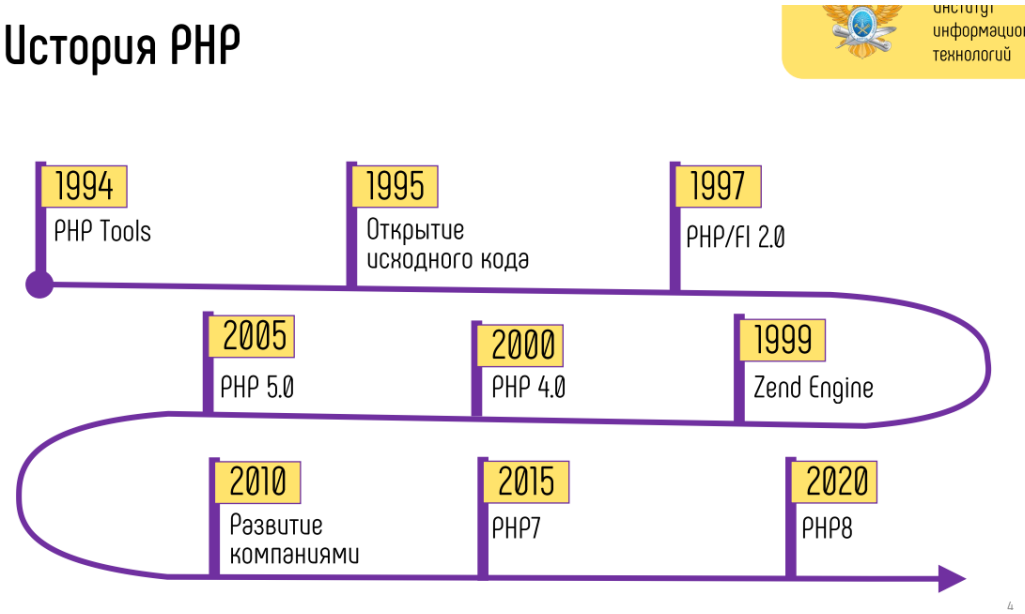
Model-View-ViewModel (MVVM) — шаблон проектирования архитектуры приложения. Представлен в 2005 году Джоном Госсманом (John Gossman) как модификация шаблона Presentation Model. Ориентирован на современные платформы разработки, такие как Windows Presentation Foundation, Silverlight от компании Microsoft, ZK framework.

34.Опишите признаки использования подхода MVVM.

- Двухсторонняя коммуникация с представлением;
- View-модель — это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами View-модели / модели
- View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings)
- *Один экземпляр View-модели связан с одним отображением.*

\\\WELCOME TO DUHOTA///

35.Кратко опишите историю развития языка программирования PHP.



36.Опишите что такое файл php.ini в рамках языка программирования PHP.

Файл «**php**.**ini**» — **это** текстовый файл с настройками **PHP**. **Php**.**ini** содержит набор директив, каждая их которых пишется с новой строки и представляет собой пару ключ-значение, разделенные знаком равно.

37.Опишите кратко содержимое файла php.ini в рамках языка программирования PHP. Приведите конкретный пример.

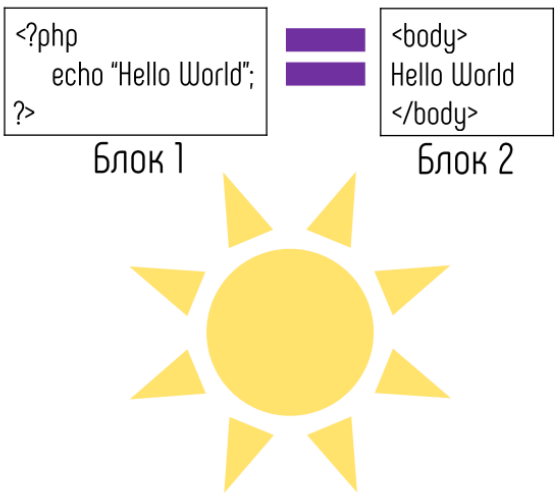
Конфигурационный файл php.ini

```
[MySQLi]
; Максимальное количество постоянных ссылок. -1 означает отсутствие ;ограничений.
mysqli.max_persistent = -1

; Разрешить доступ, с точки зрения PHP, к локальным файлам с инструкциями ;ЗАГРУЗКИ ДАННЫХ
;mysqli.allow_local_infile = On

; Разрешить или запретить постоянные ссылки.
mysqli.allow_persistent = On

; Максимальное количество ссылок. -1 означает отсутствие ограничений.
mysqli.max_links = -1

; Если используется mysqlnd: Количество слотов кэша для внутреннего кэша ;результирующего
набора
mysqli.cache_size = 2000

;Номер порта по умолчанию для mysqli_connect()
mysqli.default_port = 3306
```

38.Опишите как написать простую программу на PHP.

39.Опишите комбинирование PHP и HTML. (для 38-39)

Первая программа на PHP




```
<html>
<head>
<title>Пример комбинации PHP и HTML</title>
</head>
<body>
<h1>Пример простого списка</h1>
<!-- Далее следует список. Это комментарий-->
<ul>
<?php
echo "<li> Первый элемент списка. <br>";
echo "Продолжение первого элемента списка</li><br>";
// Это однострочный комментарий
# Это тоже однострочный комментарий
echo "<li> <i>Второй элемент списка</i></li>";
/* А это многострочный комментарий */
?> </ul>
</body>
</html>
```

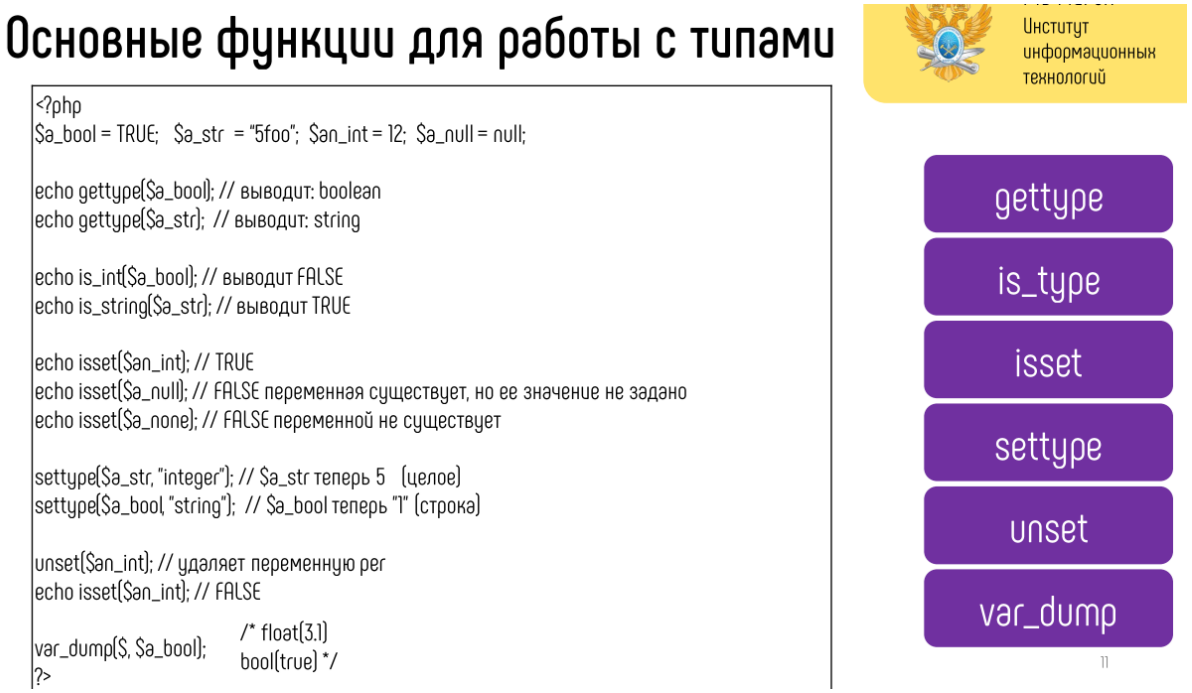
40.Опишите основные правила, связанные с переменными в PHP.

Переменные и типы

- Объявление переменной начинается со знака $
- Имена переменных должны состоять из латинских букв, цифр и нижнего подчеркивания
- Имена переменных чувствительны к регистру

41.Опишите основные типы данных в PHP.

42.Опишите основные функции для работы с переменными на уровне типов.



43.Опишите что такое и как используются предопределенные переменные в PHP.
44.Опишите что такое переменные переменных и приведите пример их использования.
(для 43-44)

Предопределенные и переменные переменных

```php
<?php
// определим обычную переменную
$a = 'hello';
// переменная переменной берет значение переменной и рассматривает
его как имя переменной.
$$a = 'world'; // значение переменной $hello теперь такое

echo "$a ${$a}"; // hello world
echo "$a $hello"; //hello world
?>
```

45.Что такое выражение и приведите примеры выражений на PHP.

Выражения

```php
<?php
$a = 5; /* константа 5 является выражением так как имеет значение 5 и после действия оператора присваивания в
переменной $a будет записано значение выражения 5*/

$b = $a; // следует понимать, что следующая запись также является выражением и это дает право на запись далее

$a = $b = 10; // переменным $a и $b присваивается значение константы 10
$a = 12;
echo "$a $b"; // вывод: 12 10, то есть переменным присвоены константы

$a = 3 * sin($b = $c + 10) + $d; // данная запись также возможна ниже приведен ее эквивалент

$b = $c + 10; // эквивалент записи выше
$a = 3 * sin($c + 10) + $d;
?>
```

46.Что такое оператор в рамках языка программирования и кратко опишите основных операторов, реализованных в PHP.

Инстру́кция или **опера́тор** (англ. statement) — наименьшая автономная часть **языка программирования**; команда или набор команд. Программа обычно представляет собой последовательность инструкций. Многие **языки** (например, Си) различают инструкцию и определение. Различие в том, что инструкция исполняет код, а определение создаёт идентификатор (то есть можно рассматривать определение как инструкцию присваивания). Ниже приведены основные общие инструкции **языков программирования** на **языке** Pascal.

Операторы инкремента и декремента

```
<?php
// Постфиксный инкремент и декремент
$a = 5;
echo $a++; // 5 сначала идет вывод, потом уже операция прибавления 1
echo $a--; // 6 сначала идет вывод, потом уже операция отнимания 1

// Префиксный инкремент и декремент
$a = 5;
echo ++$a; // 6 сначала идет операция прибавления 1, потом вывод
echo --$a; // 5 сначала идет операция отнимания 1, потом вывод

?>
```

Логические операторы

Название	Обозначение	Описание
И	$a and $b	true, если и $a, и $b true.
И	$a && $b	true, если и $a, и $b true.
ИЛИ	$a or $b	true, если или $a, или $b true.
ИЛИ	$a || $b	true, если или $a, или $b true.
Отрицание	! $a	true, если $a не true.
Исключающее ИЛИ	$a xor $b	true, если $a, или $b true, но не оба.

47.Опишите особенности оператора присваивания в PHP.

Оператор присваивания

```
<?php
$a = ($b = 4) + 5; // $a теперь равно 9, а $b было присвоено 4.

$a = 3;
$a += 5; // устанавливает $a в 8, как если бы мы написали: $a = $a + 5;
$b = "Привет";
$b .= "-привет!"; // устанавливает $b в "Привет-привет!",  как и $b = $b . "-привет!";
?>
```

48.Опишите особенности операторов сравнения в PHP.

Операторы сравнения

Название	Обозначение	Описание
Равно	$a == $b	true если $a равно $b после преобразования типов.
Тождественно равно	$a === $b	true если $a равно $b и имеет тот же тип.
Не равно	$a != $b	true если $a не равно $b после преобразования типов.
Не равно	$a <> $b	true если $a не равно $b после преобразования типов.
Тождественно не равно	$a !== $b	true если $a не равно $b, или они разных типов.
Меньше	$a < $b	true если $a строго меньше $b.
Больше	$a > $b	true если $a строго больше $b.
Меньше или равно	$a <= $b	true если $a меньше или равно $b.
Больше или равно	$a >= $b	true если $a больше или равно $b.
Космический корабль (spaceship)	$a <=> $b	Число типа int меньше, больше или равное нулю, когда $a соответственно меньше, больше или равно $b.

49.Опишите особенности операторов, работающих с массивами в PHP.

Операторы, работающие с массивами

Название	Обозначение	Описание
Объединение	$a + $b	Объединение массива $a и массива $b.
Равно	$a == $b	true в случае, если $a и $b содержат одни и те же пары ключ/значение.
Тождественно равно	$a === $b	true в случае, если $a и $b содержат одни и те же пары ключ/значение в том же самом порядке и того же типа.
Не равно	$a != $b	true, если массив $a не равен массиву $b.
Не равно	$a <> $b	true, если массив $a не равен массиву $b.
Тождественно не равно	$a !== $b	true, если массив $a не равен тождественно массиву $b.

50.Кратко опишите основные управляющие конструкции в PHP.
If, else, switch, while, do{}while, for
51.Кратко опишите циклы в PHP, приведите примеры.

Циклы: foreach

```
<?php
// вывод всех переменных окружения
foreach($_SERVER as $key => $value){
    echo "<b>$k</b> => <tt>$value</tt><br />\n";
}

// вывод всех элементов списка
// под списком понимается массив, где ключами являются последовательные
// числовые индексы
$array = array("foo", "bar", "hallo", "world");
foreach($array as $value){
    echo "$value<br />\n";
}
?>
```

Циклы: for

```
<?php
for($i = 0, $j = 0, $k = "Points"; $i<100; $j++, $i += $j) $k = $k.".";
// первыми выполняются единожды инициализирующие команды
// потом проверяется условие цикла, в данном примере оно простое
// если условие истинно, то выполняется итерация цикла
// после выполнения итерации цикла выполняются команды после прохода
// в данном примере нет тела цикла, т.к. в теле цикла всего 1 команда

// пример вывода чисел от 0 до 9 с использованием цикла for
for($i = 0; $i < 10; $i++){
    echo $i;
}
?>
```

Циклы: do-while

```
<?php
// вывод чисел от 0 до 10
$i = 0;
do {
    echo $i;
} while ($i < 10);

?>
```

Циклы: while

```
<?php
$i = 1;
while ($i <= 10) {
    echo $i++;
/* выводиться будет значение переменной $i перед её
увеличением (post-increment) */
}
?>
```

52.Опишите использование инструкций break и continue. Приведите примеры.

Циклы: инструкции break и continue

```php
<?php
    $arr = array('один', 'два', 'три', 'четыре', 'стоп', 'пять');
    for ($i = 0; $i < count($arr); $i++) {
        if ($arr[$i] == 'стоп') {
            break;    // эквивалентно записи break 1; или break(1);
        }
        echo "{$arr[$i]}<br />\n";
    }

    $matrix = array (
    array (1, 2, 3, 6, 8, 9),
    array (4, 7, 3, 3, 1, 5)
     );
    for ($i = 0; $i < count($matrix); $i++){
        for ($j = 0; $j < count($matrix[$i]); $j++){
            if ($matrix[$i][$j] == 0) break(2);         }
    }
?>
```

53.Опишите операторы switch и match. Их сходства и различия.

switch

```php
<?php
switch ($i) {
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
        break;
    case 2:
        echo "i равно 2";
        break;
    default:
       echo "i не равно 0, 1 и 2";
}
?>
```

Match

```php
<?php
$expressionResult = match ($condition) {
    1, 2 => foo(),
    3, 4 => bar(),
    5 => 5,
    default => baz(),
}

// удобное использование для вхождения в диапазоны чисел
$result = match (true) {
    $age >= 65 => 'elderly',
    $age >= 25 => 'adult',
    $age >= 18 => 'full age',
    default => 'child',
};
?>
```

54.Опишите операторы include и require. Приведите пример использования.

Отличие include от **require отличается** в том, что **require** падает в Fatal error при невозможности подключения файла по любой причине. А **include** только выдает Warning и спокойно продолжает работу.

Конструкции включений в PHP

Конструкции включений позволяют собирать PHP программу (скрипт) из нескольких отдельных файлов.

В PHP существуют две основные конструкции включений: **require** и **include**.

Конструкция включений require

Конструкция **require** позволяет включать файлы в сценарий PHP исполнения сценария PHP. Общий синтаксис **require** такой:

```
require имя_файла;
```

При запуске (именно при запуске, а не при исполнении!) программы интерпретатор просто заменит инструкцию на содержимое файла имя_файла (этот файл может также содержать сценарий на PHP, обрамленный, как обычно, тэгами **<?** и **?>**). Причем сделает он это непосредственно перед запуском программы (в отличие от **include**, который рассматривается

ниже). Это бывает довольно удобно для включения в вывод сценария различных шаблонных страниц HTML-кодом. Приведем пример:

```
Файл header.html:
<html>
<head><title>It is a title</title></head>
<body bgcolor=green>
Файл footer.html:
&copy; Home Company, 2005.
</body></html>
Файл script.php
<?php
require "header.htm";
// Сценарий выводит само тело документа
require "footer.htm";
?>
```

Таким образом, конструкция **require** позволяет собирать сценарии PHP из нескольких отдельных файлов, которые могут быть как *html*-страницами, так и *php*-скриптами.

Конструкция **require** поддерживает включения удаленных файлов (начиная с версии PHP 4.3.0). Например:

```
<?php
// Следующий пример на работает, поскольку пытается включить локальный файл
require 'file.php?foo=1&bar=2';
// Следующий пример работает
require 'http://www.example.com/file.php?foo=1&bar=2';
?>
```

! Конструкция **require** позволяет включать удаленные файлы, если такая возможность включена в конфигурационном файле PHP. Подробная информация далее.

Конструкция включений include

Конструкция **include** также предназначена для включения файлов в код сценария PHP.

В отличие от конструкции **require** конструкция **include** позволяет включать файлы в код PHP скрипта выполнения сценария. Синтаксис конструкции include выглядит следующим образом:

```
include имя_файла;
```

Поясним принципиальную разницу между конструкциями **require** и **include** на конкретном практическом примере. Создадим 10 файлов с именами 1.txt, 2.txt и так далее до 10.txt, содержимое этих файлов - просто десятичные цифры 1, 2 …… 10 (по одной цифре в каждом файле). Создадим такой сценарий PHP:

```
<?php
// Создаем цикл, в теле которого конструкция include
for($i=1; $i<=10; $i++) {
include "$i.txt";
}
// Включили десять файлов: 1.txt, 2.txt, 3.txt ... 10.txt
```

```
// Результат - вывод 12345678910
?>
```

В результате мы получим вывод, состоящий из 10 цифр: "12345678910". Из этого мы можем слелать вывод, что каждый из файлов был включен по одному разу прямо во время выполнения цикла! Если мы поставим теперь вместо **include** **require**, то сценарий сгенерирует критическую ошибку (fatal error). Сравните результат.

PHP преобразует сценарий во внутреннее представление, анализируя строки сценария по очереди, пока не доходит до конструкции **include**. Дойдя до **include**, PHP прекращает транслировать сценарий и переключается на указанный в **include** файл. Таким образом из-за подобного поведения транслятора, быстродействие сценария снижается, особенно при большом колличестве включаемых с помощью **include** файлов. С require таких проблем нет, поскольку файлы с помощью require включаются до выполнения сценария, то есть на момент трансляции файл уже включен в сценарий.

Таким образом, целесообразнее использовать конструкцию **require** там, где не требуется динамическое включение файлов в сценарий, а конструкцию **include** использовать только с целью динамического включения файлов в код PHP скрипта.

Конструкция **include** поддерживает включения удаленных файлов (начиная с версии PHP 4.3.0). Например:

```
<?php
// Следующий пример на работает, поскольку пытается включить локальный файл
include 'file.php?foo=1&bar=2';
// Следующий пример работает
include 'http://www.example.com/file.php?foo=1&bar=2';
?>
```

! Конструкция **include** позволяет включать удаленные файлы, если такая возможность включена в конфигурационном файле PHP.

55.Опишите использование альтернативного синтаксиса управляющих структур.

Альтернативный синтаксис управляющих структур

```
<?php if (condition): ?>

html code to run if condition is true

<?php else: ?>

html code to run if condition is false

<?php endif ;?>

<?php while(condition):?>

html code to run while condition is true

<?php endwhile;?>
```

56.Опишите синтаксис описания функции.

Функции

```
<?php
/*
function nameOfFunction($arg1[=val1], $arg2[=val2], ..., $argN[=valN]){
    function body operators;
}
*/
// пример простой функции
function plus($a, $b){
    return $a + $b;
}
// все будет корректно работать если вы будете подавать корректные значения
?>
```

57.Опишите особенности вопроса области видимости в PHP.

Область видимости ¶

Область видимости свойства, метода или константы (начиная с PHP 7.1.0) может быть определена путём использования следующих ключевых слов в объявлении: `public`, `protected` или `private`. Доступ к свойствам и методам класса, объявленным как public (общедоступный), разрешён отовсюду. Модификатор protected (защищённый) разрешает доступ самому классу, наследующим его классам и родительским классам. Модификатор private (закрытый) ограничивает область видимости так, что только класс, где объявлен сам элемент, имеет к нему доступ.

Область видимости свойства ¶

Свойства класса должны быть определены через модификаторы public, private или protected. Если же свойство определено с помощью `var`, то оно будет объявлено общедоступным свойством.
Пример #1 Объявление свойства класса

```php
<?php
/**
 * Определение MyClass
 */
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Работает
echo $obj->protected; // Неисправимая ошибка
echo $obj->private; // Неисправимая ошибка
$obj->printHello(); // Выводит Public, Protected и Private


/**
 * Определение MyClass2
 */
class MyClass2 extends MyClass
{
    // Мы можем переопределить общедоступные и защищённые свойства, но не зак
```

```
рытые
    public $public = 'Public2';
    protected $protected = 'Protected2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // Работает
echo $obj2->private; // Неопределён
echo $obj2->protected; // Неисправимая ошибка
$obj2->printHello(); // Выводит Public2, Protected2, Undefined

?>
```

Область видимости метода ¶

Методы класса могут быть определены как public, private или protected. Методы, объявленные без указания области видимости, определяются как public.

Пример #2 Объявление метода

```
<?php
/**
 * Определение MyClass
 */
class MyClass
{
    // Объявление общедоступного конструктора
    public function __construct() { }

    // Объявление общедоступного метода
    public function MyPublic() { }

    // Объявление защищённого метода
    protected function MyProtected() { }

    // Объявление закрытого метода
    private function MyPrivate() { }

    // Это общедоступный метод
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
```

```
}

$myclass = new MyClass;
$myclass->MyPublic(); // Работает
$myclass->MyProtected(); // Неисправимая ошибка
$myclass->MyPrivate(); // Неисправимая ошибка
$myclass->Foo(); // Работает общедоступный, защищённый и закрытый


/**
 * Определение MyClass2
 */
class MyClass2 extends MyClass
{
    // Это общедоступный метод
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Неисправимая ошибка
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Работает
$myclass2->Foo2(); // Работает общедоступный и защищённый, закрытый не работа
ет

class Bar
{
    public function test() {
        $this->testPrivate();
        $this->testPublic();
    }

    public function testPublic() {
        echo "Bar::testPublic\n";
    }

    private function testPrivate() {
        echo "Bar::testPrivate\n";
    }
}

class Foo extends Bar
{
    public function testPublic() {
        echo "Foo::testPublic\n";
    }

    private function testPrivate() {
```

```
        echo "Foo::testPrivate\n";
    }
}

$myFoo = new Foo();
$myFoo->test(); // Bar::testPrivate
                // Foo::testPublic
?>
```

Область видимости констант ¶

Начиная с PHP 7.1.0, константы класса могут быть определены как public, private или protected. Константы, объявленные без указания области видимости, определяются как public.

Пример #3 Объявление констант, начиная с PHP 7.1.0

```
<?php
/**
 * Объявление класса MyClass
 */
class MyClass
{
    // Объявление общедоступной константы
    public const MY_PUBLIC = 'public';

    // Объявление защищённой константы
    protected const MY_PROTECTED = 'protected';

    // Объявление закрытой константы
    private const MY_PRIVATE = 'private';

    public function foo()
    {
        echo self::MY_PUBLIC;
        echo self::MY_PROTECTED;
        echo self::MY_PRIVATE;
    }
}

$myclass = new MyClass();
MyClass::MY_PUBLIC; // Работает
MyClass::MY_PROTECTED; // Неисправимая ошибка
MyClass::MY_PRIVATE; // Неисправимая ошибка
$myclass->foo(); // Выводятся константы public, protected и private


/**
 * Объявление класса MyClass2
 */
class MyClass2 extends MyClass
{
    // Публичный метод
```

```
    function foo2()
    {
        echo self::MY_PUBLIC;
        echo self::MY_PROTECTED;
        echo self::MY_PRIVATE; // Неисправимая ошибка
    }
}

$myclass2 = new MyClass2;
echo MyClass2::MY_PUBLIC; // Работает
$myclass2->foo2(); // Выводятся константы public и protected, но не private
?>
```

Видимость из других объектов ¶

Объекты, которые имеют общий тип (наследуются от одного класса), имеют доступ к элементам с модификаторами private и protected друг друга, даже если не являются одним и тем же экземпляром. Это объясняется тем, что реализация видимости элементов известна внутри этих объектов.
Пример #4 Доступ к элементам с модификатором private из объектов одного типа

```
<?php
class Test
{
    private $foo;

    public function __construct($foo)
    {
        $this->foo = $foo;
    }

    private function bar()
    {
        echo 'Доступ к закрытому методу.';
    }

    public function baz(Test $other)
    {
        // Мы можем изменить закрытое свойство:
        $other->foo = 'привет';
        var_dump($other->foo);

        // Мы также можем вызвать закрытый метод:
        $other->bar();
    }
}

$test = new Test('test');

$test->baz(new Test('other'));
?>
```

Результат выполнения данного примера:

```
string(6) "привет"
Доступ к закрытому методу.
```

58.Опишите особенности приоритетности операторов в PHP.

Приоритет оператора ¶

Приоритет оператора определяет, насколько "тесно" он связывает между собой два выражения. Например, выражение `1 + 5 * 3` вычисляется как `16`, а не `18`, поскольку оператор умножения ("*") имеет более высокий приоритет, чем оператор сложения ("+"). Круглые скобки могут использоваться для принудительного указания порядка выполнения операторов. Например, выражение `(1 + 5) * 3` вычисляется как `18`.

Если операторы имеют равный приоритет, то будут ли они выполняться справа налево или слева направо определяется их ассоциативностью. К примеру, "-" является лево-ассоциативным оператором. Следовательно `1 - 2 - 3` сгруппируется как `(1 - 2) - 3` и пересчитается в `-4`. С другой стороны "=" - право-ассоциативный оператор, так что `$a = $b = $c` сгруппируется как `$a = ($b = $c)`.

Неассоциативные операторы с одинаковым приоритетом не могут использоваться совместно. К примеру `1 < 2 > 1` не будет работать в PHP. Выражение `1 <= 1 == 1`, с другой стороны, будет, поскольку `==` имеет более низкий приоритет чем `<=`.

Ассоциативность имеет смысл только для двоичных (и тернарных) операторов. Унарные операторы являются префиксными или постфиксными, поэтому это понятие не применимо. Например, `!!$a` можно сгруппировать только как `!(!$a)`.

Использование скобок, кроме случаев когда они строго необходимы, может улучшить читаемость кода, группируя явно, а не полагаясь на приоритеты и ассоциативность.

В следующей таблице приведён список операторов, отсортированный по убыванию их приоритетов. Операторы, размещённые в одной строке имеют одинаковый приоритет и порядок их выполнения определяется исходя из их ассоциативности.

Порядок выполнения операторов

Ассоциативность	Оператор	Дополнительная информация
(н/а)	clone new	clone и new
правая	**	арифметические операторы
(н/а)	+ - ++ -- ~ (int) (float) (string) (array) (object) (bool) @	арифметические операторы (унарные + и -), инкремент/декремент, побитовые операторы, приведение типов и оператор управления ошибками
левая	instanceof	типы
(н/а)	!	логические операторы
левая	* / %	арифметические операторы
левая	+ - .	арифметические операторы (бинарные + и -), операторы, работающие с массивами и строковые операторы (. до PHP 8.0.0)
левая	<< >>	побитовые операторы
левая	.	строковые операторы (начиная с PHP 8.0.0)
неассоциативна	< <= > >=	операторы сравнения
неассоциативна	== != === !== <> <=>	операторы сравнения
левая	&	побитовые операторы и ссылки
левая	^	побитовые операторы
левая	|	побитовые операторы
левая	&&	логические операторы
левая	||	логические операторы
правая	??	операторы сравнения с null
неассоциативна	? :	тернарный оператор (лево-ассоциативный до PHP 8.0.0)
правая	= += -= *= **= /= .= %= &= |= ^= <<= >>= ??=	операторы присваивания
(н/а)	yield from	yield from
(н/а)	yield	yield
(н/а)	print	print
левая	and	логические операторы
левая	xor	логические операторы
левая	or	логические операторы

Пример #1 Ассоциативность

```php
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
// ассоциативность тернарных операторов отличается от C/C++
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2 (до PHP 8.0.0)

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```

Приоритет и ассоциативность оператора определяет только то, как группируется выражение, а не порядок его вычисления. Обычно PHP не указывает, в каком порядке вычисляются выражения и кода, который предполагает специфичный порядок вычисления следует избегать, потому, что поведение может меняться в разных версиях PHP или в зависимости от окружающего кода.

Пример #2 Неопределённый порядок вычисления

```php
<?php
$a = 1;
```

```php
echo $a + $a++; // может вывести как 2 так и 3

$i = 1;
$array[$i] = $i++; // может установить индекс как 1, так 2
?>
```

Пример #3 +, - и . имеют одинаковый приоритет (до PHP 8.0.0)

```php
<?php
$x = 4;
// следующий код может выдать неожиданный результат:
echo "x минус 1 равно " . $x-1 . ", ну я надеюсь\n";
// поскольку он вычисляется таким образом (до PHP 8.0.0):
echo (("x минус один равно " . $x) - 1) . ", ну я надеюсь\n";
// требуемый приоритет следует задать скобками:
echo "x минус 1 равно " . ($x-1) . ", ну я надеюсь\n";
?>
```

Результат выполнения данного примера:

```
-1, ну я надеюсь
-1, ну я надеюсь
x минус один равно 3, ну я надеюсь
```

Замечание:
Несмотря на то, что = имеет более низкий приоритет, чем большинство других операторов, PHP всё же позволяет делать так: `if (!$a = foo())`, в этом примере результат выполнения `foo()` будет присвоен *$a*.

59.Опишите реализацию конкатенации строк в PHP.

Строковые операторы ¶

В PHP есть два оператора для работы со строками (string). Первый - оператор конкатенации ('.'), который возвращает строку, представляющую собой соединение левого и правого аргумента. Второй - оператор присваивания с конкатенацией ('.='), который присоединяет правый аргумент к левому. Для получения более полной информации ознакомьтесь с разделом Операторы присваивания.

```php
<?php
$a = "Привет, ";
$b = $a . "Мир!"; // $b теперь содержит строку "Привет, Мир!"

$a = "Привет, ";
$a .= "Мир!";     // $a теперь содержит строку "Привет, Мир!"
?>
```

60.Опишите принцип разделения инструкций в PHP.

Разделение инструкций ¶

Как в C или Perl, PHP требует окончания инструкций точкой запятой в конце каждой инструкции. Закрывающий тег блока PHP-кода автоматически применяет точку с запятой; т.е. нет необходимости ставить точку с запятой в конце последней строки блока с PHP-кодом. Закрывающий тег блока

"поглотит" немедленно следующий за ним переход на новую строку, если таковой будет обнаружен.
Пример #1 Пример, показывающий закрывающий тег, охватывающий завершающую новую строку

```
<?php echo "Какой-то текст"; ?>
Нет новой строки
<?= "А сейчас, новая строка" ?>
```

Результат выполнения данного примера:

```
Какой-то текстНет новой строки
А сейчас, новая строка
```

Примеры входа и выхода из парсера PHP:

```
<?php
    echo 'Это тест';
?>

<?php echo 'Это тест' ?>

<?php echo 'Мы опустили последний закрывающий тег';
```

Замечание:
Закрывающий тег PHP-блока в конце файла не является обязательным, и в некоторых случаях его опускание довольно полезно, например, при использовании include или require, так, что нежелательные пробелы не останутся в конце файла и вы всё ещё сможете добавить http-заголовки после подключения к ответу сервера. Это также удобно при использовании буферизации вывода, где также нежелательно иметь пробелы в конце частей ответа, сгенерированного подключаемыми файлами.

61.Опишите особенности типа данных массив в PHP.

Массивы ¶

На самом деле массив в PHP - это упорядоченное отображение, которое устанавливает соответствие между *значением* и *ключом*. Этот тип оптимизирован в нескольких направлениях, поэтому вы можете использовать его как собственно массив, список (вектор), хеш-таблицу (являющуюся реализацией карты), словарь, коллекцию, стек, очередь и, возможно, что-то ещё. Так как значением массива может быть другой массив PHP, можно также создавать деревья и многомерные массивы.
Объяснение этих структур данных выходит за рамки данного справочного руководства, но вы найдёте как минимум один пример по каждой из них. За дополнительной информацией вы можете обратиться к соответствующей литературе по этой обширной теме.

Синтаксис ¶

Определение при помощи array() ¶

Массив (тип array) может быть создан языковой конструкцией array(). В качестве параметров она принимает любое количество разделённых запятыми пар `key => value` (ключ => значение).

```
array(
    key  => value,
    key2 => value2,
    key3 => value3,
    ...
)
```

Запятая после последнего элемента массива необязательна и может быть опущена. Обычно это делается для однострочных массивов, то есть `array(1, 2)` предпочтительней `array(1, 2, )`. Для многострочных массивов с другой стороны обычно используется завершающая запятая, так как позволяет легче добавлять новые элементы в конец массива.

Замечание:
Существует короткий синтаксис массива, который заменяет `array()` на `[]`.

Пример #1 Простой массив

```php
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// Использование синтаксиса короткого массива
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

key может быть либо типа int, либо типа string. value может быть любого типа.
Дополнительно с ключом key будут сделаны следующие преобразования:

- Строки (string), содержащие целое число (int) (исключая случаи, когда число предваряется знаком `+`) будут преобразованы к типу int. Например, ключ со значением `"8"` будет в действительности сохранён со значением `8`. С другой стороны, значение `"08"` не будет преобразовано, так как оно не является корректным десятичным целым.
- Числа с плавающей точкой (float) также будут преобразованы к типу int, то есть дробная часть будет отброшена. Например, ключ со значением `8.7` будет в действительности сохранён со значением `8`.
- Тип bool также преобразовываются к типу int. Например, ключ со значением `true` будет сохранён со значением `1` и ключ со значением `false` будет сохранён со значением `0`.
- Тип null будет преобразован к пустой строке. Например, ключ со значением `null` будет в действительности сохранён со значением `""`.

- Массивы (array) и объекты (object) *не могут* использоваться в качестве ключей. При подобном использовании будет генерироваться предупреждение: `Недопустимый тип смещения (Illegal offset type)`.

Если несколько элементов в объявлении массива используют одинаковый ключ, то только последний будет использоваться, а все другие будут перезаписаны.

Пример #2 Пример преобразования типов и перезаписи элементов

```
<?php
$array = array(
    1    => "a",
    "1"  => "b",
    1.5  => "c",
    true => "d",
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(1) {
  [1]=>
  string(1) "d"
}
```

Так как все ключи в вышеприведённом примере преобразуются к `1`, значение будет перезаписано на каждый новый элемент и останется только последнее присвоенное значение `"d"`.

Массивы в PHP могут содержать ключи типов int и string одновременно, так как PHP не делает различия между индексированными и ассоциативными массивами.

Пример #3 Смешанные ключи типов int и string

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100   => -100,
    -100  => 100,
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

Параметр key является необязательным. Если он не указан, PHP будет использовать предыдущее наибольшее значение ключа типа int, увеличенное на 1.

Пример #4 Индексированные массивы без ключа

```php
<?php
$array = array("foo", "bar", "hallo", "world");
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hallo"
  [3]=>
  string(5) "world"
}
```

Возможно указать ключ только для некоторых элементов и пропустить для других:

Пример #5 Ключи для некоторых элементов

```php
<?php
$array = array(
         "a",
         "b",
     6 => "c",
         "d",
);
var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(4) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7]=>
  string(1) "d"
}
```

Как вы видите последнее значение `"d"` было присвоено ключу `7`. Это произошло потому, что самое большое значение ключа целого типа перед этим было `6`.

Пример #6 Расширенный пример преобразования типов и перезаписи элементов

```php
<?php
$array = array(
    1    => 'a',
    '1'  => 'b', // значение "b" перезапишет значение "a"
    1.5  => 'c', // значение "c" перезапишет значение "b"
    -1 => 'd',
    '01'  => 'e', // поскольку это не целочисленная строка, она НЕ перезапише
т ключ для 1
    '1.5' => 'f', // поскольку это не целочисленная строка, она НЕ перезапише
т ключ для 1
```

```
    true => 'g', // значение "g" перезапишет значение "c"
    false => 'h',
    '' => 'i',
    null => 'j', // значение "j" перезапишет значение "i"
    'k', // значение "k" присваивается ключу 2. Потому что самый большой цело
численный ключ до этого был 1
    2 => 'l', // значение "l" перезапишет значение "k"
);

var_dump($array);
?>
```

Результат выполнения данного примера:

```
array(7) {
  [1]=>
  string(1) "g"
  [-1]=>
  string(1) "d"
  ["01"]=>
  string(1) "e"
  ["1.5"]=>
  string(1) "f"
  [0]=>
  string(1) "h"
  [""]=>
  string(1) "j"
  [2]=>
  string(1) "l"
}
```

Этот пример включает все вариации преобразования ключей и перезаписи элементов

Доступ к элементам массива с помощью квадратных скобок ¶

Доступ к элементам массива может быть осуществлён с помощью синтаксиса `array[key]`.

Пример #7 Доступ к элементам массива

```
<?php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
?>
```

Результат выполнения данного примера:

```
string(3) "bar"
```

```
int(24)
string(3) "foo"
```

Замечание:
До PHP 8.0.0 квадратные и фигурные скобки могли использоваться взаимозаменяемо для доступа к элементам массива (например, в примере выше `$array[42]` и `$array{42}` делали то же самое). Синтаксис фигурных скобок устарел в PHP 7.4.0 и больше не поддерживается в PHP 8.0.0.

Пример #8 Разыменование массива

```
<?php
function getArray() {
    return array(1, 2, 3);
}

$secondElement = getArray()[1];

// или так
list(, $secondElement) = getArray();
?>
```

Замечание:
Попытка доступа к неопределённому ключу в массиве - это то же самое, что и попытка доступа к любой другой неопределённой переменной: будет сгенерирована ошибка уровня `E_NOTICE`, и результат будет `null`.

Замечание:
Массив, разыменовывающий скалярное значение, которое не является строкой (string), отдаст `null`. До PHP 7.4.0 не выдаётся сообщение об ошибке. Начиная с PHP 7.4.0, выдаётся ошибка `E_NOTICE`; с PHP 8.0.0 выдаётся ошибка `E_WARNING`.

Создание/модификация с помощью синтаксиса квадратных скобок ¶

Существующий массив может быть изменён путём явной установкой значений в нем.

Это выполняется присвоением значений массиву (array) с указанием в скобках ключа. Кроме того, ключ можно опустить, в результате получится пустая пара скобок (`[]`).

```
$arr[key] = value;
$arr[] = value;
// key может быть int или string
// value может быть любым значением любого типа
```

Если массив *`$arr`* ещё не существует или для него задано значение `null` или `false`, он будет создан. Таким образом, это ещё один способ определить массив array. Однако такой способ применять не рекомендуется, так как если переменная *`$arr`* уже содержит некоторое значение (например, значение типа string из переменной запроса), то это значение останется на месте и `[]` может на самом деле означать доступ к символу в строке. Лучше инициализировать переменную путём явного присваивания значения.

Замечание: Начиная с PHP 7.1.0, используя в оператор "пустой индекс" на строке, приведёт к фатальной ошибке. Ранее, в этом случае, строка молча преобразовывалась в массив.

Замечание: Начиная с PHP 8.1.0, создание нового массива с помощью значения `false` устарело. Создание нового массива из `null` и неопределённого значения по-прежнему разрешено.

Для изменения определённого значения просто присвойте новое значение элементу, используя его ключ. Если вы хотите удалить пару ключ/значение, вам необходимо использовать функцию unset().

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // В этом месте скрипта это
                // то же самое, что и $arr[13] = 56;

$arr["x"] = 42; // Это добавляет к массиву новый
                // элемент с ключом "x"

unset($arr[5]); // Это удаляет элемент из массива

unset($arr);    // Это удаляет массив полностью
?>
```

Замечание:

Как уже говорилось выше, если ключ не был указан, то будет взят максимальный из существующих целочисленных (int) индексов, и новым ключом будет это максимальное значение (в крайнем случае 0) плюс 1. Если целочисленных (int) индексов ещё нет, то ключом будет `0` (ноль).

Учтите, что максимальное целое значение ключа *не обязательно существует в массиве в данный момент*. Оно могло просто существовать в массиве какое-то время, с тех пор как он был переиндексирован в последний раз. Следующий пример это иллюстрирует:

```
<?php
// Создаём простой массив.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Теперь удаляем каждый элемент, но сам массив оставляем нетронутым:
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Добавляем элемент (обратите внимание, что новым ключом будет 5, вместо 0).
$array[] = 6;
print_r($array);

// Переиндексация:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

Результат выполнения данного примера:

```
Array
(
```

```
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

Полезные функции ¶

Для работы с массивами существует достаточное количество полезных функций. Смотрите раздел функции для работы с массивами.

Замечание:
Функция unset() позволяет удалять ключи массива. Обратите внимание, что массив *НЕ* будет переиндексирован. Если вы действительно хотите поведения в стиле "удалить и сдвинуть", можно переиндексировать массив используя array_values().

```php
<?php
$a = array(1 => 'один', 2 => 'два', 3 => 'три');
unset($a[2]);
/* даст массив, представленный так:
   $a = array(1 => 'один', 3 => 'три');
   а НЕ так:
   $a = array(1 => 'один', 2 => 'три');
*/

$b = array_values($a);
// Теперь $b это array(0 => 'один', 1 => 'три')
?>
```

Управляющая конструкция foreach существует специально для массивов. Она предоставляет возможность легко пройтись по массиву.

Что можно и нельзя делать с массивами ¶

Почему `$foo[bar]` неверно? ¶

Всегда заключайте в кавычки строковый литерал в индексе ассоциативного массива. К примеру, пишите `$foo['bar']`, а не `$foo[bar]`. Но почему? Часто в старых скриптах можно встретить следующий синтаксис:

```php
<?php
$foo[bar] = 'враг';
```

```
echo $foo[bar];
// и т.д.
?>
```

Это неверно, хотя и работает. Причина в том, что этот код содержит неопределённую константу (`bar`), а не строку (`'bar'` - обратите внимание на кавычки). Это работает, потому что PHP автоматически преобразует "*голую строку*" (не заключённую в кавычки строку, которая не соответствует ни одному из известных символов языка) в строку со значением этой "голой строки". Например, если константа с именем `bar` не определена, то PHP заменит bar на строку `'bar'` и использует её.

Внимание

Резервный вариант для обработки неопределённой константы как пустой строки вызывает ошибку уровня `E_NOTICE`. Начиная с PHP 7.2.0 поведение объявлено устаревшим и вызывает ошибку уровня `E_WARNING`. Начиная с PHP 8.0.0, удалено и выбрасывает исключение Error.

Замечание: Это не означает, что нужно *всегда* заключать ключ в кавычки. Нет необходимости заключать в кавычки константы или переменные, поскольку это помешает PHP обрабатывать их.

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', true);
ini_set('html_errors', false);
// Простой массив:
$array = array(1, 2);
$count = count($array);
for ($i = 0; $i < $count; $i++) {
    echo "\nПроверяем $i: \n";
    echo "Плохо: " . $array['$i'] . "\n";
    echo "Хорошо: " . $array[$i] . "\n";
    echo "Плохо: {$array['$i']}\n";
    echo "Хорошо: {$array[$i]}\n";
}
?>
```

Результат выполнения данного примера:

```
Проверяем 0:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Плохо:
Хорошо: 1
Notice: Undefined index:  $i in /path/to/script.html on line 11
Плохо:
Хорошо: 1

Проверяем 1:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Плохо:
Хорошо: 2
Notice: Undefined index:  $i in /path/to/script.html on line 11
Плохо:
Хорошо: 2
```

Дополнительные примеры, демонстрирующие этот факт:

```
<?php
// Показываем все ошибки
```

```php
error_reporting(E_ALL);

$arr = array('fruit' => 'apple', 'veggie' => 'carrot');

// Верно
print $arr['fruit'];  // apple
print $arr['veggie']; // carrot

// Неверно. Это работает, но из-за неопределённой константы с
// именем fruit также вызывает ошибку PHP уровня E_NOTICE
//
// Notice: Use of undefined constant fruit - assumed 'fruit' in...
print $arr[fruit];    // apple

// Давайте определим константу, чтобы продемонстрировать, что
// происходит. Мы присвоим константе с именем fruit значение 'veggie'.
define('fruit', 'veggie');

// Теперь обратите внимание на разницу
print $arr['fruit'];  // apple
print $arr[fruit];    // carrot

// Внутри строки это нормально. Внутри строк константы не
// рассматриваются, так что ошибки E_NOTICE здесь не произойдёт
print "Hello $arr[fruit]";      // Hello apple

// С одним исключением: фигурные скобки вокруг массивов внутри
// строк позволяют константам там находиться
print "Hello {$arr[fruit]}";    // Hello carrot
print "Hello {$arr['fruit']}";  // Hello apple

// Это не будет работать и вызовет ошибку обработки, такую как:
// Parse error: parse error, expecting T_STRING' or T_VARIABLE' or T_NUM_STRI
NG'
// Это, конечно, также действует и с суперглобальными переменными в строках
print "Hello $arr['fruit']";
print "Hello $_GET['foo']";

// Ещё одна возможность - конкатенация
print "Hello " . $arr['fruit']; // Hello apple
?>
```

Если вы переведёте error_reporting в режим отображения ошибок уровня `E_NOTICE` (например, такой как `E_ALL`), вы сразу увидите эти ошибки. По умолчанию error_reporting установлена их не отображать.
Как указано в разделе синтаксис, внутри квадратных скобок ('`[`' и '`]`') должно быть выражение. Это означает, что можно писать вот так:

```php
<?php
echo $arr[somefunc($bar)];
?>
```

Это пример использования возвращаемого функцией значения в качестве индекса массива. PHP известны также и константы:

```php
<?php
$error_descriptions[E_ERROR]   = "Произошла фатальная ошибка";
$error_descriptions[E_WARNING] = "PHP сообщает о предупреждении";
$error_descriptions[E_NOTICE]  = "Это лишь неофициальное замечание";
?>
```

Обратите внимание, что `E_ERROR` - это такой же верный идентификатор, как и `bar` в первом примере. Но последний пример по сути эквивалентен такой записи:

```php
<?php
$error_descriptions[1] = "Произошла фатальная ошибка";
$error_descriptions[2] = "PHP сообщает о предупреждении";
$error_descriptions[8] = "Это лишь неофициальное замечание";
?>
```

поскольку `E_ERROR` соответствует `1` и т.д.

Так что же в этом плохого?

Когда-нибудь в будущем команда разработчиков PHP возможно пожелает добавить ещё одну константу или ключевое слово, либо константа из другого кода может вмешаться и тогда у вас могут возникнуть проблемы. Например, вы уже не можете использовать таким образом слова `empty` и `default`, поскольку они являются зарезервированными ключевыми словами.

Замечание: Повторим, внутри строки (string), заключённой в двойные кавычки, корректно не окружать индексы массива кавычками, поэтому `"$foo[bar]"` является верной записью. Более подробно почему - смотрите вышеприведённые примеры, а также раздел по обработке переменных в строках.

Преобразование в массив ¶

Для любого из типов int, float, string, bool и resource, преобразование значения в массив даёт результатом массив с одним элементом (с индексом 0), являющимся скалярным значением, с которого вы начали. Другими словами, `(array)$scalarValue` - это точно то же самое, что и `array($scalarValue)`.

Если вы преобразуете в массив объект (object), вы получите в качестве элементов массива свойства (переменные-члены) этого объекта. Ключами будут имена переменных-членов, с некоторыми примечательными исключениями: целочисленные свойства станут недоступны; к закрытым полям класса (private) спереди будет дописано имя класса; к защищённым полям класса (protected) спереди будет добавлен символ '*'. Эти добавленные значения с обоих сторон также имеют `NUL` байты.

Неинициализированные типизированные свойства автоматически отбрасываются.

```php
<?php
class A {
    private $B;
    protected $C;
    public $D;
    function __construct()
```

```
    {
        $this->{1} = null;
    }
}
var_export((array) new A());
?>
```

Результат выполнения данного примера:

```
array (
  '' . "\0" . 'A' . "\0" . 'B' => NULL,
  '' . "\0" . '*' . "\0" . 'C' => NULL,
  'D' => NULL,
  1 => NULL,
)
```

Это может вызвать несколько неожиданное поведение:

```
<?php
class A {
    private $A; // Это станет '\0A\0A'
}
class B extends A {
    private $A; // Это станет '\0B\0A'
    public $AA; // Это станет 'AA'
}
var_dump((array) new B());
?>
```

Результат выполнения данного примера:

```
array(3) {
  ["BA"]=>
  NULL
  ["AA"]=>
  NULL
  ["AA"]=>
  NULL
}
```

Вышеприведённый код покажет 2 ключа с именем 'AA', хотя один из них на самом деле имеет имя '\0A\0A'.
Если вы преобразуете в массив значение `null`, вы получите пустой массив.

Сравнение ¶

Массивы можно сравнивать при помощи функции array_diff() и операторов массивов.

Распаковка массива ¶

Массив с префиксом ... будет распакован во время определения массива. Только массивы и объекты, которые реализуют интерфейс Traversable, могут быть распакованы. Распаковка массива с помощью ... доступна, начиная с PHP 7.4.0.
Массив можно распаковывать несколько раз и добавлять обычные элементы до или после оператора ...:
Пример #9 Простая распаковка массива

```php
<?php
// Использование короткого синтаксиса массива.
// Также работает с синтаксисом array().
$arr1 = [1, 2, 3];
$arr2 = [...$arr1]; // [1, 2, 3]
$arr3 = [0, ...$arr1]; // [0, 1, 2, 3]
$arr4 = [...$arr1, ...$arr2, 111]; // [1, 2, 3, 1, 2, 3, 111]
$arr5 = [...$arr1, ...$arr1]; // [1, 2, 3, 1, 2, 3]
function getArr() {
  return ['a', 'b'];
}
$arr6 = [...getArr(), 'c' => 'd']; // ['a', 'b', 'c' => 'd']
?>
```

Распаковка массива с помощью оператора ... следует семантике функции array_merge(). То есть более поздние строковые ключи перезаписывают более ранние, а целочисленные ключи перенумеровываются:
Пример #10 Распаковка массива с дублирующим ключом

```php
<?php
// строковый ключ
$arr1 = ["a" => 1];
$arr2 = ["a" => 2];
$arr3 = ["a" => 0, ...$arr1, ...$arr2];
var_dump($arr3); // ["a" => 2]
// целочисленный ключ
$arr4 = [1, 2, 3];
$arr5 = [4, 5, 6];
$arr6 = [...$arr4, ...$arr5];
var_dump($arr6); // [1, 2, 3, 4, 5, 6]
// Который [0 => 1, 1 => 2, 2 => 3, 3 => 4, 4 => 5, 5 => 6]
// где исходные целочисленные ключи не были сохранены.
?>
```

Замечание:
Ключи, которые не являются ни целыми числами, ни строками, вызывают ошибку TypeError. Такие ключи могут быть сгенерированы только объектом Traversable.
Замечание:
До PHP 8.1 распаковка массива со строковым ключом не поддерживалась:

```php
<?php
$arr1 = [1, 2, 3];
$arr2 = ['a' => 4];
$arr3 = [...$arr1, ...$arr2];
// Ошибка: невозможно распаковать массив со строковыми ключами в example.php:
5
$arr4 = [1, 2, 3];
$arr5 = [4, 5];
$arr6 = [...$arr4, ...$arr5]; // работает. [1, 2, 3, 4, 5]
?>
```

Примеры ¶

Тип массив в PHP является очень гибким, вот несколько примеров:

```php
<?php
// это
$a = array( 'color' => 'red',
            'taste' => 'sweet',
            'shape' => 'round',
            'name'  => 'apple',
            4        // ключом будет 0
          );

$b = array('a', 'b', 'c');

// . . .полностью соответствует
$a = array();
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name']  = 'apple';
$a[]        = 4;        // ключом будет 0

$b = array();
$b[] = 'a';
$b[] = 'b';
$b[] = 'c';

// после выполнения этого кода, $a будет массивом
// array('color' => 'red', 'taste' => 'sweet', 'shape' => 'round',
// 'name' => 'apple', 0 => 4), а $b будет
// array(0 => 'a', 1 => 'b', 2 => 'c'), или просто array('a', 'b', 'c').
?>
```

Пример #11 Использование array()

```php
<?php
// Массив как карта (свойств)
$map = array( 'version'    => 4,
              'OS'         => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );

// исключительно числовые ключи
$array = array( 7,
                8,
                0,
                156,
                -10
              );
// это то же самое, что и array(0 => 7, 1 => 8, ...)

$switching = array(         10, // ключ = 0
                     5    =>  6,
                     3    =>  7,
                     'a'  =>  4,
```

```
                           11, // ключ = 6 (максимальным числовым индексом б
ыло 5)
                    '8'  =>  2, // ключ = 8 (число!)
                    '02' => 77, // ключ = '02'
                    0    => 12  // значение 10 будет перезаписано на 12
                  );

// пустой массив
$empty = array();
?>
```

Пример #12 Коллекция

```
<?php
$colors = array('red', 'blue', 'green', 'yellow');

foreach ($colors as $color) {
    echo "Вам нравится $color?\n";
}

?>
```

Результат выполнения данного примера:

```
Вам нравится red?
Вам нравится blue?
Вам нравится green?
Вам нравится yellow?
```

Изменение значений массива напрямую возможно путём передачи их по ссылке.

Пример #13 Изменение элемента в цикле

```
<?php
foreach ($colors as &$color) {
    $color = strtoupper($color);
}
unset($color); /* это нужно для того, чтобы последующие записи в
$color не меняли последний элемент массива */

print_r($colors);
?>
```

Результат выполнения данного примера:

```
Array
(
    [0] => RED
    [1] => BLUE
    [2] => GREEN
    [3] => YELLOW
)
```

Следующий пример создаёт массив, начинающийся с единицы.

Пример #14 Индекс, начинающийся с единицы

```
<?php
$firstquarter  = array(1 => 'Январь', 'Февраль', 'Март');
print_r($firstquarter);
?>
```

Результат выполнения данного примера:

```
Array
```

```
(
    [1] => 'Январь'
    [2] => 'Февраль'
    [3] => 'Март'
)
```

Пример #15 Заполнение массива

```php
<?php
// заполняем массив всеми элементами из директории
$handle = opendir('.');
while (false !== ($file = readdir($handle))) {
    $files[] = $file;
}
closedir($handle);
?>
```

Массивы упорядочены. Вы можете изменять порядок элементов, используя различные функции сортировки. Для дополнительной информации смотрите раздел функции для работы с массивами. Вы можете подсчитать количество элементов в массиве с помощью функции count().

Пример #16 Сортировка массива

```php
<?php
sort($files);
print_r($files);
?>
```

Поскольку значение массива может быть чем угодно, им также может быть другой массив. Таким образом вы можете создавать рекурсивные и многомерные массивы.

Пример #17 Рекурсивные и многомерные массивы

```php
<?php
$fruits = array ( "fruits"  => array ( "a" => "апельсин",
                                       "b" => "банан",
                                       "c" => "яблоко"
                                     ),
                  "numbers"   => array ( 1,
                                         2,
                                         3,
                                         4,
                                         5,
                                         6
                                       ),
                  "holes"   => array (      "первая",
                                         5 => "вторая",
                                              "третья"
                                       )
                );

// Несколько примеров доступа к значениям предыдущего массива
echo $fruits["holes"][5];    // напечатает "вторая"
echo $fruits["fruits"]["a"]; // напечатает "апельсин"
unset($fruits["holes"][0]);  // удалит "первая"

// Создаст новый многомерный массив
```

```
$juices["apple"]["green"] = "good";
?>
```

Обратите внимание, что при присваивании массива всегда происходит копирование значения. Чтобы скопировать массив по ссылке, вам нужно использовать оператор ссылки.

```
<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // $arr2 изменился,
             // $arr1 всё ещё array(2, 3)

$arr3 = &$arr1;
$arr3[] = 4; // теперь $arr1 и $arr3 одинаковы
?>
```

62.Опишите использование фигурных скобок при форматировании строк в PHP.

Обработка переменных ¶

Если строка указывается в двойных кавычках, либо при помощи heredoc, переменные внутри неё обрабатываются.
Существует два типа синтаксиса: простой и сложный. Простой синтаксис более лёгок и удобен. Он даёт возможность обработки переменной, значения массива (array) или свойства объекта (object) с минимумом усилий.
Сложный синтаксис может быть определён по фигурным скобкам, окружающим выражение.

Простой синтаксис

Если интерпретатор встречает знак доллара ($), он захватывает так много символов, сколько возможно, чтобы сформировать правильное имя переменной. Если вы хотите точно определить конец имени, заключайте имя переменной в фигурные скобки.

```
<?php
$juice = "apple";

echo "He drank some $juice juice.".PHP_EOL;

// Некорректно. 's' - верный символ для имени переменной, но переменная имеет
 имя $juice.
echo "He drank some juice made of $juices.";

// Корректно. Строго указан конец имени переменной с помощью скобок:
echo "He drank some juice made of ${juice}s.";
?>
```

Результат выполнения данного примера:

```
He drank some apple juice.
He drank some juice made of .
He drank some juice made of apples.
```

Аналогично могут быть обработаны элемент массива (array) или свойство объекта (object). В индексах массива закрывающая квадратная скобка (])

обозначает конец определения индекса. Для свойств объекта применяются те же правила, что и для простых переменных.

Пример #15 Пример простого синтаксиса

```
<?php
define('KOOLAID', 'koolaid1');
$juices = array("apple", "orange", "koolaid1" => "purple");

echo "He drank some $juices[0] juice.".PHP_EOL;
echo "He drank some $juices[1] juice.".PHP_EOL;
echo "He drank some $juices[koolaid1] juice.".PHP_EOL;

class people {
    public $john = "John Smith";
    public $jane = "Jane Smith";
    public $robert = "Robert Paulsen";

    public $smith = "Smith";
}

$people = new people();

echo "$people->john drank some $juices[0] juice.".PHP_EOL;
echo "$people->john then said hello to $people->jane.".PHP_EOL;
echo "$people->john's wife greeted $people->robert.".PHP_EOL;
echo "$people->robert greeted the two $people->smiths."; // Не сработает
?>
```

Результат выполнения данного примера:

```
He drank some apple juice.
He drank some orange juice.
He drank some purple juice.
John Smith drank some apple juice.
John Smith then said hello to Jane Smith.
John Smith's wife greeted Robert Paulsen.
Robert Paulsen greeted the two .
```

В PHP 7.1.0 добавлена поддержка *отрицательных* числовых индексов.

Пример #16 Отрицательные числовые индексы

```
<?php
$string = 'string';
echo "Символ с индексом -2 равен $string[-2].", PHP_EOL;
$string[-3] = 'o';
echo "Изменение символа на позиции -3 на 'o' даёт следующую строку: $string."
, PHP_EOL;
?>
```

Результат выполнения данного примера:

```
Символ с индексом -2 равен n.
Изменение символа на позиции -3 на 'o' даёт следующую строку: strong
```

Для чего-либо более сложного, используйте сложный синтаксис.

Сложный (фигурный) синтаксис

Он называется сложным не потому, что труден в понимании, а потому что позволяет использовать сложные выражения.

Любая скалярная переменная, элемент массива или свойство объекта, отображаемое в строку, может быть представлена в строке этим синтаксисом. Выражение записывается так же, как и вне строки, а затем заключается в `{` и `}`. Поскольку `{` не может быть экранирован, этот синтаксис будет распознаваться только когда `$` следует непосредственно за `{`. Используйте `{\$`, чтобы напечатать `{$`. Несколько поясняющих примеров:

```php
<?php
// Показываем все ошибки
error_reporting(E_ALL);

$great = 'здорово';

// Не работает, выводит: Это { здорово}
echo "Это { $great}";

// Работает, выводит: Это здорово
echo "Это {$great}";

// Работает
echo "Этот квадрат шириной {$square->width}00 сантиметров.";

// Работает, ключи, заключённые в кавычки, работают только с синтаксисом фигу
рных скобок
echo "Это работает: {$arr['key']}";

// Работает
echo "Это работает: {$arr[4][3]}";

// Это неверно по той же причине, что и $foo[bar] вне
// строки. Другими словами, это по-прежнему будет работать,
// но поскольку PHP сначала ищет константу foo, это вызовет
// ошибку уровня E_NOTICE (неопределённая константа).
echo "Это неправильно: {$arr[foo][3]}";

// Работает. При использовании многомерных массивов внутри
// строк всегда используйте фигурные скобки
echo "Это работает: {$arr['foo'][3]}";

// Работает.
echo "Это работает: " . $arr['foo'][3];

echo "Это тоже работает: {$obj->values[3]->name}";

echo "Это значение переменной по имени $name: {${$name}}";

echo "Это значение переменной по имени, которое возвращает функция getName():
 {${getName()}}";

echo "Это значение переменной по имени, которое возвращает \$object->getName(
): {${$object->getName()}}";
```

```php
// Не работает, выводит: Это то, что возвращает getName(): {getName()}
echo "Это то, что возвращает getName(): {getName()}";

// Не работает, выводит: C:\folder\{fantastic}.txt
echo "C:\folder\{$great}.txt"

// Работает, выводит: C:\folder\fantastic.txt
echo "C:\\folder\\{$great}.txt"
?>
```

С помощью этого синтаксиса также возможен доступ к свойствам объекта внутри строк.

```php
<?php
class foo {
    var $bar = 'I am bar.';
}

$foo = new foo();
$bar = 'bar';
$baz = array('foo', 'bar', 'baz', 'quux');
echo "{$foo->$bar}\n";
echo "{$foo->{$baz[1]}}\n";
?>
```

Результат выполнения данного примера:

```
I am bar.
I am bar.
```

Замечание:
Значение, к которому осуществляется доступ из функций, вызовов методов, статических переменных класса и констант класса внутри `{$}`, будет интерпретироваться как имя переменной в области, в которой определена строка. Использование одинарных фигурных скобок (`{}`) не будет работать для доступа к значениям функций, методов, констант классов или статических переменных класса.

```php
<?php
// Показываем все ошибки
error_reporting(E_ALL);

class beers {
    const softdrink = 'rootbeer';
    public static $ale = 'ipa';
}

$rootbeer = 'A & W';
$ipa = 'Alexander Keith\'s';

// Это работает, выводит: Я бы хотел A & W
echo "Я бы хотел {${beers::softdrink}}\n";

// Это тоже работает, выводит: Я бы хотел Alexander Keith's
echo "Я бы хотел {${beers::$ale}}\n";
?>
```

Доступ к символу в строке и его изменение ¶

Символы в строках можно использовать и модифицировать, определив их смещение относительно начала строки, начиная с нуля, в квадратных скобках после строки, например, `$str[42]`. Думайте о строке для этой цели, как о массиве символов. Если нужно получить или заменить более 1 символа, можно использовать функции substr() и substr_replace().

Замечание: Начиная с PHP 7.1.0, поддерживаются отрицательные значения смещения. Они задают смещение с конца строки. Ранее отрицательные смещение вызывали ошибку уровня `E_NOTICE` при чтении (возвращая пустую строку) либо `E_WARNING` при записи (оставляя строку без изменений).

Замечание: До PHP 8.0.0 для доступа к символу в строке (string) также можно было использовать фигурные скобки, например, `$str{42}`, для той же цели. Синтаксис фигурных скобок устарел в PHP 7.4.0 и больше не поддерживается в PHP 8.0.0.

Внимание

Попытка записи в смещение за границами строки дополнит строку пробелами до этого смещения. Нецелые типы будет преобразованы в целые. Неверный тип смещения вызовет ошибку уровня `E_WARNING`. Используется только первый символ присваемой строки. Начиная с PHP 7.1.0, присвоение пустой строки вызовет фатальную ошибку. Ранее в таком случае присваивался нулевой байт (NULL).

Внимание

Строки в PHP внутренне представляют из себя массивы байт. Как результат, доступ или изменение строки по смещению небезопасно с точки зрения многобайтной кодировки, и должно выполняться только со строками в однобайтных кодировках, таких как, например, ISO-8859-1.

Замечание: Начиная с PHP 7.1.0, использование пустого индекса вызывает фатальную ошибку, ранее в подобном случае строка преобразовывалась в массив без предупреждения.

Пример #17 Несколько примеров строк

```php
<?php
// Получение первого символа строки
$str = 'This is a test.';
$first = $str[0];

// Получение третьего символа строки
$third = $str[2];

// Получение последнего символа строки
$str = 'This is still a test.';
$last = $str[strlen($str)-1];

// Изменение последнего символа строки
$str = 'Look at the sea';
$str[strlen($str)-1] = 'e';

?>
```

Смещение в строке должно задаваться либо целым числом, либо строкой, содержащей цифры, иначе будет выдаваться предупреждение.

63.Опишите особенности приоритетности операторов в PHP. Что такое ассоциативность операторов.

Приоритет оператора ¶

Приоритет оператора определяет, насколько "тесно" он связывает между собой два выражения. Например, выражение `1 + 5 * 3` вычисляется как `16`, а не `18`, поскольку оператор умножения ("*") имеет более высокий приоритет, чем оператор сложения ("+"). Круглые скобки могут использоваться для принудительного указания порядка выполнения операторов. Например, выражение `(1 + 5) * 3` вычисляется как `18`.

Если операторы имеют равный приоритет, то будут ли они выполняться справа налево или слева направо определяется их ассоциативностью. К примеру, "-" является лево-ассоциативным оператором. Следовательно `1 - 2 - 3` сгруппируется как `(1 - 2) - 3` и пересчитается в `-4`. С другой стороны "=" - право-ассоциативный оператор, так что `$a = $b = $c` сгруппируется как `$a = ($b = $c)`.

Неассоциативные операторы с одинаковым приоритетом не могут использоваться совместно. К примеру `1 < 2 > 1` не будет работать в PHP. Выражение `1 <= 1 == 1`, с другой стороны, будет, поскольку `==` имеет более низкий приоритет чем `<=`.

Ассоциативность имеет смысл только для двоичных (и тернарных) операторов. Унарные операторы являются префиксными или постфиксными, поэтому это понятие не применимо. Например, `!!$a` можно сгруппировать только как `!(!$a)`.

Использование скобок, кроме случаев когда они строго необходимы, может улучшить читаемость кода, группируя явно, а не полагаясь на приоритеты и ассоциативность.

В следующей таблице приведён список операторов, отсортированный по убыванию их приоритетов. Операторы, размещённые в одной строке имеют одинаковый приоритет и порядок их выполнения определяется исходя из их ассоциативности.

Порядок выполнения операторов

Ассоциативность	Оператор	Дополнительная информация
(н/а)	clone new	clone и new
правая	**	арифметические операторы
(н/а)	+ - ++ -- ~ (int) (float) (string) (array) (object) (bool) @	арифметические операторы (унарные + и -), инкремент/декремент, побитовые операторы, приведение типов и оператор управления ошибками
левая	instanceof	типы
(н/а)	!	логические операторы
левая	* / %	арифметические операторы
левая	+ - .	арифметические операторы (бинарные + и -), операторы, работающие с массивами и строковые операторы (. до PHP 8.0.0)
левая	<< >>	побитовые операторы
левая	.	строковые операторы (начиная с PHP 8.0.0)
неассоциативна	< <= > >=	операторы сравнения
неассоциативна	== != === !== <> <=>	операторы сравнения
левая	&	побитовые операторы и ссылки
левая	^	побитовые операторы
левая	|	побитовые операторы
левая	&&	логические операторы
левая	||	логические операторы
правая	??	операторы сравнения с null
неассоциативна	? :	тернарный оператор (лево-ассоциативный до PHP 8.0.0)
правая	= += -= *= **= /= .= %= &= |= ^= <<= >>= ??=	операторы присваивания
(н/а)	yield from	yield from
(н/а)	yield	yield
(н/а)	print	print
левая	and	логические операторы
левая	xor	логические операторы
левая	or	логические операторы

Пример #1 Ассоциативность

```php
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
// ассоциативность тернарных операторов отличается от C/C++
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2 (до PHP 8.0.0)

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```

Приоритет и ассоциативность оператора определяет только то, как группируется выражение, а не порядок его вычисления. Обычно PHP не указывает, в каком порядке вычисляются выражения и кода, который предполагает специфичный порядок вычисления следует избегать, потому, что поведение может меняться в разных версиях PHP или в зависимости от окружающего кода.

Пример #2 Неопределённый порядок вычисления

```php
<?php
$a = 1;
```

```php
echo $a + $a++; // может вывести как 2 так и 3

$i = 1;
$array[$i] = $i++; // может установить индекс как 1, так 2
?>
```

Пример #3 +, - и . имеют одинаковый приоритет (до PHP 8.0.0)

```php
<?php
$x = 4;
// следующий код может выдать неожиданный результат:
echo "x минус 1 равно " . $x-1 . ", ну я надеюсь\n";
// поскольку он вычисляется таким образом (до PHP 8.0.0):
echo (("x минус один равно " . $x) - 1) . ", ну я надеюсь\n";
// требуемый приоритет следует задать скобками:
echo "x минус 1 равно " . ($x-1) . ", ну я надеюсь\n";
?>
```

Результат выполнения данного примера:

```
-1, ну я надеюсь
-1, ну я надеюсь
x минус один равно 3, ну я надеюсь
```

Замечание:
Несмотря на то, что = имеет более низкий приоритет, чем большинство других операторов, PHP всё же позволяет делать так: `if (!$a = foo())`, в этом примере результат выполнения `foo()` будет присвоен *$a*.

64.Опишите особенности арифметических операторов в PHP. Как работает автоматическое преобразование из строки в число и когда?

Арифметические операторы ¶

Помните школьные основы арифметики? Описанные ниже операторы работают так же.

Арифметические операции		
Пример	**Название**	**Результат**
+$a	Идентичность	Конвертация *$a* в int или float, что более подходит.
-$a	Отрицание	Смена знака *$a*.
$a + $b	Сложение	Сумма *$a* и *$b*.
$a - $b	Вычитание	Разность *$a* и *$b*.
$a * $b	Умножение	Произведение *$a* и *$b*.
$a / $b	Деление	Частное от деления *$a* на *$b*.
$a % $b	Деление по модулю	Целочисленный остаток от деления *$a* на *$b*.
$a ** $b	Возведение в степень	Возведение *$a* в степень *$b*.

Операция деления ("/") возвращает число с плавающей точкой, кроме случая, когда оба значения являются целыми числами (или строками, которые преобразуются в целые числа), которые делятся нацело - в этом случае возвращается целое значение. Для целочисленного деления используйте intdiv().

При делении по модулю операнды преобразуются в целые числа (int) (путём удаления дробной части) до начала операции. Для деления по модулю чисел с плавающей точкой используйте fmod().
Результат операции остатка от деления `%` будет иметь тот же знак, что и делимое — то есть, результат `$a % $b` будет иметь тот же знак, что и `$a`.
Например:

```php
<?php

echo (5 % 3)."\n";          // выводит 2
echo (5 % -3)."\n";         // выводит 2
echo (-5 % 3)."\n";         // выводит -2
echo (-5 % -3)."\n";        // выводит -2

?>
```

Преобразование типов переменных **в PHP** происходит в зависимости от контекста. Когда использовали **арифметический оператор**, предназначенный для работы с **числами**, строковую переменную **PHP автоматически** преобразовал **в число**. При использовании **оператора** конкатенации **строк число** было **автоматически** преобразовано в **строку**.

65.Что такое веб-сервер?

Веб-сервер — это сервер, принимающий HTTP-запросы от клиентов, обычно веб браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер.

66.Что такое сервер приложения и чем он отличается от веб-сервера?

Основное различие между веб-сервером и сервером приложений заключается в том, что веб-сервер предназначен для обслуживания статических страниц, например HTML и CSS, тогда как сервер приложений отвечает за генерацию динамического содержимого путем выполнения кода на стороне сервера, например, JSP, сервлета или EJB.

67.Кратко опишите историю развития интернета в рамках развития веб-серверов.

Первый сервер ARPANET был установлен 1 сентября 1969 года в Калифорнийском университете в Лос-Анджелесе. Компьютер Honeywell DP-516 имел 24 Кб оперативной памяти.

29 октября 1969 года в 21:00 между двумя первыми узлами сети ARPANET, находящимися на расстоянии в 640 км — в Калифорнийском университете Лос-Анджелеса (UCLA) и в Стэнфордском исследовательском институте (SRI) — провели сеанс связи. Чарли Клайн (Charley Kline) пытался выполнить удалённое подключение к компьютеру в SRI. Успешную передачу каждого введённого символа его коллега Билл Дювалль (Bill Duvall) из SRI подтверждал по телефону.

В первый раз удалось отправить всего три символа «LOG», после чего сеть перестала функционировать. LOG должно было быть словом LOGON (команда входа в систему). В рабочее состояние систему вернули уже к 22:30 и следующая попытка оказалась успешной. Именно эту дату можно считать днём рождения Интернета. К 1971 году была разработана первая программа для отправки электронной почты по сети. Эта программа сразу стала очень популярна. В 1973 году к сети были подключены через трансатлантический телефонный кабель первые иностранные организации из Великобритании и Норвегии, сеть стала международной. В 1970-х годах

сеть в основном использовалась для пересылки электронной почты, тогда же появились первые списки почтовой рассылки, новостные группы и доски объявлений. Однако в то время сеть ещё не могла легко взаимодействовать с другими сетями, построенными на других технических стандартах. К концу 1970-х годов начали бурно развиваться протоколы передачи данных, которые были стандартизированы в 1982—83 годах. Активную роль в разработке и стандартизации сетевых протоколов играл Джон Постел. 1 января 1983 года сеть ARPANET перешла с протокола NCP на TCP/IP, который успешно применяется до сих пор для объединения (или, как ещё говорят, «наслоения») сетей. Именно в 1983 году термин «Интернет» закрепился за сетью ARPANET. В 1984 году была разработана система доменных имён (англ. Domain Name System, DNS).
В 1984 году у сети ARPANET появился серьёзный соперник: Национальный научный фонд США (NSF) основал обширную межуниверситетскую сеть NSFNet (англ. National Science Foundation Network), которая была составлена из более мелких сетей (включая известные тогда сети Usenet и Bitnet) и имела гораздо бóльшую пропускную способность, чем ARPANET. К этой сети за год подключились около 10 тыс. компьютеров, звание «Интернет» начало плавно переходить к NSFNet. В 1988 году был разработан протокол Internet Relay Chat (IRC), благодаря чему в Интернете стало возможно общение в реальном времени (чат). В 1989 году в Европе, в стенах Европейского совета по ядерным исследованиям (фр. Conseil Européen pour la Recherche Nucléaire, CERN) родилась концепция Всемирной паутины. Её предложил знаменитый британский учёный Тим Бернерс-Ли, он же в течение двух лет разработал протокол HTTP, язык HTML и идентификаторы URI. В 1990 году сеть ARPANET прекратила своё существование, полностью проиграв

конкуренцию NSFNet. В том же году было зафиксировано первое подключение к Интернету по телефонной линии (т. н. «дозво́н» — англ. Dialup access). В 1991 году Всемирная паутина стала общедоступна в Интернете, а в 1993 году появился знаменитый веб-браузер NCSA Mosaic. Всемирная паутина набирала популярность. Можно считать что существует две ясно различимые эры в истории Web: [до браузера Mosaic] Марка Андриссена и после. Именно сочетание веб-протокола от Тима Бернерс-Ли, который обеспечивал коммуникацию, и браузера (Mosaic) от Марка Андриссена, который предоставил функционально совершенный пользовательский интерфейс, создало условия для наблюдаемого взрыва (интереса к Веб). За первые 24 месяца, истекшие после появления браузера Mosaic, Web прошел стадию от полной неизвестности (за пределами считанного числа людей внутри узкой группы ученых и специалистов лишь одного мало кому известного профиля деятельности) до полной и абсолютно везде в мире его распространенности. A Brief History of Cyberspace, Mark Pesce, ZDNet, 15 октября 1995В 1995 году NSFNet вернулась к роли исследовательской сети, маршрутизацией всего трафика Интернета теперь занимались сетевые провайдеры, а не суперкомпьютеры Национального научного фонда.

В том же 1995 году Всемирная паутина стала основным поставщиком информации в Интернете, обогнав по трафику протокол пересылки файлов FTP. Был образован Консорциум всемирной паутины (W3C). Можно сказать, что Всемирная паутина преобразила Интернет и создала его современный облик. С 1996 года Всемирная паутина почти полностью подменяет собой понятие «Интернет». В 1990-е годы Интернет объединил в себе большинство существовавших тогда сетей (хотя некоторые, как Фидонет, остались обособленными). Объединение выглядело

привлекательным благодаря отсутствию единого руководства, а также благодаря открытости технических стандартов Интернета, что делало сети независимыми от бизнеса и конкретных компаний. К 1997 году в Интернете насчитывалось уже около 10 млн компьютеров, было зарегистрировано более 1 млн доменных имён. Интернет стал очень популярным средством для обмена информацией. В настоящее время подключиться к Интернету можно через спутники связи, радио-каналы, кабельное телевидение, телефон, сотовую связь, специальные оптико-волоконные линии или электропровода. Всемирная сеть стала неотъемлемой частью жизни в развитых и развивающихся странах.
В течение пяти лет Интернет достиг аудитории свыше 50 миллионов пользователей.

68.Кратко опишите протокол HTTP.

HTTP — это протокол, позволяющий получать различные ресурсы, например HTML-документы. Протокол HTTP лежит в основе обмена данными в Интернете. HTTP является протоколом клиент-серверного взаимодействия, что означает инициирование запросов к серверу самим получателем, обычно веб-браузером (web-browser).

69.Опишите механизм взаимодействия HTTP-сервера, HTTP-клиента и пользователя.

Когда пользователь хочет перейти на страницу, браузер отправляет HTTP-запрос GET с указанием URL-адреса его HTML-страницы. Сервер извлекает запрошенный документ из своей файловой системы и возвращает HTTP-ответ, содержащий документ и код состояния HTTP Response status code 200 OK (успех). Сервер может вернуть другой код состояния, например, «404 Not Found», если файл отсутствует на сервере или «301 Moved Permanently», если файл существует, но был перемещён в другое место.

70.Опишите цели и задачи веб-сервера.

Цель веб-сервера проста - обслуживать одновременно большое количество клиентов, максимально эффективно используя hardware. Главная задача веб сервера принимать HTTP-запросы от пользователей, обрабатывать их, переводить в цифровой компьютерный код. Затем выдавать HTTP-ответы, преобразуя их из миллионов нолей и единичек в изображения, медиа-потоки, буквы, HTML страницы.Любой веб сервер, для удобства его использования пользователями, должен иметь удобный веб-браузер. Он передает веб серверу запросы, преобразованные в URL-адреса интернет - ресурсов.Наряду со стандартными функциями, некоторые веб серверы имеют дополнительные. Так, к примеру, соответствующее программное обеспечение может фиксировать число обращений пользователей к тому или иному ресурсу, записывать их в отдельный журнал. А еще они могут поддерживать HTTPS, что не маловажно для защищенного соединения между сайтами и пользователями. Зачастую веб-сервер устанавливается вместе с мейл-сервером. Это позволяет пользователям быстро переходить на страничку почты прямо с сайта, нажав всего лишь на одну гиперссылку.

71.Опишите технологию SSI.

SSI – это простая и удобная технология организации динамических страничек. SSI экономит место на сервере, и одновременно делает администрирование сайта удобней в десятки раз.

72.Что такое система управления контентом?

Система управления контентом (CMS) — это программное обеспечение, которое работает в вашем браузере. Она позволяет создавать, управлять и изменять веб-сайт и его содержимое, не имея никаких знаний в области программирования. Система управления контентом предоставляет вам

графический интерфейс пользователя. В нём вы можете управлять всеми аспектами вашего сайта.

73.Верно ли, что сервер приложения умеет работать с протоколом HTTP?
Верно.

HTTP-клиентами чаще всего являются браузеры — Google Chrome, Mozilla Firefox, Safari, Opera, Yandex Browser и другие. А серверами являются веб-сервера. Вот эта приставка «веб-» и указывает на то, что это не просто какой-то сервер, а сервер, который умеет принимать запросы и отвечать на них по протоколу HTTP.

74.Что такое CGI?
CGI (от англ. Common Gateway Interface — «общий интерфейс шлюза») — стандарт интерфейса, используемого для связи внешней программы с веб-сервером. Программу, которая работает по такому интерфейсу совместно с веб-сервером, принято называть шлюзом

75.Как работает система с использованием интерфейс шлюза - CGI?
Сам интерфейс разработан таким образом, чтобы можно было использовать любой язык программирования, который может работать со стандартными устройствами ввода-вывода. Такими возможностями обладают даже скрипты для встроенных командных интерпретаторов операционных систем, поэтому в простых случаях могут использоваться даже командные скрипты.

76.Назовите достоинства и недостатки CGI.

- CGI не налагает особых условий на платформу и web - сервер, поэтому работает на всех популярных платформах и web - серверах. Также технология не привязана к конкретному языку программирования и может быть использована на любом языке, работающем со стандартными потоками ввода/вывода.

- Производительность CGI - программ не высока. Основной причиной этого является то, что при очередном обращении к серверу для работы CGI – программы создается отдельный процесс, что требует большого количества системных ресурсов.
- Встроенных средств масштабируемости технология не предусматривает.
- CGI - программа представляет из себя готовый к исполнению файл, что препятствует легкому расширению системы.

77.Что такое FastCGI?

Интерфейс FastCGI — клиент-серверный протокол взаимодействия веб-сервера и приложения, дальнейшее развитие технологии CGI. По сравнению с CGI является более производительным и безопасным.

78.Назовите основные отличия CGI от FastCGI.

CGI-программа, запущенная в цикле. Если обычная CGI-программа заново запускается для каждого нового запроса, то в FastCGI-программе используется очередь запросов, которые обрабатываются последовательно.

79.Что такое менеджер процессов?

Поток process_manager отвечает за генерацию новых экземпляров процесса, за связь со средой и между экземплярами процесса, а также за выполнение экземпляров процесса.

80.Что такое PHP-FPM?

PHP-FPM — это альтернативная реализация PHP FastCGI с несколькими дополнительными возможностями, которые обычно используются для высоконагруженных сайтов.

81.Что такое Spawn-fcgi?

spawn-fcgi — одна из составных частей проекта Lighttpd. Предназначен он для того, что бы запустить php, как FastCGI сервер, ну а с этим сервером может работать потом практически любой http сервер.

82.Что такое Lighttpd?

lighttpd это веб-сервер с открытым исходным кодом, оптимизированный для критически важных сред, отвечает за предоставление доступа через HTTP или HTTPS протокол к статическому контенту.

83.Что такое chroot окружение?

Chroot окружение — способ запуска программ, системный вызов и просто команда, позволяющая изменить корневой каталог в системе.

Chroot используется для создания jail и изоляции процессов, а также для сборки пакетов методом debootstrap. Командой chroot с указанием каталога запускается механизм, создающий систему директорий в этом каталоге идентичную той, что существует в корне. Команда выполняется только от имени суперпользователя.

84.Опишите механизм взаимодействия серверов с использованием FastCGI.

FastCGI — открытый унифицированный стандарт, расширяющий интерфейс CGI и позволяющий создавать высокопроизводительные web-приложения без использования специфичных API web-сервера. Цель данной спецификации — с точки зрения FastCGI-приложения описать интерфейс взаимодействия между ним и web-сервером, также реализующим интерфейс FastCGI. Связь между web-сервером и FastCGI-процессом осуществляется через один сокет, который процесс должен слушать на предмет входящих подключений от web-сервера. После приема соединения от web-сервера FastCGI-процесс обменивается данными с использованием простого протокола, решающего две задачи: организация двунаправленного обмена в рамках одного соединения (для эмуляции STDIN, STDOUT, STDERR) и организация нескольких независимых FastCGI-сессий в рамках одного соединения.

85.Опишите процесс выбора встроенного или внешнего менеджера процессов.

Менеджер процессов тесно взаимодействует с микроядром, чтобы обеспечить услуги, составляющие сущность операционной системы. Менеджер процессов отвечает за создание новых процессов в системе и за

управление основными ресурсами, связанными с процессом. Все эти услуги предоставляются посредством сообщений.
Встроенный менеджер процессов – удобная программа, с помощью которой можно определять запущенные на ПК приложения, процессы и службы, управлять вышеперечисленными компонентами (запускать, останавливать или завершать). С помощью нее можно оценивать и влиять на быстродействие компьютера, а также выполнять другие задачи.
К сожалению, встроенный менеджер процессов для анализа и обработки запущенных процессов не всегда удобен и имеет очень ограниченные возможности, он не в полной мере отображает процессы, запущенные на компьютере, а средства анализа запущенных процессов вообще весьма ограничены. Из-за недостатков в применении менеджера процессов для анализа процессов, запущенных на компьютере, приходится прибегать к внешним менеджерам процессов. Они более качественно отображают информацию и позволяют лучше контролировать процессы.

86.Что такое интерфейс шлюза?

Сетевой шлюз - аппаратный маршрутизатор или программное обеспечение для сопряжения компьютерных сетей, использующих разные протоколы (например, локальной и глобальной). Сетевой шлюз конвертирует протоколы одного типа физической среды в протоколы другой физической среды (сети).

87.Что такое SCGI?

SCGI (Simple Common Gateway Interface) - простой общий интерфейс шлюза - разработан как альтернатива CGI и во многом аналогичен FastCGI, но более прост в реализации.

88.Что такое PCGI

PCGI (Perl Common Gateway Interface) — библиотека к языку программирования Perl для работы с интерфейсом CGI. Библиотека позволяет с высокой скоростью обрабатывать входящий поток данных. Основное достоинство заключается в том, что библиотека позволяет совершенно безопасно принимать сколь угодно крупные объёмы данных, при этом очень экономично потребляя оперативную память.

89.Что такое PSGI?

PSGI (Perl Web Server Gateway Interface) – это спецификация, предназначенная для отделения среды веб-сервера от кода веб-фреймворка. PSGI не является программным интерфейсом (API) для веб-приложений.

90.Что такое WSGI?

WSGI — стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером, например Apache. [WSGI предоставляет простой и универсальный интерфейс между большинством веб-серверов и веб-приложениями или фреймворками]. Это протокол, спецификация, которая была предложена в PEP 3333. Этот протокол предназначен для решения проблем совместимости многих веб-платформ и программного обеспечения веб-сервера. Благодаря WSGI вам больше не нужно выбирать конкретное программное обеспечение веб-сервера из-за используемой веб-платформы.

91.Опишите механизм взаимодействия серверов Apache и PHP.

Когда компьютер обращается к web-серверу Apache, то он запускает интерпретатор PHP. Он выполняет скрипт записанный в файле index.php. То есть Apache это сервер, который взаимодействует с клиентом (принимает и отвечает на запросы клиента), а затем сервер выполняет запрос клиента, используя PHP

92.Опишите преимущества веб-сервера Apache.

Основные достоинства Apache - надежность, безопасность и гибкость настройки. Apache позволяет подключать различные модули, добавляющие в него новые возможности - например, можно подключить модуль, обеспечивающий поддержку PHP или любого другого Web-ориентированного языка программирования.

93.Опишите недостатки веб-сервера Apache.

Недостатки - отсутствие удобного графического интерфейса администратора. Настройка Apache осуществляется путем редактирования его конфигурационного файла. В Интернете можно найти простые конфигураторы Apache, но их возможностей явно не хватает для настройки всех функций Web-сервера.

94.Опишите архитектуру веб-сервера Apache.

Apache состоит из ядра и динамической модульной системы. Параметры системы изменяются с помощью конфигурационных файлов.

(В архитектуру Apache входит: простое ядро, платформо-зависимый уровень (APR), и модули.)

95.Опишите функции ядра веб-сервера Apache.

Ядро Apache включает в себя основные функциональные возможности, такие как обработка конфигурационных файлов, протокол HTTP и система загрузки модулей. Ядро (в отличие от модулей) полностью разрабатывается Apache Software Foundation, без участия сторонних программистов.

Теоретически ядро apache может функционировать в чистом виде, без использования модулей. Однако функциональность такого решения крайне ограничена.

Ядро Apache полностью написано на языке программирования C

96.Опишите конфигурацию веб-сервера Apache.

Система конфигурации Apache основана на текстовых конфигурационных файлах. Имеет три условных уровня конфигурации:

- Конфигурация сервера
- Конфигурация виртуального хоста
- Конфигурация уровня каталога

97.Что такое URI, URL и чем они различаются.

URL — это URI, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса. A URN — это URI, который только идентифицирует ресурс в определённом пространстве имён (и, соответственно, в определённом контексте), но не указывает его местонахождение.

(URI: Это лишь обобщенное понятие (множество) идентификации ресурса, включающее в нашем случае как URL, так и URN, как по отдельности, так и совместно. Т.е. мы можем считать, что: URI = URL или URI = URN или URI = URL + URN)

\\\ВОПРОСЫ ИЗ 4 ПРАКТИКИ///

98.Что такое HTTP-запрос?

HTTP запросы - это сообщения, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера.

99.Опишите существующие HTTP-запросы.

Особенности GET запроса:

-может быть закэширован

-остается в истории браузера

-может быть закладкой в браузере

-не должен использоваться при работе с крайне -важными данными

-имеет ограниченную длину

-должен применяться только для получения данных (ред.)

Особенности POST запроса:

-не кэшируется

-не может быть закладкой в браузере

-не остаётся в истории браузера

-нет ограничений по длине запроса

100. Опишите обработку запроса на PHP. Что нужно использовать, как вычленить параметры запроса?

Внутри PHP-скрипта имеется несколько способов получения доступа к данным, переданным клиентом по протоколу HTTP. До версии PHP 4.1.0 доступ к таким данным осуществлялся по именам переданных переменных (напомним, что данные передаются в виде пар "имя переменной, символ "=", значение переменной"). Таким образом, если, например, было передано first_name=Nina, то внутри скрипта появлялась переменная $first_name со значением Nina. Если требовалось различать, каким методом были переданы данные, то использовались ассоциативные массивы $HTTP_POST_VARS и $HTTP_GET_VARS , ключами которых являлись имена переданных переменных, а значениями – соответственно значения этих переменных. Таким образом, если пара first_name = Nina передана методом GET , то $HTTP_GET_VARS["first_name"]="Nina".

Использовать в программе имена переданных переменных напрямую небезопасно. Поэтому было решено начиная с PHP 4.1.0 задействовать для обращения к переменным, переданным с помощью HTTP-запросов, специальный массив – $_REQUEST . Этот массив содержит данные, переданные методами POST и GET , а также с помощью HTTP cookies. Это суперглобальный ассоциативный массив, т.е. его значения можно получить в любом месте программы, используя в качестве ключа имя соответствующей переменной (элемента формы).

101. Опишите создание HTML-форм на PHP.

Вставка формы осуществляется напрямую в HTML—код страницы. Главный элемент формы называется <form>. Уже внутрь него добавляются все остальные элементы – текстовые поля, «чекбоксы», переключатели и т.д. У элемента <form> имеется несколько атрибутов, один из которых

является обязательным. Он называется action. В action указывается, где именно будет приниматься и обрабатываться информация, переданная посредством формы. Как правило, обработка происходит в стороннем PHP—файле. Пример использования атрибута– action=»obrabotchik.php«. Атрибут method позволяет задать метод передачи информации. По умолчанию (если не прописывать атрибут) будет указан метод GET. В данном случае информация передается напрямую через URL—адрес. Для каждого элемента формы будет создана пара следующего вида – «имя элемента = значение, которое в нем лежит». Все эти пары, разделенные знаком «амперсанд» будут перечислены в адресной строке. Если прописать method=»POST» (регистр не важен), то данные будут передаваться не через URL, а через тело запроса (в скрытом режиме)

102. Что такое API?

API — описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой. Обычно входит в описание какого-либо интернет-протокола, программного каркаса или стандарта вызовов функций операционной системы.

103. Опишите API как средство интеграции приложений.

Если программу (модуль, библиотеку) рассматривать как чёрный ящик, то API — это набор «ручек», которые доступны пользователю данного ящика и которые он может вертеть и дёргать.

Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию — высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов. По такому принципу построены протоколы передачи данных по Интернету. Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью предыдущего

(«нижележащего») уровня передачи данных и, в свою очередь, предоставляет нужную функциональность следующему («вышележащему») уровню.

Понятие протокола близко по смыслу к понятию API. И то, и другое является абстракцией функциональности, только в первом случае речь идёт о передаче данных, а во втором — о взаимодействии приложений. API библиотеки функций и классов включает в себя описание сигнатур и семантики функций

104. Что такое Web API?

Это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения, и поэтому обычно не включает детали реализации веб-сервера или браузера, такие как SAPI или API, если они не доступны для общего доступа через удаленное веб-приложение.

105. Приведите пример API.

Каждый раз, когда пользователь посещает какую-либо страницу в сети, он взаимодействует с API удалённого сервера. API — это составляющая часть сервера, которая получает запросы и отправляет ответы.

106. Что такое REST?

Это архитектура, т.е. принципы построения распределенных гипермедиа систем, того что другими словами называется World Wide Web, включая универсальные способы обработки и передачи состояний ресурсов по HTTP.

107. Как организована передача данных в архитектуре REST?

Архитектура REST требует соблюдения следующего условия. В период между запросами серверу не нужно хранить информацию о состоянии клиента и наоборот. Все запросы от клиента должны быть составлены так,

чтобы сервер получил всю необходимую информацию для выполнения запроса. Таким образом и сервер, и клиент могут «понимать» любое принятое сообщение, не опираясь при этом на предыдущие сообщения

108. Как организована работа REST?

Это набор принципов и ограничений взаимодействия клиента и сервера в сети интернет, использующий существующие стандарты (HTTP протокол, стандарт построения URL, форматы данных JSON и XML) в ходе взаимодействии

109. Что такое SOAP?

SOAP (Simple Object Access Protocol) — стандартный протокол по версии W3C.

110. Чем SOAP отличается от REST?

· SOAP обозначает простой протокол доступа к объектам, тогда как REST обозначает передачу представительного состояния.

· SOAP — это протокол, тогда как REST — это архитектурный паттерн.

· SOAP использует сервисные интерфейсы для предоставления своих функций клиентским приложениям, а REST использует унифицированные локаторы сервисов для доступа к компонентам на аппаратном устройстве.

· SOAP требует большей пропускной способности для его использования, тогда как REST не требует большой пропускной способности.

· SOAP работает только с форматами XML, тогда как REST работает с простым текстом, XML, HTML и JSON.

· SOAP не может использовать REST, тогда как REST может использовать SOAP.

111. Для чего нужен SOAP-процессор?

SOAP основан на языке XML и расширяет некоторый протокол прикладного уровня — HTTP, FTP, SMTP и т.д. Как правило чаще всего используется HTTP. Вместо использования HTTP для запроса HTML-страницы, которая будет показана в браузере, SOAP отправляет

посредством HTTP-запроса XML-сообщение и получает результат в HTTP-отклике. Для правильной обработки XML-сообщения процесс-«слушатель» HTTP (напр. Apache или Microsoft IIS) должен предоставить SOAP-процессор, или, другими словами, должен иметь возможность обрабатывать XML

112. Опишите общую структуру SOAP-сообщения.

Сообщение SOAP состоит из заголовка — элемент SOAP-ENV:Header и основной части — элемент SOAP-ENV:Body. Заголовок может содержать метаданные, относящиеся к сообщению в целом. В теле сообщения передается элемент params с входными параметрами метода. Формат элемента params отличается для разных методов

113. Что такое и что содержит Конверт (SOAP Envelope)?

Является самым «верхним» элементом SOAP сообщения. Содержит корневой элемент

XML-документа. Описывается с помощью элемента Envelope с обязательным

пространством имен http://www.w3.org/2003/05/soap-envelope для версии 1.2 и

http://schemas.xmlsoap.org/soap/ для версии 1.1.

У элемента Envelope могут быть атрибуты xmlns, определяющие пространства

имен, и другие атрибуты, снабженные префиксами.

Envelope может иметь необязательный дочерний элемент Header с тем же

пространством имен — заголовок. Если этот элемент присутствует, то он должен быть

Рисунок 2. Структура

SOAP сообщения

первым прямым дочерним элементом конверта.

Следующий дочерний элемент конверта должен иметь имя Body и то же самое

пространство имен - тело. Это обязательный элемент и он должен быть вторым прямым

дочерним элементом конверта, если есть заголовок, или первым — если заголовка нет.

Версия 1.1 позволяла после тела сообщения записывать произвольные элементы,

снабженные префиксами. Версия 1.2 это запрещает.

Элементы Header и Body могут содержать элементы из различных пространств

имен.

Конверт изменяется от версии к версии. SOAP-процессоры, совместимые с версией

1.1, при получении сообщения, содержащего конверт с пространством имен версии 1.2,

будут генерировать сообщение об ошибке. Аналогично для SOAP-процессоров, совместимых
с версией 1.2. Ошибка — VersionMismatch.

114. Что такое и что содержит Заголовок SOAP (SOAP Header)?

Заголовок может содержать метаданные, относящиеся к сообщению в целом.Заголовок содержит элемент locale, устанавливающий русский язык для ответных сообщений, и элемент token — авторизационный токен.

115. Что такое и что содержит Тело SOAP (SOAP Body)?

Тело SOAP-сообщения является обязательным элементом внутри env:Envelope, содержащим основную информацию SOAP-сообщения, которая должна быть передана из начальной точки пути сообщения в конечную

116. Опишите SOAP-сообщение с вложением.

SOAP-сообщение представляет собой XML-документ; сообщение состоит из трех

основных элементов: конверт (SOAP Envelope), заголовок (SOAP Header) и тело (SOAP Body)

117. Что такое graphql?

Язык запросов и обработки данных с открытым исходным кодом для API и среда выполнения для выполнения запросов с существующими данными.

118. Что такое Распознаватели (resolvers) в graphql?

Resolver или распознаватель — функция, которая возвращает данные для определённого поля. Resolver’ы возвращают данные того типа, который определён в схеме. Распознаватели могут быть асинхронными.

119. Из чего состоит экосистема graphql, что нужно, чтобы использовать данную технологию?

Она состоит из двух взаимосвязанных объектов: TypeDefs и Resolvers. Выше были описаны основные типы GraphQL. Чтобы сервер мог с ними работать, эти типы необходимо определить. Объект typeDef определяет список типов, которые доступны в проекте

120. Что такое валидация данных и для чего она нужна?

Валидация – это проверка продукта, процесса или системы на соответствие требованиям клиента

121. Где и когда выполнять валидацию данных?

Валидация проводится тогда, когда невозможно оценить соответствие продукта, процесса или системы требованиям клиента до того, как клиент начнет этим продуктом пользоваться. Например, если речь идет о программном обеспечении, в него встраивается валидационный код. Этот код клиент вводит, если продукт полностью соответствует его ожиданиям и выполняет нужные задачи. В противном случае доступ к продукту прекращается и проводятся его доработки либо исполнитель возвращает деньги.

122. Как выполнять валидацию данных?

Перед отправкой данных на сервер важно убедиться, что все обязательные поля формы заполнены данными в корректном формате. Это называется валидацией на стороне клиента и помогает убедиться, что данные, введённые в каждый элемент формы, соответствуют требованиям.

123. Приведите пример с поэтапной валидацией данных.

Регистрация и авторизация: пользователь вводит валидные данные email и валидные данные пароль

124. Что такое запрос и мутация в graphql и чем они отличаются?

К первому виду относятся запросы на чтение данных, которые в терминологии GraphQL называются просто запросами (query) и относятся к букве R (reading, чтение) акронима CRUD. Запросы второго вида — это запросы на изменение данных, которые в GraphQL называют мутациями (mutation).

\\\ВОПРОСЫ ИЗ 5 ПРАКТИКИ///

125. Что такое сессия в рамках веб-разработки?

Сессия — это диалоговое состояние между клиентом и сервером, включающее информацию о предыдущих запросах клиента и ответах сервера. В силу того, что протокол и сами веб-сервера не обеспечивают сохранение состояния, то единственной возможностью для этого является передача всей необходимой информации о запросе в нём самом.

126. Что такое cookie в рамках веб-разработки?

Куки, или cookies, — это служебные файлы, которые создаёт браузер для получения необходимой информации о пользователе.

127. Опишите механизм использования cookies.

Cookie используются веб-серверами для идентификации пользователей и хранения данных о них. К примеру, если вход на сайт осуществляется при помощи cookie, то, после ввода пользователем своих данных на странице входа, cookie позволяют серверу запомнить, что пользователь уже идентифицирован и ему разрешён доступ к соответствующим услугам и операциям

128. Опишите простой пример работы сессий в PHP.

Пример будет состоять из трёх файлов: index.php, authorize.php и secretplace.php. Файл index.php содержит форму, где пользователь введёт свой логин и пароль. Эта форма передаст данные файлу authorize.php, который в случае успешной авторизации допустит пользователя к файлу secretplace.php, а в противном случае выдаст сообщение об ошибке

129. Опишите способы защиты сессии пользователя.

Использование cookie, шифрования, проверка браузера, срок действия сессии, привязка по IP-адресу

130. Верно ли, что можно хранить данные сессии в БД?

Сессия как правило - это необходимые горячие данные пользователя, что нужно сохранить между запросами. БД - одно из самых медленных хранилищ этих данных. В файлах кстати тоже не стоит хранить. При большой нагрузке io будет подтормаживать. Длительный период сессии обычно не хранятся, вместо этого на клиент задается токен, по которому человек через много времени может автоматически авторизироваться

131. Опишите жизненный цикл сессии.

Когда вы запускаете или продолжаете сеанс с помощью session_start()$_SESSION Когда выполнение скрипта заканчивается, данные сохраняются обратно в файл. Поэтому, когда вы устанавливаете переменную сеанса, она не сохраняется сразу. Конечно, вы можете заставить сеанс сохранять данные, вызывая session_write_close() session_set_save_handler() Требуется шесть аргументов, каждый из которых является обратным вызовом, который обрабатывает определенную стадию жизненного цикла сеанса. Они есть:

Открытие файла сеанса. Закрытие файла сеанса. Чтение данных сеанса. Запись данных сеанса. Уничтожение сессии. Сборка мусора из файла сессии и данных. Вы должны зарегистрировать функцию для каждого этапа жизненного цикла, иначе PHP выдаст предупреждение о том, что не может найти функцию

132. Верно ли, что можно убрать механизм обработки сессий?

Скорее всего да, но тогда это может повлиять на безопасность сайта. Не будет возможности отслеживать действия пользователей, анализировать их для дальнейших действий, в случае подозрительного поведения. Также очень легко будет подвергнуться атакам со стороны взломщиков и тд.

133. Опишите примеры настройки сессии во время выполнения.

session.save_handler stringsession.save_handler определяет имя обработчика, который используется для хранения и извлечения данных, связанных с сессией. По умолчанию имеет значение files. Следует обратить внимание, что некоторые модули могут зарегистрировать собственные обработчики (save_handler). Текущие зарегистрированные обработчики отображаются в phpinfo().

134. Опишите директивы конфигурации файловой системы и потоков в PHP.

allow_url_fopen bool. Данная директива включает поддержку обёрток URL (URL wrappers), которые позволяют работать с объектами URL как с обычными файлами. Обёртки, доступные по умолчанию, служат для работы с удалёнными файлами с использованием ftp или http протокола. Некоторые модули, например, zlib, могут регистрировать собственные обёртки.

135. Какой тип ресурса использует файловая система. Опишите данный тип.

Например, стандарт NTFS разработан с целью устранения недостатков, присущих более ранним версиям ФС. Впервые он был реализован в Windows NT в 1995 году, и в настоящее время является основной файловой системой для Windows. Система NTFS расширила допустимый предел размера файлов до шестнадцати гигабайт, поддерживает разделы диска до 16 Эб (эксабайт, 1018 байт). Использование системы шифрования Encryption File System (метод «прозрачного шифрования») осуществляет разграничение доступа к данным для различных пользователей, предотвращает несанкционированный доступ к

содержимому файла. Файловая система позволяет использовать расширенные имена файлов, включая поддержку многоязычности в стандарте юникода UTF, в том числе в формате кириллицы. Встроенное приложение проверки жесткого диска или внешнего накопителя на ошибки файловой системы chkdsk повышает надежность работы харда, но отрицательно влияет на производительность.

136. Как открыть и закрыть файл с помощью PHP.

Функция fopen() в PHP — это встроенная функция, которая используется для открытия файла или URL-адреса. В случае сбоя, она возвращает FALSE и выдаёт ошибку. Если вам нужно скрыть вывод ошибки добавьте символ '@' перед именем функции. Функция fclose() используется, чтобы закрыть открытый файл.

137. Как производится чтение и запись файлов в PHP.

Для простого отображения всего содержимого файла идеально подходит функция readfile (). Для записи текстовых данных в файл существует две идентичные функции: int fputs (int file, string string [, int length]), int fwrite (int file, string string [, int length]).

138. Опишите как считать только часть файла, как считывать файл последовательно и считать весь файл целиком.

string fgets (int file, int length) – функция считывает строку длиною length. Функция fgets() используется для чтения одной строки из файла. Функция fgetc() используется для чтения одного символа из файла. ункция feof() полезна для цикла работающего с данным неизвестной длины

139. Как производится создание и удаление файлов с помощью PHP.

Чтобы создать файл php, можно использовать функцию fopen() в режиме доступа «w» или «w+». Или функцию touch (). Она устанавливает время изменения файла. При отсутствии элемента с искомым именем он будет создан. Удалить файл можно с помощью функции unlink().

140. С помощью каких функций и какую информацию о файле можно получить с помощью PHP?

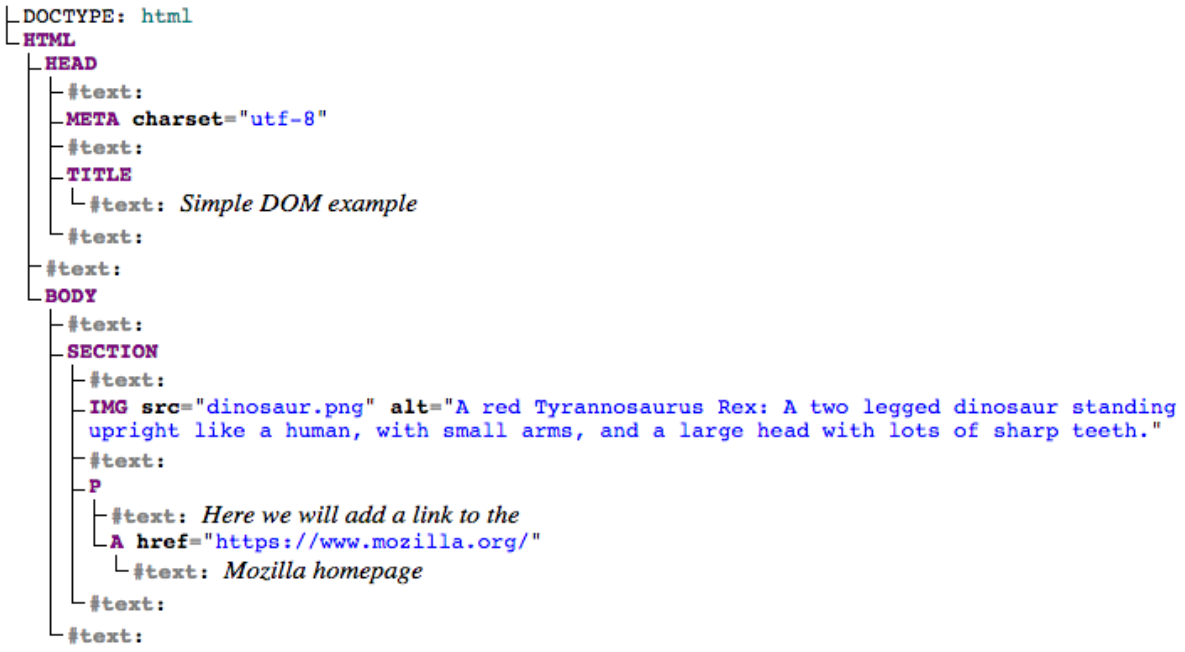
Для получения информации о файлах в php используется целый ряд функций:

bool fileexists (string filename) – проверяет, существует ли элемент;
int fileatime (string filename) – возвращает время последнего открытия;
int filesize (string filename) – возвращает байтовый размер файла;
string filetype (string filename) – тип файла

141. Что такое DOM?

DOM означает объектную модель документа. Это программный интерфейс, который позволяет нам создавать, изменять или удалять элементы из документа. Мы также можем добавлять события к этим элементам, чтобы сделать нашу страницу более динамичной. Модель DOM рассматривает документ HTML как дерево узлов.

142. Как создать документ и работать с ним с помощью модуля DOM?



143. Что такое JSON?

JSON — это формат, который хранит структурированную информацию и в основном используется для передачи данных между сервером и клиентом. Файл JSON представляет собой более простую и лёгкую альтернативу расширению с аналогичными функциями XML (Extensive Markup Language).

144. Как декодировать строку JSON и вернуть JSON-представление данных?

Структуры JSON-данных очень похожи на массивы PHP. PHP имеет встроенные функции для кодирования и декодирования данных JSON. Это функции json_encode() и json_decode() соответственно. Обе функции работают только со строковыми данными в кодировке UTF-8.

145. Как проанализировать и выявить ошибки при кодировании и декодировании JSON?

Для этого можно воспользоваться сервисами валидации json файлов, которые могут проанализировать и указать на наличие ошибок в json файлах.

146. Опишите создание, сохранение, парсинг XML-документа я помощью PHP.

В php xml файлы создаются с помощью DOM функций, которые создают дерево объектов, в точности повторяющее дерево элементов XML документа. Так как DOM строит дерево всего документа, то и ресурсов он кушает немало (памяти и процессора он потребляет достаточно). Для создания XML документа используется класс – DomDocument.

Пример парсинга:

```
<?xml version='1.0'?> <worker> <name>Коля</name> <age>25</age> <salary>1000</salary> </worker>
```

Пример PHP:

```
$xml = simplexml_load_file(путь к файлу или урл);
echo $xml->name; //выведет 'Коля'
echo $xml->age; //выведет 25
echo $xml->salary; //выведет 1000
```

DOMDocument::saveXML — Сохраняет XML-дерево из внутреннего представления в виде строки

147. Что такое драйвер в рамках взаимодействия с СУБД?

Драйвера это библиотеки, позволяющие осуществлять взаимодействие между программой и СУБД/БД.

148. Опишите добавление записи в рамках использования модуля mysqli для взаимодействия с БД MYSQL.

```
<?php
1. if (isset($_POST["username"]) && isset($_POST["userage"])) {
2.
3.     $conn = new mysqli("localhost", "root", "mypassword", "testdb2");
4.     if($conn->connect_error){
5.         die("Ошибка: " . $conn->connect_error);
6.     }
7.     $name = $conn->real_escape_string($_POST["username"]);
8.     $age = $conn->real_escape_string($_POST["userage"]);
9.     $sql = "INSERT INTO Users (name, age) VALUES ('$name', $age)";
10.    if($conn->query($sql)){
11.        echo "Данные успешно добавлены";
12.    } else{
13.        echo "Ошибка: " . $conn->error;
14.    }
15.    $conn->close();
16.}
17.?>
```

149. Что такое постоянное соединение, опишите проблемы данного подхода и его решение в модуле mysqli.

Идея постоянных подключений состоит в том, чтобы соединение между клиентским процессом и базой данных можно было использовать повторно, особенно когда требуется создавать и закрывать соединения множество раз. Это бы позволило снизить накладные расходы на создание новых подключений каждый раз, когда они требуются, за счёт использования существующих кешированных подключений, свободных для повторного использования. При использовании постоянных соединений можно столкнуться с проблемой, которая заключается в том, что клиенты могут оставлять такие подключения в непредсказуемом

состоянии. Например, клиент ставит блокировку на таблицу, а затем аварийно завершает работу. То есть блокировка снята не будет. Новый клиентский процесс, использующий это подключение повторно, получит его как есть, и вынужден будет провести какую-то очистку подключения, прежде чем начать его использовать. Соответственно, в задачи программиста входит ещё и проверка подобных ситуаций и внедрение кода, осуществляющего такую очистку. Тем не менее, в mysqli эта проблема решена. В модуле есть встроенный функционал, осуществляющий очистку соединений и переводящий их в состояние пригодное для использования. Код очистки, реализованный в mysqli включает следующие операции:

Откат активных транзакций

Закрытие и удаление временных таблиц

Снятие блокировки с таблиц

Сброс переменных сессии

Закрытие подготовленных запросов (всегда происходит в PHP)

Закрытие обработчиков

Снятие блокировок, установленных функцией GET_LOCK()

Это позволяет быть уверенным в том, что возвращённые из пула соединения готовы к использованию в клиентских процессах. Модуль mysqli делает очистку соединений автоматически путём вызова C-API функции mysql_change_user().

150. Опишите основные особенности БД MongoDB.

Главные особенности MongoDB:

Это кроссплатформенная документоориентированная база данных NoSQL с открытым исходным кодом.

Она не требует описания схемы таблиц, как в реляционных БД. Данные хранятся в виде коллекций и документов.

Между коллекциями нет сложных соединений типа JOIN, как между таблицами реляционных БД. Обычно соединение производится при сохранении данных путем объединения документов.

Данные хранятся в формате BSON (бинарные JSON-подобные документы).

У коллекций не обязательно должна быть схожая структура. У одного документа может быть один набор полей, в то время как у другого документа — совершенно другой (как тип, так и количество полей).

В одном документе могут быть поля разных типов данных, данные не нужно приводить к одному типу. Основное преимущество MongoDB заключается в том, что она может хранить любые данные, но эти данные должны быть в формате JSON

151. Опишите процесс добавления новой записи в СУБД MongoDB с помощью соответствующего драйвера.

Для добавления в коллекцию могут использоваться три ее метода:

insertOne(): добавляет один документ

insertMany(): добавляет несколько документов

insert(): может добавлять как один, так и несколько документов

Пример добавления:

db.users.insertOne({"name": "Tom", "age": 28, languages: ["english", "spanish"]});

152. Опишите процесс получения и обработки записей с помощью драйвера MongoDB.

Для получения всех записей в коллекции используйте метод find() экземпляра коллекции. Обновление записи происходит с помощью метода updateOne(), который принимает условие для записей, которые необходимо обновить, и объект со свойством $set, в значении которого указываются, какие именно поля нужно изменить. За удаление записей отвечают методы deleteOne() и deleteMany().

153. Опишите получение записей в рамках использования модуля mysqli для взаимодействия с БД MYSQL.

Для получения данных в MySQL применяется команда SELECT. При выполнении команды SELECT метод query() объекта mysqli возвращает набор полученных строк, который мы можем перебрать с помощью цикла

154. Опишите поиск записей, подсчет и ограничение выборки с помощью драйвера MongoDB.

Функция find() позволяет возвратить несколько документов. Еще одна функция findOne() работает похожим образом, только возвращает один документ. Например:

```
db.users.find({name: "Tom", age: 32})
```

Для подсчета выборки можно использовать следующее:

```
db.collection.count(query, options)
```

Для того, чтобы извлечь лишь одно значение из полученной выборки можно воспользоваться методом findOne(). Однако, к методу findOne() прибегают чаще в тех ситуациях, когда ожидается, что результирующая коллекция будет содержать лишь один документ. В тех же случаях, когда следует ограничить выборку

несколькими документами предпочитают использовать метод limit(), который принимает в качестве аргумента количество извлекаемых документов

\\\ВОПРОСЫ ИЗ 6 ПРАКТИКИ///

155. Что такое Composer и для чего он используется?

Composer - это менеджер для подключения и управления сторонними библиотеками или пакетам в PHP. Его называют пакетный менеджер.
Зачем же в таком случае нужен composer?

1. Много рутинных операций при установке. Нужно производить настройки, подстраивать автозагрузку компонентов и.т.д.

2. Трудность в обновлении библиотек на новые версии. Библиотеки имеют такое свойство расширяться и обновляться. Если вы вручную будете обновлять каждый пакет в вашем проекте, это работа довольно трудная и большая. В composer для обновления пакетов достаточно выполнить одну команду и все пакеты успешно обновляться.

3. Одна библиотека может требовать для своей работы другую, другая третью и.т.д. В итоге, может получаться ряд зависимостей, которые вы должны подключать. Бывают такие библиотеки, которые для своей работы могут требовать десяток и даже сотни других библиотек. Если это все скачивать и подключать вручную на это может уйти месяцы работы. В случае с composer вы просто выполняете команду установки какой-либо библиотеки и он уже автоматически подключит все необходимые для этого другие библиотеки. Пожалуй, это самое важное преимущество почему стоит использовать composer в своей работе.

4. Трудность переноса проекта на рабочий сервер из-за большого объема библиотек. В вашем проекте код самого проекта может занимать всего десятки килобайт, а библиотеки, которые подключены к этому проекту могут занимать сотни и 10-ки сотен мегабайт. Для того, чтобы перенести изменения на рабочем сервере, вам каждый раз нужно будет передавать такой большой объем данных.
Решение этой проблемы с composer довольно простое. В программе composer есть файл настроек, который вы устанавливаете на вашем домашнем компьютере и файл настроек, который вы устанавливаете на удаленном компьютер

156. Опишите работу с календарем в PHP: опишите преобразование дат из одного календаря в другой.

jdtogregorian — Переводит число дней в юлианском летоисчислении в дату по Григорианскому календарю

```
jdtogregorian(int $julian_day): string
```

Переводит число дней в юлианском летоисчислении в строку, содержащую григорианскую дату в формате "месяц/день/год".

cal_from_jd — Преобразует дату, заданную в юлианском календаре, в дату указанного календаря

```
cal_from_jd(int $julian_day, int $calendar): array
```

`cal_from_jd()` преобразует дату юлианского календаря, заданную в julian_day, в дату указанного календаря calendar. Поддерживаемые значения `calendar`: `CAL_GREGORIAN`, `CAL_JULIAN`, `CAL_JEWISH` и `CAL_FRENCH`.

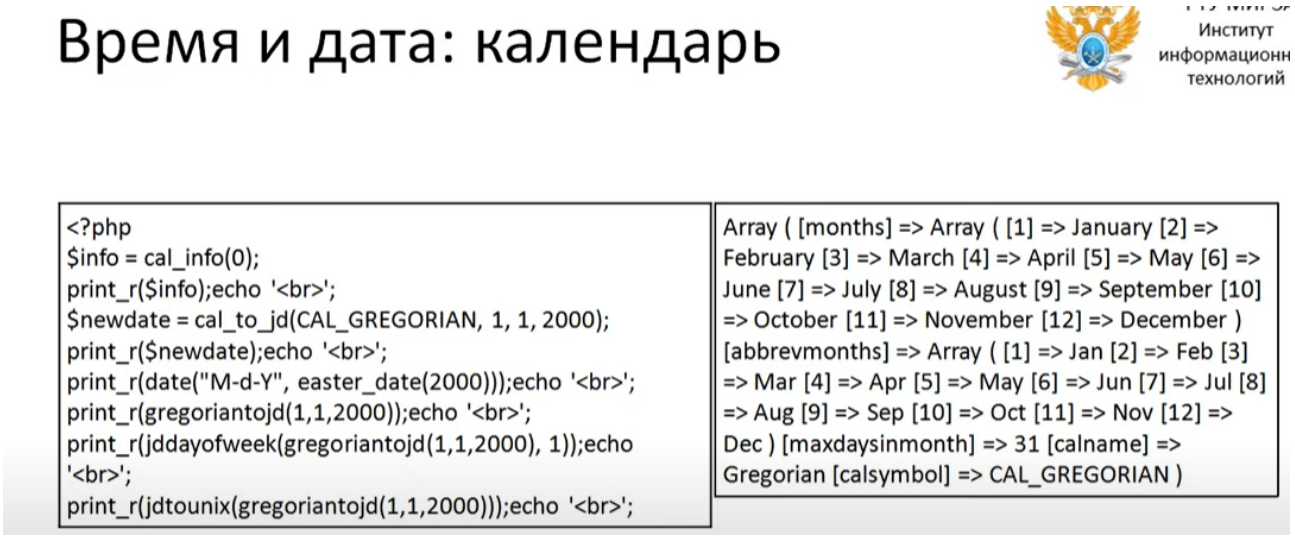


Рисунок 4 - Пример из лекции о дате и времени

157. Опишите понятие серверное время и процесс форматирования времени для разных часовых поясов.

Чтобы получить время сервера, используйте функции time() или date() (есть и другие методы, но их должно быть достаточно). Однако time() может вызывать медленно, как и другие функции, которые получают текущее время (поскольку для получения времени требуются системные вызовы). Вы также должны знать, что он будет меняться от вызова к вызову, если выполнение сценария между вызовами занимает более 1 секунды. Это может вызвать проблемы с согласованностью. Обходной путь заключается в использовании $_SERVER['REQUEST_TIME'], если это возможно, и устранит ошибки постоянства и проблемы со скоростью, которые могут быть вызваны повторным вызовом time(), date() и т. Д., Которые активно получают текущее системное время.

- checkdate — *Проверяет правильность даты по грегорианскому календарю*
- date(gmdate) — *Форматирует системную дату/время (по Гринвичу)*
- getdate — *Возвращает информацию о дате/времени*
- gettimeofday — *Возвращает текущее время*
- localtime — Возвращает системное время
- mktime(gmmktime) — Возвращает метку времени для заданной даты (по Гринвичу)
- strftime(gmstrftime) — *Форматирует дату/время с учетом текущей локали (по Гринвичу)*

- strtotime — *Преобразует текстовое представление даты на английском языке в метку времени Unix*

158. Опишите способ получения времени заката и рассвета с использованием языка программирования PHP.

```
date_sun_info ( int $time , float $latitude , float $longitude )
```

159. Опишите константы, используемые в модуле “Время и дата”.

Предопределённые константы ¶

Константы DATE_ предлагают стандартные представления даты, которые могут быть использованы вместе с функциями форматирования даты (например date()).
Следующие константы определяют формат возвращаемый функциями date_sunrise() и date_sunset().

SUNFUNCS_RET_TIMESTAMP (int)
Время в секундах с начала эпохи Unix

SUNFUNCS_RET_STRING (int)
Часы:минуты (например: 08:02)

SUNFUNCS_RET_DOUBLE (int)
Часы как число с плавающей точкой (например: 8.75)

160. Приведите принципы арифметики даты и времени.

Пример #1 DateTimeImmutable::add/sub добавляет интервалы, охватывающие прошедшее время

Добавление PT24H через переход DST приведёт к добавлению 23/25 часов (для большинства часовых поясов).

```
<?php
$dt = new DateTimeImmutable("2015-11-01 00:00:00", new
DateTimeZone("America/New_York"));
echo "Начало: ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
$dt = $dt->add(new DateInterval("PT3H"));
echo "Конец:  ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
?>
```

Результат выполнения данного примера:
Начало: 2015-11-01 00:00:00 -04:00
Конец: 2015-11-01 02:00:00 -05:00

Пример #2 DateTimeImmutable::modify и strtotime увеличит или уменьшит значения индивидуальных компонентов
Добавление +24 часов через переход DST добавит точно 24 часов (вместо учёта перехода на зимнее или летнее время).

```
<?php
$dt = new DateTimeImmutable("2015-11-01 00:00:00", new
DateTimeZone("America/New_York"));
echo "Начало: ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
$dt = $dt->modify("+24 hours");
echo "Конец:  ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
?>
```

Результат выполнения данного примера:

Начало: 2015-11-01 00:00:00 -04:00

Конец: 2015-11-02 00:00:00 -05:00

Пример #3 Добавление или вычитание времени может уменьшить или увеличить дату

Например, 31 января + 1 месяц вернёт 2 марта (високосный год) или 3 марта (обычный год).

```
<?php
echo "Обычный год:\n"; // В феврале 28 дней
$dt = new DateTimeImmutable("2015-01-31 00:00:00", new
DateTimeZone("America/New_York"));
echo "Начало: ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
$dt = $dt->modify("+1 month");
echo "Конец:  ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;

echo "Високосный год:\n"; // В феврале 29 дней
$dt = new DateTimeImmutable("2016-01-31 00:00:00", new
DateTimeZone("America/New_York"));
echo "Начало: ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
$dt = $dt->modify("+1 month");
echo "Конец:  ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
?>
```

Результат выполнения данного примера:

Обычный год:

Начало: 2015-01-31 00:00:00 -05:00

Конец: 2015-03-03 00:00:00 -05:00

Високосный год:

Начало: 2016-01-31 00:00:00 -05:00

Конец: 2016-03-02 00:00:00 -05:00

Для получения последнего дня следующего месяца (то есть чтобы предотвратить переполнение) существует директива last day of.

```
<?php
echo "Обычный год:\n"; // Февраль содержит 28 дней
$dt = new DateTimeImmutable("2015-01-31 00:00:00", new
DateTimeZone("America/New_York"));
echo "Начало: ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
```

```
$dt = $dt->modify("last day of next month");
echo "Конец:  ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;

echo "Високосный год:\n"; // Февраль содержит 29 дней
$dt = new DateTimeImmutable("2016-01-31 00:00:00", new
DateTimeZone("America/New_York"));
echo "Начало: ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
$dt = $dt->modify("last day of next month");
echo "Конец:  ", $dt->format("Y-m-d H:i:s P"), PHP_EOL;
?>
```

Результат выполнения данного примера:

Обычный год:
Начало: 2015-01-31 00:00:00 -05:00
Конец: 2015-02-28 00:00:00 -05:00
Високосный год:
Начало: 2016-01-31 00:00:00 -05:00
Конец: 2016-02-29 00:00:00 -05:00

161. Чем отличается фреймворк от библиотеки? Приведите пример.

«**Фреймворк**» **отличается** от понятия **библиотеки** тем, что **библиотека** может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений.

Yii - достаточно "возрастной" фреймворк, который продолжает обновляться в наши дни. Отличается удобным функционалом - кэширование, высокая производительность, полная обработка ошибок, возможность переноса (миграции) существующих баз данных, использование jQuery и другое. Фреймворк Yii отличается своей простотой, можно быстро освоить его основы, нет никаких сложностей в работе и использовании основного функционала. Начните изучать Yii2 фрейморк на нашем курсе. Данный PHP-фреймворк часто советуют людям, которые делают первые шаги в понимании PHP-программирования.

Официальная страница yiiframework.com

CodeIgniter - еще один "возрастной" фреймворк, появившийся в начале 2006 года. Именно тогда состоялся его публичный релиз. Среди основных преимуществ этого фреймворка:

- Хорошая документация;
- Небольшой вес и быстрая установка;
- Простота использования.

Многие используют CodeIgniter в качестве базы для обучения - его простота действительно считается наиболее значимым преимуществом. Постоянно появляются новые версии, каждая из которых отличается большим количеством нововведений, исправленных багов.

Официальная страница codeigniter.com

Symfony - невероятно стабильный, мощный фреймворк, который специалисты рекомендуют применять для созданиях крупных проектов. Значительный функционал, гибкость в настройках - популярность этого фреймворка обусловлена его преимуществами. Присутствует огромное количество полезных, многоразовых компонентов, которые можно использовать для создания большого сайта. Сюда можно отнести шаблоны, настройки форм, безопасность.

Официальная страница symfony.com

Laravel - частый лидер разнообразных опросов и рейтингов, посвященных php-фреймворкам. Проект является действительно многообещающим, получил признание достаточно опытных специалистов. Фреймворк просто освоить, является идеальным вариантом для небольших, а также средних по сложности проектов. Подойдет для быстрого, удобного написания требуемого кода.

Официальная страница laravel.com

Phalcon PHP - отличается открытым кодом (языки программирования C, C++, PHP), поддержкой практически всех современных ОС. Производительность этого фреймворка находится на высоком уровне - это подтверждено множеством специализированных тестирований, и, как следствие, его популярностью. Есть возможность использования на собственном сервере.

Официальная страница phalconphp.com

162. Опишите возможные форматы даты и времени и примеры их использования.

Форматы даты

На этой странице описаны форматы даты, которые понимает парсер функций strtotime(), DateTime и date_create().

Используемые символы

Описание	Формат	Примеры
daysuf (суффикс порядкового числительного дня месяца)	"st" | "nd" | "rd" | "th"	
dd (день месяца без ведущих нулей)	([0-2]?[0-9] | "3"[01]) daysuf?	"7th", "22nd", "31"
DD (день месяца, 2 цифры с ведущим нулём)	"0" [0-9] | [1-2][0-9] | "3" [01]	"07", "31"
m (полное или сокращённое название месяца)	'january' | 'february' | 'march' | 'april' | 'may' | 'june' | 'july' | 'august' | 'september' | 'october' | 'november' | 'december' | 'jan' | 'feb' | 'mar' | 'apr' | 'may' | 'jun' | 'jul' | 'aug' | 'sep' | 'sept' | 'oct' | 'nov' | 'dec' | "I" | "II" | "III" | "IV" | "V" | "VI" | "VII" | "VIII" | "IX" | "X" | "XI" | "XII"	
M (сокращённое название месяца)	'jan' | 'feb' | 'mar' | 'apr' | 'may' | 'jun' | 'jul' | 'aug' | 'sep' | 'sept' | 'oct' | 'nov' | 'dec'	
mm (порядковый номер месяца)	"0"? [0-9] | "1"[0-2]	"0", "04", "7", "12"
MM (порядковый номер месяца, 2 цифры с ведущим нулём)	"0" [0-9] | "1"[0-2]	"00", "04", "07", "12"
y (порядковый номер года)	[0-9]{1,4}	"00", "78", "08", "8", "2008"
yy (порядковый номер года, 2 цифры)	[0-9]{2}	"00", "08", "78"
YY (порядковый номер года, 4 цифры)	[0-9]{4}	"2000", "2008", "1978"

Рисунок - Форматы даты php

Региональные нотации		
Описание	**Формат**	**Примеры**
Месяц и день в американской нотации	mm "/" dd	"5/12", "10/27"
Месяц, день и год в американской нотации	mm "/" dd "/" y	"12/22/78", "1/17/2006", "1/17/6"
4 цифры года, месяц и день со слешем-разделителем	YY "/" mm "/" dd	"2008/6/30", "1978/12/22"
4 цифры года и месяц (GNU)	YY "-" mm	"2008-6", "2008-06", "1978-12"
Год, месяц и день с дефисом-разделителем	y "-" mm "-" dd	"2008-6-30", "78-12-22", "8-6-21"
День, месяц и 4 цифры года с разделителем в виде точки, символа табуляции или дефиса	dd [.\t-] mm [.-] YY	"30-6-2008", "22.12.1978"
День, месяц и 2 цифры года с разделителем в виде точки или символа табуляции	dd [.\t] mm "." yy	"30.6.08", "22.12.78"
День, название месяца и год	dd ([\t.-])* m ([\t.-])* y	"30-June 2008", "22DEC78", "14 III 1879"
Название месяца и 4 цифры года (день месяца сбрасывается на 1)	m ([\t.-])* YY	"June 2008", "DEC1978", "March 1879"
4 цифры года и название месяца (день месяца сбрасывается на 1)	YY ([\t.-])* m	"2008 June", "1978-XII", "1879.MArCH"
Название месяца, день и год	m ([.\t-])* dd [,.stndrh\t]+ y	"July 1st, 2008", "April 17, 1790", "May.9,78"
Название месяца и день	m ([.\t-])* dd [,.stndrh\t]*	"July 1st,", "Apr 17", "May.9"
День и название месяца	d ([.\t-])* m	"1 July", "17 Apr", "9.May"
Сокращённое название месяца, день и год	M "-" DD "-" y	"May-09-78", "Apr-17-1790"
Год, сокращённое название месяца и день	y "-" M "-" DD	"78-Dec-22", "1814-MAY-17"
Только год	YY	"1978", "2008"
Только название месяца	m	"March", "jun", "DEC"

Нотации ISO8601		
Описание	**Формат**	**Примеры**
8 цифр (год, месяц и день)	YY MM DD	"15810726", "19780417", "18140517"
8 цифр (год, месяц и день) со слешем-разделителем	YY "/" MM "/" DD	"2008/06/30", "1978/12/22"
2 цифры года, месяц и день с дефисом-разделителем	yy "-" MM "-" DD	"08-06-30", "78-12-22"
4 цифры года с необязательным знаком, месяц и день	[+-]? YY "-" MM "-" DD	"-0002-07-26", "+1978-04-17", "1814-05-17"

Рисунок - Форматы даты и времени php

date(...)

- d - номер дня в месяце. Если меньше 10, то с нулём: "09", "05"
- m - номер месяца. Если меньше 10, то с нулём: "09", "05"
- Y - год, 4 цифры.
- y - год, две цифры.
- n - номер месяца. Без первого нуля, если меньше 10
- j - номер дня в месяце. Без первого нуля, если меньше 10
- H - часы в 24-часовом формате. Если меньше 10, то с нулём: "09", "05"
- s - секунды. Если меньше 10, то с нулём: "09", "05"
- i - минуты. Если меньше 10, то с нулём: "09", "05"
- z - номер дня от начала года.

- w - день недели (0 - воскресенье, 1 - понедельник и т.д.).
- h - часы в 12-часовом формате
- L - 1, если високосный год, 0, если не високосный.
- W - порядковый номер недели года.
- U - количество секунд, прошедших с 1 января 1970 года (то есть timestamp).

163. Опишите работу с датой и временем в подходе ООП.

PHP-расширение Date / Time представляет собой набор классов, которые позволяют вам работать практически со всеми задачами, связанными с датой и временем. Он был доступен с момента выпуска PHP 5.2, и расширение представило несколько новых классов, все из которых сопоставлены с реальными сценариями:

- Дата или время представлены объектом DateTime .
- Часовой пояс мира представлен объектом DateTimeZone .
- Объекты DateInterval представляют интервал. Например, когда мы говорим «два дня», «два дня» — это интервал. Объект DateInterval не зависит от конкретной даты или времени.

Объекты DatePeriod представляют период между двумя датами

164. Опишите использование класса DateTime.

Листинг - Примеры методов DateTime

```
/* Методы */
public __construct(string $datetime = "now", ?DateTimeZone $timezone =
null)
public add(DateInterval $interval): DateTime
public static createFromFormat(string $format, string $datetime,
?DateTimeZone $timezone = null): DateTime|false
public static createFromImmutable(DateTimeImmutable $object): DateTime
public static createFromInterface(DateTimeInterface $object): DateTime
public static getLastErrors(): array|false
public modify(string $modifier): DateTime|false
public static __set_state(array $array): DateTime
public setDate(int $year, int $month, int $day): DateTime
public setISODate(int $year, int $week, int $dayOfWeek = 1): DateTime
public setTime(
    int $hour,
    int $minute,
    int $second = 0,
    int $microsecond = 0
): DateTime
public setTimestamp(int $timestamp): DateTime
public setTimezone(DateTimeZone $timezone): DateTime
public sub(DateInterval $interval): DateTime
public diff(DateTimeInterface $targetObject, bool $absolute = false):
DateInterval
public format(string $format): string
public getOffset(): int
public getTimestamp(): int
public getTimezone(): DateTimeZone|false
public __wakeup(): void
```

165. Опишите использование класса DateTimeImmutable.

Листинг - Примеры констант и методов DateTimeImmutable

```
class DateTimeImmutable implements DateTimeInterface {
/* Наследуемые константы */
const string DateTimeInterface::ATOM = "Y-m-d\TH:i:sP";
const string DateTimeInterface::COOKIE = "l, d-M-Y H:i:s T";
const string DateTimeInterface::ISO8601 = "Y-m-d\TH:i:sO";
const string DateTimeInterface::RFC822 = "D, d M y H:i:s O";
const string DateTimeInterface::RFC850 = "l, d-M-y H:i:s T";
const string DateTimeInterface::RFC1036 = "D, d M y H:i:s O";
const string DateTimeInterface::RFC1123 = "D, d M Y H:i:s O";
const string DateTimeInterface::RFC7231 = "D, d M Y H:i:s \G\M\T";
const string DateTimeInterface::RFC2822 = "D, d M Y H:i:s O";
const string DateTimeInterface::RFC3339 = "Y-m-d\TH:i:sP";
const string DateTimeInterface::RFC3339_EXTENDED = "Y-m-d\TH:i:s.vP";
const string DateTimeInterface::RSS = "D, d M Y H:i:s O";
const string DateTimeInterface::W3C = "Y-m-d\TH:i:sP";
/* Методы */
public __construct(string $datetime = "now", ?DateTimeZone $timezone =
null)
public add(DateInterval $interval): DateTimeImmutable
public static createFromFormat(string $format, string $datetime,
?DateTimeZone $timezone = null): DateTimeImmutable|false
public static createFromInterface(DateTimeInterface $object):
DateTimeImmutable
public static createFromMutable(DateTime $object): DateTimeImmutable
public static getLastErrors(): array|false
public modify(string $modifier): DateTimeImmutable|false
public static __set_state(array $array): DateTimeImmutable
public setDate(int $year, int $month, int $day): DateTimeImmutable
public setISODate(int $year, int $week, int $dayOfWeek = 1):
DateTimeImmutable
public setTime(
    int $hour,
    int $minute,
    int $second = 0,
    int $microsecond = 0
): DateTimeImmutable
public setTimestamp(int $timestamp): DateTimeImmutable
public setTimezone(DateTimeZone $timezone): DateTimeImmutable
public sub(DateInterval $interval): DateTimeImmutable
public diff(DateTimeInterface $targetObject, bool $absolute = false):
DateInterval
public format(string $format): string
public getOffset(): int
public getTimestamp(): int
public getTimezone(): DateTimeZone|false
public __wakeup(): void
}
```

166. Опишите использование класса DateTimeZone.

Листинг - Примеры констант и методов DateTimeZone

```
class DateTimeZone {
/* Константы */
const int AFRICA = 1;
const int AMERICA = 2;
const int ANTARCTICA = 4;
const int ARCTIC = 8;
const int ASIA = 16;
const int ATLANTIC = 32;
const int AUSTRALIA = 64;
const int EUROPE = 128;
const int INDIAN = 256;
const int PACIFIC = 512;
const int UTC = 1024;
const int ALL = 2047;
const int ALL_WITH_BC = 4095;
const int PER_COUNTRY = 4096;
/* Методы */
public __construct(string $timezone)
public getLocation(): array|false
public getName(): string
public getOffset(DateTimeInterface $datetime): int
public getTransitions(int $timestampBegin = PHP_INT_MIN, int
$timestampEnd = PHP_INT_MAX): array|false
public static listAbbreviations(): array
public static listIdentifiers(int $timezoneGroup = DateTimeZone::ALL,
?string $countryCode = null): array
}
```

167. Какие библиотеки используются для работы с изображениями в PHP.

1. Goutte
2. GoogChart
3. JpGraph
4. Imagine
5. Php Graphic Works
6. Zebra Image
7. Php5 Image Manipulation
8. Dynamic Dummy Image Generator
9. WideImage
10. Image Cache

168. Опишите основные возможности библиотеки GD.

GD Graphics Library (GD) — программная библиотека, написанная Томасом Баутелом (Thomas Boutell) и другими разработчиками для динамической работы с изображениями.

В PHP библиотека GD значительно расширена. Начиная с версии PHP 4.3 входит в стандартную поставку интерпретатора. До этой версии могла подключаться как отдельная библиотека. Поддерживает почти все существующие форматы графики для использования в веб: PNG, JPEG, GIF, ICO и различные методы работы с графическими файлами (применение фильтров, текст, изменение размера, и прочее).

Функции GD и функции для работы с изображениями:

- getimagesize — Получение размера изображения
- image_type_to_extension — Получение расширения файла для типа изображения
- mage2wbmp — Выводит изображение в браузер или пишет в файл
- imagealphablending — Задание режима сопряжения цветов для изображения
- imagearc — Рисование дуги
- imageavif — Выводит изображение в браузер или пишет в файл
- imagebmp — Вывести BMP-изображение в браузер или файл
- imagechar — Рисование символа по горизонтали
- imagecharup — Рисование символа вертикально
- imagecolorallocate — Создание цвета для изображения
- imagecolorallocatealpha — Создание цвета для изображения
- imagecolorat — Получение индекса цвета пиксела
- imagecolorexact — Получение индекса заданного цвета
- imagecolorset — Установка набора цветов для заданного индекса палитры
- imagecolorsforindex — Получение цветов, соответствующих индексу
- imagecolorstotal — Определение количества цветов в палитре изображения
- imagecopy — Копирование части изображения
- imagecopyresized — Копирование и изменение размера части изображения
- imagecreate — Создание нового палитрового изображения
- imagecreatefromavif — Создаёт новое изображение из файла или URL
- imagecreatefrombmp — Создаёт новое изображение из файла или URL
- imagecreatetruecolor — Создание нового полноцветного изображения
- imagecrop — Обрезать изображение до заданного прямоугольника
- imagedashedline — Рисование пунктирной линии
- imagedestroy — Уничтожение изображения
- imageellipse — Рисование эллипса
- imagefill — Заливка

- imagefilledarc — Рисование и заливка дуги
- imagefilledellipse — Рисование закрашенного эллипса
- imagefilledpolygon — Рисование закрашенного многоугольника
- imagefilledrectangle — Рисование закрашенного прямоугольника
- imagefilltoborder — Заливка цветом

и т.д.

169. Как использовать Composer для подключения библиотек к проекту?

Composer скачивает и устанавливает библиотеки по их имени. Это означает, что сначала нужно найти нужную библиотеку, перейти на её сайт, и найти там в описании её имя. Например, название библиотеки может быть таким: fzaninotto/faker.

Теперь мы можем попросить composer установить библиотеку. Для этого введите команду `composer require <имя библиотеки>`. Composer загрузит и установит библиотеку в папку vendor. Останется подключить установленную библиотеку в сценарии и можно её использовать.

Пример для библиотеки для валидации форм — GUMP внутри php:

```
<?php
require 'vendor/autoload.php';

$rules = [
    'email' => 'required|valid_email',
    'password' => 'required|min_len,8',
    'login' => 'required|alpha_numeric',
    'phone' => 'phone_number'
];

$gump = new GUMP('ru');
$gump->validation_rules($rules);
$validated_data = $gump->run($_POST);
```

170. Что такое PEAR? В чём разница работы PEAR и Composer?

PEAR - это аббревиатура от "PHP Extension and Application Repository" (Репозиторий приложений и модулей PHP).

PEAR - это:

структурированная библиотека открытого кода, созданная для пользователей PHP;

система управления пакетами и распространения кода среди разработчиков;

стандарт написания PHP-кода (подробнее о стандарте см. здесь);

базовые классы PHP-кода (подробнее о базовых классах см. здесь;

библиотека дополнительных модулей для PHP (The PHP Extension Code Library, PECL), подробную информацию о PECL можно узнать здесь;

веб-сайт, листы рассылки и зеркала для загрузки - все это предназначено для поддержания и развития сообщества разработчиков PHP/PEAR.

Различия PEAR и Composer:
Невозможно установить пакет в PEAR для одного проекта, все пакеты устанавливаются глобально, тогда как в Composer можно выбирать;
Отсутствие управления зависимостями в PEAR;
В PEAR лучше совместимость с PECL. Хотя, существует pickle , который использует Composer и позволяет вам определять собственные зависимости расширений PHP в вашем файле composer.json . Проект pickle в настоящее время находится в стадии разработки и пока не должен считаться зрелым

171. Как использовать PEAR для установки библиотек?

```
pear install <имя библиотеки>
```

172. Как использовать Composer для обработки зависимостей PEAR?

Флаги для установки модулей с зависимостями:

- all deps - необходимые и необязательные зависимости
- only req deps - только необходимые зависимости []

173. Что такое PECL?

PECL — это репозиторий модулей для PHP, написанных на C, доступных через систему пакетов PEAR. PECL был создан, когда возникла проблема удаления некоторых модулей из стандартной поставки PHP. Модули PECL разработаны в соответствии со стандартами кодирования, которые приняты командой разработчиков PHP.

174. Как называется репозиторий, содержащий Composer-совместимые библиотеки?

Pickle

175. С помощью какой библиотеки можно получить детальный отчет о работе приложения?

PHP BenchMark

176. С помощью какой библиотеки можно упростить себе работу с регулярными выражениями в PHP?

RegExp Builder

177. С помощью какой библиотеки возможны быстрые и эффективные запросы на PHP, данная библиотека является аналогом jQuery.

phpQuery

178. С помощью какой библиотеки возможно простое и эффективное генерирование документов в формате PDF?

mPDF

179. С помощью какой библиотеки возможен эффективный парсинг HTML/XML?

Goutte

180. С помощью какой библиотеки возможно создание диаграмм на движке GOOGLE?

GoogChart

181. С помощью какой библиотеки возможно сканирование конфигурационного файла PHP на предмет безопасности?

IniScan

182. С помощью какой библиотеки возможен анализ HTML и удаление вредоносного кода для защиты от XSS атак.

HTML Purifier

183. С помощью какой библиотеки возможно построение графиков, диаграмм и другого структурированного контента на PHP?

JpGraph

184. С помощью какой библиотеки облегчается процесс загрузки и валидации файлов на PHP?

Upload

\\\ВОПРОСЫ ИЗ 7 ПРАКТИКИ///

185. Назовите основные признаки ООП.

Абстракция – это способ придания объектам определённых характеристик, которые отличают его от всех остальных объектов.
Инкапсуляция – один из принципов ООП, позволяющее объединить данные и методы, работающие с ними, в классе.
Наследование – один из принципов ООП, позволяющий описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью.
Полиморфизм – один из принципов ООП, позволяющий использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта

186. Опишите как определить класс в PHP.

Для примера представлен класс машины.

```
<?php
class Car {
    public $brand;
    public $color;
    public function __construct($brand, $color = "blue")
    {
        $this->brand = $brand;
        $this->color = $color;
    }
    public function getBrand()
    {
        return $this->brand;
    }
    public function getColor()
    {
        return $this->color;
    }
}
?>
```

187. Как создать экземпляр класса в PHP.

Создадим экземпляр класса машины. Ниже представлен пример.

```
$car = new Car(“Mercedes”, “black”);
```

188. Опишите механизм наследования в PHP.

```
<?php
class A {
```

```
    // code here
}
class B extends A {
    // code here
}
class C extends B {
    // code here
}
?>
```

189. Опишите правила совместимости сигнатур.

При переопределении метода его сигнатура должна быть совместима с родительским методом. В противном случае выдаётся фатальная ошибка или, до PHP 8.0.0, генерируется ошибка уровня E_WARNING. Сигнатура является совместимой, если она соответствует правилам контравариантности, делает обязательный параметр необязательным и если какие-либо новые параметры являются необязательными. Это известно как принцип подстановки Барбары Лисков или сокращённо LSP. Правила совместимости не распространяются на конструктор и сигнатуру private методов, они не будут выдавать фатальную ошибку в случае несоответствия сигнатуры. Ниже представлен пример.

```
<?php
class Base
{
   public function foo(int $a) {
      echo "Допустимо\n";
   }
}
class Extend1 extends Base
{
   function foo(int $a = 5)
   {
      parent::foo($a);
   }
}
class Extend2 extends Base
{
   function foo(int $a, $b = 5)
   {
      parent::foo($a);
   }
}
$extended1 = new Extend1();
$extended1->foo();
```

```
$extended2 = new Extend2();
$extended2->foo(1);
```

190. Опишите методы и свойства Nullsafe.

Начиная с PHP 8.0.0, к свойствам и методам можно также обращаться с помощью оператора "nullsafe": ?->. Оператор nullsafe работает так же, как доступ к свойству или методу, как указано выше, за исключением того, что если разыменование объекта выдаёт null, то будет возвращён null, а не выброшено исключение. Если разыменование является частью цепочки, остальная часть цепочки пропускается.
Аналогично заключению каждого обращения в is_null(), но более компактный. Ниже представлен пример.

```
<?php
// Начиная с PHP 8.0.0, эта строка:
$result = $repository?->getUser(5)?->name;
// Эквивалентна следующему блоку кода:
if (is_null($repository)) {
   $result = null;
} else {
   $user = $repository->getUser(5);
   if (is_null($user)) {
      $result = null;
   } else {
      $result = $user->name;
   }
}
?>
```

191. Опишите понятие автоматическая загрузка классов.

Большинство разработчиков объектно-ориентированных приложений используют такое соглашение именования файлов, в котором каждый класс хранится в отдельно созданном для него файле. Одна из самых больших неприятностей - необходимость писать в начале каждого скрипта длинный список подгружаемых файлов (по одному для каждого класса).
Функция spl_autoload_register() позволяет зарегистрировать необходимое количество автозагрузчиков для автоматической загрузки классов и интерфейсов, если они в настоящее время не определены. Регистрируя автозагрузчики, PHP получает последний шанс для интерпретатора загрузить класс прежде, чем он закончит выполнение скрипта с ошибкой. Ниже представлен пример автоматической загрузки.

```
<?php
spl_autoload_register(function ($class_name) {
   include $class_name . '.php';
});
$obj = new MyClass1();
$obj2 = new MyClass2();
?>
```

192. Опишите конструкторы и деструкторы в PHP.

PHP позволяет объявлять методы-конструкторы. Классы, в которых объявлен метод-конструктор, будут вызывать этот метод при каждом создании нового объекта, так что это может оказаться полезным, например, для инициализации какого-либо состояния объекта перед его использованием. Конструктор задаётся следующим образом.

```
__construct(mixed ...$values = ""): void
```

PHP предоставляет концепцию деструктора, аналогичную с той, которая применяется в других ОО-языках, таких как C++. Деструктор будет вызван при освобождении всех ссылок на определённый объект или при завершении скрипта (порядок выполнения деструкторов не гарантируется).

```
__destruct(): void
```

193. Опишите понятие области видимости и модификаторы доступа в PHP.

Область видимости свойства, метода или константы (начиная с PHP 7.1.0) может быть определена путём использования следующих ключевых слов в объявлении: public, protected или private. Доступ к свойствам и методам класса, объявленным как public (общедоступный), разрешён отовсюду. Модификатор protected (защищённый) разрешает доступ самому классу, наследующим его классам и родительским классам. Модификатор private (закрытый) ограничивает область видимости так, что только класс, где объявлен сам элемент, имеет к нему доступ

194. Опишите оператор разрешения области видимости.

Оператор разрешения области видимости (также называемый "Paamayim Nekudotayim") или просто "двойное двоеточие" — это лексема, позволяющая обращаться к статическим свойствам, константам и переопределённым свойствам или методам класса.
При обращении к этим элементам извне класса, необходимо использовать имя этого класса.

195. Опишите абстрактные классы и методы в PHP.

PHP поддерживает определение абстрактных классов и методов. На основе абстрактного класса нельзя создавать объекты, и любой класс, содержащий хотя бы один абстрактный метод, должен быть определён как абстрактный. Методы, объявленные абстрактными, несут, по существу, лишь описательный смысл и не могут включать реализацию.
При наследовании от абстрактного класса, все методы, помеченные абстрактными в родительском классе, должны быть определены в дочернем классе и следовать обычным правилам наследования и совместимости сигнатуры.
Ниже представлен пример наследования от абстрактного класса на PHP.

```
<?php
abstract class AbstractClass
{
  /* Данный метод должен быть определён в дочернем классе */
   abstract protected function getValue();
   abstract protected function prefixValue($prefix);
```

```
   /* Общий метод */
    public function printOut() {
        print $this->getValue() . "\n";
    }
}
class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }
    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}
?>
```

196. Опишите интерфейсы в PHP.

Интерфейсы объектов позволяют создавать код, который указывает, какие методы должен реализовать класс, без необходимости определять, как именно они должны быть реализованы. Интерфейсы разделяют пространство имён с классами и трейтами, поэтому они не могут называться одинаково.

Ниже приведён пример интерфейса и его имплементации.

```
<?php
// Объявим интерфейс 'Template'
interface Template
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Реализация интерфейса
// Это будет работать
class WorkingTemplate implements Template
{
    private $vars = [];
    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }
```

```
        return $template;
    }
}
?>
```

197. Что такое трейт и как это используется?

Трейт — это механизм обеспечения повторного использования кода в языках с поддержкой только одиночного наследования, таких как PHP. Трейт предназначен для уменьшения некоторых ограничений одиночного наследования, позволяя разработчику повторно использовать наборы методов свободно, в нескольких независимых классах и реализованных с использованием разных архитектур построения классов. Семантика комбинации трейтов и классов определена таким образом, чтобы снизить уровень сложности, а также избежать типичных проблем, связанных с множественным наследованием и смешиванием (mixins).

198. Что такое магические методы? Приведите примеры.

Магические методы — это специальные методы, которые переопределяют действие PHP по умолчанию, когда над объектом выполняются определённые действия.
Например,__construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), _ _sleep(), __wakeup(), __serialize(), __unserialize(), __toString(), __invoke(), __set_state(), __clone() и __debugInfo().

199. Что такое позднее статическое связывание?

PHP реализует функцию, называемую позднее статическое связывание, которая может быть использована для того, чтобы получить ссылку на вызываемый класс в контексте статического наследования.
Если говорить более точно, позднее статическое связывание сохраняет имя класса указанного в последнем "неперенаправленном вызове". В случае статических вызовов это явно указанный класс (обычно слева от оператора ::); в случае не статических вызовов это класс объекта. "Перенаправленный вызов" - это статический вызов, начинающийся с self::, parent::, static::, или, если двигаться вверх по иерархии классов, forward_static_call(). Функция get_called_class() может быть использована для получения строки с именем вызванного класса, а static:: представляет её область действия.
Само название "позднее статическое связывание" отражает в себе внутреннюю реализацию этой особенности. "Позднее связывание" отражает тот факт, что обращения через static:: не будут вычисляться по отношению к классу, в котором вызываемый метод определён, а будут вычисляться на основе информации в ходе исполнения. Также эта особенность была названа "статическое связывание" потому, что она может быть использована (но не обязательно) в статических методах

200. Что такое ковариантность и контравариантность?

Ковариантность – это сохранение иерархии наследования исходных типов в производных типах в том же порядке. Ниже приведён пример ковариантности на языке PHP.

```
<?php
abstract class Animal
{
    protected string $name;
    public function __construct(string $name)
    {
        $this->name = $name;
    }
    abstract public function speak();
}
class Dog extends Animal
{
    public function speak()
    {
        echo $this->name . " лает";
    }
}
class Cat extends Animal
{
    public function speak()
    {
        echo $this->name . " мяукает";
    }
}
interface AnimalShelter
{
    public function adopt(string $name): Animal;
}
class CatShelter implements AnimalShelter
{
    public function adopt(string $name): Cat // Возвращаем класс Cat вместо Animal
    {
        return new Cat($name);
    }
}
class DogShelter implements AnimalShelter
{
    // Возвращаем класс Dog вместо Animal
    public function adopt(string $name): Dog
    {
        return new Dog($name);
    }
}
?>
```

Контравариантность – это обращение иерархии исходных типов на противоположную в производных типах. Ниже приведён пример контравариантности на языке PHP.

```
<?php
class Food {}
class AnimalFood extends Food {}
abstract class Animal
{
   protected string $name;
   public function __construct(string $name)
   {
      $this->name = $name;
   }
   public function eat(AnimalFood $food)
   {
      echo $this->name . " ест " . get_class($food);
   }
}
class Dog extends Animal
{
   public function eat(Food $food) {
      echo $this->name . " ест " . get_class($food);
   }
}
?>
```

201. Опишите понятие чистой архитектуры.

Понятие чистой архитектуры пошло из одноименной статьи Роберта Мартина 2012 года. Оно включает в себя несколько принципов:

- Независимость от фреймворков. Архитектура не должна полагаться на существование какой-либо библиотеки. Так вы сможете использовать фреймворки как инструменты, а не пытаться загнать свою систему в их ограничения;
- Тестируемость. Бизнес-логика должна быть тестируемой без любых внешних элементов вроде интерфейса, базы данных, сервера или любого другого элемента;
- Независимость от интерфейса. Интерфейс должен легко изменяться и не требовать изменения остальной системы. Например, веб-интерфейс должен заменяться на интерфейс консоли без необходимости изменения бизнес-логики;
- Независимость от базы данных. Ваша бизнес-логика не должна быть привязана и к конкретным базам данных;

Независимость от любого внешнего агента. Ваша бизнес-логика не должна знать вообще ничего о внешнем мире.

202. Сформулируйте правило зависимостей.

Зависимости в исходном коде могут указывать только во внутрь. Ничто из внутреннего круга не может знать что-либо о внешнем круге, ничто из внутреннего круга не может указывать на внешний круг. Это касается функций, классов, переменных и т. д. Более того, структуры данных, используемых во внешнем круге, не должны быть использованы во внутреннем круге, особенно если эти структуры генерируются фреймворком во внешнем круге. Мы не используем ничего из внешнего круга, чтобы могло повлиять на внутренний.

203. Чем определяются сущности, чем они могут быть?

Сущность (entity) – это объект, который может быть идентифицирован неким способом, отличающим его от других объектов. Примеры: конкретный человек, предприятие, событие и т.д.
Сущности определяются бизнес-правилами предприятия. Сущность может быть объектом с методами или она может представлять собой набор структур данных и функций. Не имеет значения как долго сущность может быть использована в разных приложениях.
Если же вы пишете просто одиночное приложение, в этом случае сущностями являются бизнес-объекты этого приложения. Они инкапсулируют наиболее общие высокоуровневые правила. Наименее вероятно, что они изменятся при каких-либо внешних изменениях.
Например, они не должны быть затронуты при изменении навигации по страницам или правил безопасности. Внешние изменения не должны влиять на слой сущностей

204. Что такое слой сценариев?

В данном слое реализуется специфика бизнес-правил. Он инкапсулирует и реализует все случаи использования системы. Эти сценарии реализуют поток данных в и из слоя сущностей для реализации бизнес-правил.
Мы не ожидаем изменения в этом слое, влияющих на сущности. Мы также не ожидаем, что этот слой может быть затронут внешними изменениями, таких как базы данных, пользовательским интерфейсом или фреймворком. Этот слой изолирован от таких проблем.
Мы, однако, ожидаем, что изменения в работе приложения повлияет на Сценарии. Если будут какие-либо изменения в поведении приложения, то они несомненно затронут код в данном слое.

205. Что такое DTO?

Data Transfer Object (DTO) — один из шаблонов проектирования, используется для передачи данных между подсистемами приложения

206. Что является деталью в рамках чистой архитектуры?

Это части структуры, которые позволяют миру взаимодействовать с бизнес-правилами.
Например: веб, фреймворк

207. Опишите принципы организации компонентов.

Под понятием компонентов принято понимать единицы развертывания. Например, это может быть библиотека или исполняемый файл, в Java таковым назвать можно jar-файлы. Принципы организации компонентов в разработки ПО помогают сгруппировать классы в компоненты и сделать их более структурируемыми и управляемыми. Принципы разделяются на две группы:

связность (component cohesion) (какие классы стоит поместить?) и сочетаемость (component coupling) компонентов (как должны взаимодействовать друг с другом?).

Три принципа связности компонентов:

- Принцип эквивалентности повторного использования и выпусков. Единица повторного использования есть единица выпуска. Этот принцип требует, чтобы компоненты проходили процесс выпуска и получали версии. В один компонент должны объединяться классы с одной целью, которая будет выражена в очередном выпуске. По документации же пользователи и разработчики должны понимать, нужно ли им переходить на новую версию.
- Принцип согласованного изменения. В один компонент должны объединяться классы, изменяющиеся по одним причинам. Идея заключения вместе сущностей, которые могут изменяться в одно и то же время и по одним причинам, является одной из ключевых идей архитектуры ПО, поэтому она проходит красной нитью по всем структурным уровням.
- Принцип совместного повторного использования. Не вынуждайте пользователей компонента зависеть от того, чего им не требуется. Главная мысль этого принципа в объединении в компоненты тех классов, которые имеют множественные зависимости друг от друга. Кроме того, нужно избегать слабых зависимостей от других компонентов — даже зависимость от одного редко используемого класса наверняка потребует повторной компиляции и тестирования всего компонента в случае изменений в зависимом.

Теперь перейдем ко второй группе принципов. Сочетаемость компонентов:

- Принцип ацикличности зависимостей. Нельзя допускать циклов в графе зависимостей. Это позволяет разбить проект на компоненты, которые будут выпускаться независимо. При возникновении цикла между зависимостями для тестирования и выпуска новой версии придется отладить и подготовить все компоненты, входящие в цикл. Они превращаются в один большой компонент.
- Принцип устойчивых зависимостей. Зависимости должны быть направлены в сторону устойчивых компонентов. Нужно использовать ссылки на компоненты, которые будут редко меняться и избегать зависимостей от изменчивых компонентов. Это добавит гибкость в разработке, так как всегда нужны компоненты, которые можно легко изменять. Если же создать большое число зависимостей от такого компонента, эта возможность легких изменений испарится.

Принцип устойчивости абстракций. Этот принцип проводит связь между устойчивыми и абстрактными компонентами. Компоненты, которые содержат интерфейс для высокоуровневой бизнес-логики, должны быть абстрактными и почти не меняться. Реализации же этого

интерфейса должны быть неустойчивыми — избегайте множественных зависимостей от таких компонентов

208. Опишите принципы дизайна архитектуры.

Под принципами дизайна архитектуры понимаются SOLID принципы. Эта аббревиатура пяти основных принципов проектирования в объектно-ориентированном программировании — Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion. В переводе на русский: принципы единственной ответственности, открытости / закрытости, подстановки Барбары Лисков, разделения интерфейса и инверсии зависимостей).

- Принцип единственной обязанности / ответственности (single responsibility principle / SRP) обозначает, что каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности.
- Принцип открытости / закрытости (open-closed principle / OCP) декларирует, что программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения. Это означает, что эти сущности могут менять свое поведение без изменения их исходного кода.
- Принцип подстановки Барбары Лисков (Liskov substitution principle / LSP) в формулировке Роберта Мартина: «функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом».
- Принцип разделения интерфейса (interface segregation principle / ISP) в формулировке Роберта Мартина: «клиенты не должны зависеть от методов, которые они не используют». Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

Принцип инверсии зависимостей (dependency inversion principle / DIP) — модули верхних уровней не должны зависеть от модулей нижних уровней, а оба типа модулей должны зависеть от абстракций; сами абстракции не должны зависеть от деталей, а вот детали должны зависеть от абстракций

209. Опишите понятие DDD (Domain Driven Design, предметно-ориентированное проектирование).

Предметно-ориентированное проектирование — это набор принципов и схем, направленных на создание оптимальных систем объектов. Сводится к созданию программных абстракций, которые называются моделями предметных областей. В эти модели входит бизнес-логика, устанавливающая связь между реальными условиями области применения продукта и кодом.

210. Что такое ограниченный контекст (Bounded Context)?

Ограниченный контекст (bounded context)– это граница, которая окружает ту или иную модель. Это держит знания внутри соответствующей границы, в то же время игнорируя помехи от внешнего мира. Это выгодно по ряду причин. Во-первых, вы можете моделировать различные аспекты проблемы, не контактируя с другими частями бизнеса. Используя предыдущий пример, объект Product в рамках складской системы должен быть связан с помощью методов и свойств этой единой системы, а не какой-либо другой коммерческой фирмы, что случается, когда пытаются соответствовать объекту под названием Product. Во-вторых, терминология в ограниченном контексте (Bounded Context) может иметь одно, четкое определение, что точно описывает конкретную проблему. Различные отделы по всей компании, как правило, имеют немного разные идеи и определения аналогичных условий, это часто может сорвать проект из-за отсутствия ясности и понимания, если сроки являются неоднозначными

211. Что такое Ubiquitous Language (Единый язык)?

Этот коллективный язык терминов называется - единый язык. (Ubiquitous Language). Это один из основных и самых важных шаблонов предметного-ориентированного проектирования. Это не бизнес-жаргон, навязанный разработчикам, а настоящий язык, созданный целостной командой – экспертами в предметной области, разработчиками, бизнес-аналитиками и всеми, кто вовлечен в создание системы. Роль в команде не столь существенна, поскольку каждый член команды использует для описания проекта единый язык.

212. Что такое Смысловое ядро (Core domain)?

Смысловое ядро – это подобласть, имеющая первостепенное значение для организации. Со стратегической точки зрения бизнес должен выделяться своим смысловым ядром. Большинство DDD проектов сосредоточены именно на смысловом ядре.

213. Что такое Предметная область (Domain)?

Множество понятий и объектов, рассматриваемых в пределах отдельного рассуждения, исследования или научной теории. Включает объекты, изучаемые теорией, а также свойства, отношения и функции, которые принимаются во внимание в теории. В анализе данных в качестве предметной области может выступать компания, в интересах которой реализуется аналитический проект, внешнее окружение, сегмент

рынка и т. д. Это понятие играет большую роль в анализе данных, поскольку используемые там подходы и методы оперируют объектами и терминами предметной области и, следовательно, зависят от нее. В хранилищах данных, которые являются предметно ориентированными, под предметной областью понимают устойчивую связь между именами, понятиями и объектами внешнего мира, не зависящую от самой информационной системы и круга ее пользователей. Введение в рассмотрение понятия предметной области ограничивает и делает обозримым пространство информационного поиска в хранилище данных и позволяет выполнять за конечное время даже сложные нерегламентированные запросы

214. Что такое Пространство задач и пространство решений?

Пространство задач и пространство решений в архитектуре подразумевает под собой набор определённых задач и их решений, в зависимости от которых будет зависеть построение целой архитектуры информационной системы/приложения

\\\ОТСЮДА ИДЁТ ПИЗДЕЦ (8 ПРАКТИКА)///

215. Что такое фреймворк?

Программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта

216. Что такое фреймворк и чем он отличается от библиотеки?

«Фреймворк» отличается от понятия библиотеки тем, что библиотека может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений. В то время как «фреймворк» диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию — «каркас», который нужно будет расширять и изменять, согласно указанным требованиям.

Также, в отличие от библиотеки, которая объединяет в себе набор близкой функциональности, — «фреймворк» может содержать в себе большое число разных по тематике библиотек.

Другим ключевым отличием «фреймворка» от библиотеки может быть инверсия управления: пользовательский код вызывает функции библиотеки (или классы) и получает управление после вызова. Во «фреймворке» пользовательский код может реализовывать конкретное поведение, встраиваемое в более общий — «абстрактный» код фреймворка. При этом «фреймворк» вызывает функции (классы) пользовательского кода

217. Из каких этапов состоит разработка веб-приложения?
Планирование, проектирование, тестирование, запуск

218. Что такое “Гибкий” фреймворк?
Фреймворк, следующий гибкой методологии разработки

219. Что такое “не гибкий” фреймворк?
Фреймворк, не следующий гибкой методологии разработки

220. Чем “гибкий” фреймворк отличается от “не гибкого” фреймворка?
В основном – разные жизненные циклы приложения. Большая возможность соответствия декларируемым в agile-манифесте принципам

221. Опишите фреймворк Laravel.
Бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC

222. Какие приложения можно создавать с помощью Laravel?
С помощью Laravel, как и на базе любого другого фреймворка, можно делать абсолютно разные типы сайтов, начиная с лендингов и заканчивая социальными сетями

223. Что такое Eloquent ORM?
Система объектно-реляционного отображения (ORM) Eloquent — красивая и простая реализация шаблона ActiveRecord в Laravel для

работы с базами данных. Каждая таблица имеет соответствующий класс-модель, который используется для работы с этой таблицей. Модели позволяют запрашивать данные из таблиц, а также вставлять в них новые записи

224. Что такое сервис-провайдеры (service providers)?

Сервис-провайдеры лежат в основе первоначальной загрузки всех приложений на Laravel. И ваше приложение, и все базовые сервисы Laravel загружаются через сервис-провайдеры.

Но что мы понимаем под "первоначальной загрузкой"? В общих чертах, мы имеем ввиду регистрацию таких вещей, как биндингов в IoC-контейнер (фасадов и т.д.), слушателей событий, фильтров роутов и даже самих роутов. Сервис-провайдеры - центральное место для конфигурирования вашего приложения.

Если вы откроете файл config/app.php, поставляемый с Laravel, то увидите массив providers. В нём перечислены все классы сервис-провайдеров, которые загружаются для вашего приложения. Конечно, многие из них являются "отложенными" провайдерами, т.е. они не загружаются при каждом запросе, а только при необходимости.

225. Что такое Сервис-контейнер в Laravel?

Сервис-контейнер в Laravel — это мощное средство для управления зависимостями классов и внедрения зависимостей. Внедрение зависимостей — это распространенный термин, который означает добавление других классов в этот класс через конструктор или, в некоторых случаях, метод-сеттер.

226. Что такое Контракты в Laravel?

Контракты в Laravel — это набор интерфейсов, которые описывают основной функционал, предоставляемый фреймворком. Например, контракт Illuminate\Contracts\Queue\Queue определяет методы, необходимые для организации очередей, в то время как контракт Illuminate\Contracts\Mail\Mailer определяет методы, необходимые для отправки электронной почты.

227. Что такое Фасады в Laravel?

Фасады предоставляют "статический" интерфейс к классам, доступным в сервис-контейнере. Laravel поставляется со множеством фасадов, которые предоставляют доступ практически ко всем функциям Laravel. Фасады Laravel служат "статическими прокси" для основополагающих классов в сервис-контейнере, предоставляя преимущество лаконичного, выразительного синтаксиса, сохраняя при этом большую тестируемость и гибкость по сравнению с обычными статическими методами.

228. Опишите структуру Laravel-приложения по умолчанию.

- Корневой каталог
  - Каталог app
  - Каталог bootstrap
  - Каталог config
  - Каталог database
  - Каталог public
  - Каталог resources
  - Каталог routes
  - Каталог storage
  - Каталог tests
  - Каталог vendor
- Каталог пространства App
  - Каталог пространства Broadcasting
  - Каталог пространства Console
  - Каталог пространства Events
  - Каталог пространства Exceptions
  - Каталог пространства Http
  - Каталог пространства Jobs
  - Каталог пространства Listeners

- Каталог пространства Mail
- Каталог пространства Models
- Каталог пространства Notifications
- Каталог пространства Policies
- Каталог пространства Providers

Каталог пространства Rules

229. Опишите преимущества фреймворка Laravel.

Достаточно неплохая и понятная документация.

Вокруг фреймворка создана мощная экосистема. Различные курсы, конференции, обучающие материалы позволяют собрать вокруг фреймворка большое количество разработчиков и спонсоров, которые заинтересованы в развитии инструмента и принимают в этом участие. Да, здесь чувствуется запах маркетинга, и неплохой.

Одним из самых очевидных плюсов Laravel, является гибкая система маршрутизации, позволяющая составить самые разные проверки маршрута веб-приложения. Вы можете выделить маршруты в специальные группы, использовать пространство имен, указать параметры маршрута, использовать регулярные выражения, настроить поддоменную маршрутизацию и многое другое.

В Laravel много синтаксического сахара. Синтаксис API фреймворка достаточно простой и понятный. Здесь нет длинных и сложных конструкций, а только краткие и продуманные названия функций.

Laravel содержит удобный механизм обработки ошибок и исключений.

Фреймворк включает в себя встроенные механизмы аутентификации и авторизации пользователей, которую можно перенастроить под свои потребности.

Laravel предоставляет из коробки механизмы для кэширования веб-приложения с помощью Memcached и Redis. Кроме этого есть удобные функции для использования простого файлового кэширования данных.

Laravel предоставляет чистый и простой API поверх популярной библиотеки SwiftMailer с драйверами для SMTP, Mailgun, SparkPost, Amazon SES и sendmail, чтобы сделать отправку почты через локальную или облачную службу по выбору. В том числе есть механизм для построения очередей отправки почты.

Laravel Cashier обеспечивает выразительный, свободный интерфейс к сервисам биллинга по подписке Stripe и Braintree.

230. Опишите недостатки фреймворка Laravel.

Синтаксический сахар в Laravel как плюс, так может быть и минусом. Очень легко привыкнуть к нему и позабыть, как пишутся чистые запросы и функции.

Нарушение обратная совместимости между версиями фреймворка.

Не логичное расположение каталогов и файлов. Например, по умолчанию в прямо в каталоге /app расположена модель User.php, которую логичней было бы расположить в каталоге /app/Models. Каталог resources с файлами представления размещен в корне приложения, хотя логичней будет его разместить в /app/resources.

231. Что включает в себя фреймворк Laravel.

Laravel включает в себя встроенную поддержку для аутентификации, локализации, модели, представления, сессий, маршрутизации и других механизмов, поддержку контроллеров, которые сделали фреймворк полностью MVC-совместимым, встроенную поддержку для инверсии управления и шаблонизатор Blade, интерфейс командной строки (CLI) под именем «Artisan», встроенную поддержку нескольких систем управления базами данных, миграции баз

данных в виде контроля версий, обработку событий, выгрузка таблиц базы данных для первоначальной популяции, поддержку очередей сообщений, встроенную поддержку отправки различных типов электронной почты и поддержку «мягкого» удаления записей базы данных, поддержку планирования периодически выполняемых задач через пакет Scheduler, слой абстракции Flysystem, который позволяет использовать удаленное хранилище так же, как и локальные файловые системы, улучшенную обработку активов пакета через Elixir и упрощенная аутентификация с внешней стороны через дополнительный пакет Socialite, Laravel Dusk, Laravel Mix, Blade Components и Slots, Markdown Emails, автоматические фасады, улучшения маршрута.

232. Опишите из чего состоит экосистема Laravel.

PHP, Composer, Dotenv, PSR-4, Eloquent ORM, Flysystem, Elixir, HHVM, Homestead, Rocketeer

233. Опишите фреймворк Symfony.

Свободный PHP фреймворк для быстрой разработки веб-приложений и решения рутинных задач веб-программистов.

Разработка и поддержка фреймворка спонсируется французской компанией Sensio.

Symfony состоит из набора не связанных между собой компонентов, которые можно использовать повторно в проектах.

Symfony позволяет устанавливать сторонние пакеты, библиотеки, компоненты и настраивать их с помощью конфигурации в форматах YAML, XML, PHP, а также .env файлах.

Symfony не обеспечивает компонент для работы с базой данных, но обеспечивает тесную интеграцию с библиотекой Doctrine.

Symfony предоставляет функцию почтовой программы на основе популярной библиотеки Swift Mailer. Эта почтовая программа

поддерживает отправку сообщений с ваших собственных почтовых серверов, а также с использованием популярных почтовых провайдеров, таких как Mandrill, SendGrid и Amazon SES.
Механизм интернационализации позволяет установить и произвести перевод сообщений веб-приложения на основе выбранного языка или страны.
Symfony предлагает систему логирования ошибок приложения, а также подключить библиотеку логирования Monolog.

234. Опишите преимущества фреймворка Symfony.

Мощная экосистема вокруг фреймворка, с хорошим сообществом и множеством разработчиков.
Хорошая и постоянно обновляемая документация для всех версий фреймворка.
Множество различных не связанных компонентов для повторного использования.
Предлагает механизм функциональных и модульных тестов для нахождения ошибок в веб-приложении.
Подходит для сложных и нагруженных веб-проектов.электронной коммерции

235. Опишите недостатки фреймворка Symfony.

Несмотря на хорошую документацию, фреймворк является сложным для изучения

236. Опишите фреймворк Code Igniter.

Это популярный PHP микро-фреймворк с открытым исходным кодом, для разработки веб-систем и приложений.
В CodeIgniter компоненты загружаются и процедуры выполняются только по запросу, а не глобально. Система не делает никаких

предположений относительно того, что может потребоваться помимо минимальных основных ресурсов, поэтому система по умолчанию очень легкая.

Компоненты фреймворка слабо связаны между собой и не зависят друг от друга. Чем меньше компонентов зависит друг от друга, тем более гибкой и многоразовой становится система.

Хотя CodeIgniter работает довольно быстро, объем динамической информации, отображаемой на страницах, будет напрямую зависеть от используемых ресурсов сервера, памяти и циклов обработки, которые влияют на скорость загрузки страниц.

Поэтому CodeIgniter позволяет кэшировать страницы для достижения максимальной производительности. с помощью встроенного компонента кэширования

237. Опишите преимущества фреймворка Code Igniter.

Отличная документация и англоязычное сообщество.

Высокая производительность фреймворка.

Небольшой размер фреймворка.

Предоставляет легкие и простые решения для разработки.

Подходит для быстрой разработки небольших сайтов и веб-приложений.

Структура фреймворка не требует строгих правил кодирования.

Не требует сложной настройки, почти нулевая конфигурация.

MVC-архитектура веб-приложения.

Слабая связанность компонентов.

Множество подключаемых библиотек и помощников

238. Опишите недостатки фреймворка Code Igniter.

Задержка в развитии и переходе на новые технологии

239. Опишите фреймворк Yii2.

Объектно-ориентированный компонентный фреймворк для PHP, реализующий парадигму MVC (Model-View-Controller). Yii является акронимом от “Yes It is”, на русском пишется и читается как “йии”. Yii2 является второй версией фреймворка Yii

240. Опишите преимущества фреймфорка Yii2.

Фреймворк прост в понимании.

Легко адаптируется под большие и маленькие проекты.

Имеет большое количество решений рутинных задач из коробки. К примеру, шаблон advanced обладает механизмом авторизации и аутентификации. Это довольно нужный механизм и он не очень прост в реализации.

Имеет замечательную документацию, гайды по старту и различные рецепты.

Yii2 популярен и довольно стар (релиз-то был аж в 2014 году), поэтому на рынке в много вакансий yii2-разработчиков, а с помощью развитого сообщества ответы на 90% вопросов вы найдете при легком гуглеже.

С помощью шаблонов и gii фреймворк подсказывает начинающему разработчику, как правильно располагать файлы. Сначала разработчик начинает повторять за тем, как это сделано в фреймворке, а потом понимает почему это хорошо. Разработчики фреймворка будто делятся опытом с новичком

241. Опишите Недостатки фреймворка Yii2.

Наличие различных антипаттернов в проекте — например, одиночка или божественный объект.

Встроенный класс User, являющийся потомком от ActiveRecord, показывает, как делать не нужно. Учит начинающих программистов, что классы, наследуемые от ActiveRecord, нужно раздувать различными методами, не связанными с работой с БД.

Сильная связность модулей в приложении. Говорят, эта проблема в Yii3 будет решена

Медленное развитие.

242. Опишите экосистему Spring.

Spring Boot, Spring MVC, Spring Web Flow, Spring Web Services, Spring Security, Spring Integration, Spring Batch, Spring Social, Spring Mobile, Spring Dynamic Modules, Spring LDAP, Spring Rich Client, Spring.NET, Spring-Flex, Spring Roo

243. Что такое Spring MVC?

Это веб-фреймворк Spring. Он позволяет создавать веб-сайты или RESTful сервисы (например, JSON/XML) и хорошо интегрируется в экосистему Spring, например, он поддерживает контроллеры и REST контроллеры в ваших Spring Boot приложениях.

244. Что такое Spring Boot?

Spring Boot — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.

245. Определите особенности Spring Boot и его роль в разработке приложений с помощью фреймворка Spring?

Если Spring Framework фокусируется на предоставлении гибкости, то Spring Boot стремится сократить длину кода и упростить разработку web-приложения. Используя конфигурацию при помощи аннотаций и стандартного кода, Spring Boot сокращает время, затрачиваемое на разработку приложений

246. Опишите процесс управления зависимостями с помощью Spring Boot.

Spring Boot решает эту проблему путём предоставления набора зависимостей, облегчая жизнь разработчикам. Например, если вы желаете использовать Spring и JPA в целях доступа к базе данных, вам достаточно просто включить в проект зависимость spring-boot-starter-data-jpa

247. Опишите процесс автоматической конфигурации на Spring Boot.

Spring Boot автоматическая настройка пытается автоматически настроить приложение Spring на основе добавленных зависимостей jar. Например, если HSQLDB находится на вашем classpath, и вы не настроили вручную никаких компонентов подключения к базе данных, то автоматически настроится база данных в памяти

248. Что такое HttpServlet?

> Сервлет является интерфейсом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ.

Хотя сервлеты могут обслуживать любые запросы, они обычно используются для расширения веб-серверов. Для таких приложений

технология Java Servlet определяет HTTP-специфичные сервлет классы.

249. Что делает Spring MVC DispatcherServlet?

Это главный контроллер в приложении Spring MVC, который обрабатывает все входящие запросы и передает их для обработки в различные методы в контроллеры

250. Что такое ViewResolver?

Интерфейс, реализуемый объектами, которые способны находить представления View по имени View Name

251. Опишите REST контроллеры.

Когда вы разрабатываете RESTFul сервисы, все немного по-другому. Ваш клиент, будь то браузер или другой веб-сервис, будет (обычно) создавать запросы JSON или XML. Клиент отправляет, скажем, запрос JSON, вы обрабатываете его, а затем отправитель ожидает возврата JSON.

Но на стороне Java (в вашей программе Spring MVC) вы не хотите иметь дело с JSON строками. Ни при получении запросов, как указано выше, ни при отправке ответов обратно клиенту. Вместо этого вы хотели бы просто иметь объекты Java, в которые Spring автоматически конвертирует JSON. i.e. DTO

Это также означает, что вам *не нужна* вся эта обработка модели и представления, которые вам приходилось делать при рендеринге HTML в ваших контроллерах. Для RESTful сервисов у вас нет библиотеки шаблонов, читающей шаблон HTML и заполняющей его данными модели, чтобы сгенерировать для вас ответ JSON.

Вместо этого вы хотите перейти непосредственно из HTTP запрос → Java объект и из Java объект → HTTP ответ.

Как вы уже догадались, это именно то, что Spring MVC обеспечивает при написании REST контроллера.

252. Что такое HttpMessageConverter?

HttpMessageConverter — это интерфейс с четырьмя методами (обратите внимание, я немного упростил интерфейс для более простого объяснения, так как он выглядит немного более продвинутым в реальной жизни).

- canRead (MediaType) → Может ли этот конвертер читать (JSON | XML | YAML | и т. д.)? Переданный здесь MediaType обычно является значением из заголовка запроса Content-Type.
- canWrite (MediaType) → Может ли этот преобразователь писать (JSON | XML | YAML | и т. д.)? Тип MediaType, переданный здесь, обычно является значением из заголовка запроса Accept.
- read(Object, InputStream, MediaType) → Читать мой Java-объект из (JSON | XML | YAML | и т. д.) InputStream
- write(Object, OutputStream, MediaType) → Записать мой Java-объект в OutputStream как (JSON | XML | YAML | и т. д.)

Короче говоря, MessageConverter должен знать, какие MediaTypes он поддерживает (например, application/json), а затем должен реализовать два метода для фактического чтения / записи в этом формате данных

253. Какие есть HttpMessageConverters?

AllEncompassingFormHttpMessageConverter

254. В чем разница между Spring MVC и Spring Boot?

Spring MVC – это полный HTTP-ориентированный MVC-фреймворк, управляемый Spring Framework и основанный на сервлетах. Spring boot - это утилита для быстрой настройки приложений, предлагающая готовую конфигурацию для создания приложений на

базе Spring. / «Нет никакой разницы, Spring Boot использует и строит приложение поверх Spring MVC.»

255. Какой тип ввода HTTP-запроса понимает Spring MVC?

Spring MVC понимает практически все, что предлагает HTTP — с помощью сторонних библиотек.

Это означает, что вы можете добавить в него тела запросов JSON, XML или HTTP (Multipart) Fileuploads, и Spring будет удобно конвертировать этот ввод в объекты Java

256. Какие HTTP-ответы может создавать Spring MVC?

Spring MVC может записывать все что угодно в HttpServletResponse — с помощью сторонних библиотек.

Будь то HTML, JSON, XML или даже тела ответов WebSocket. Более того, он берет ваши объекты Java и генерирует эти тела ответов для вас

257. В чем разница между контроллером и REST контроллером?

Контроллер по умолчанию возвращают HTML пользователям с помощью библиотеки шаблонов, если вы не добавите аннотацию @ResponseBody к определенным методам, которые также позволяют возвращать XML / JSON.

Исходный код REST контроллера показывает, что на самом деле это контроллер с добавленной аннотацией @ResponseBody. Что эквивалентно написанию контроллера с аннотацией @ResponseBody для каждого метода.

258. Как получить доступ к текущей HttpSession пользователя?

В Spring MVC контроллере или REST контроллере вы можете просто указать HttpSession в качестве аргумента метода, и

Spring автоматически вставит его (создав его, если он еще не существует).

259. Как получить доступ к HttpServletRequest?

В вашем Spring MVC контроллере или REST контроллере вы можете просто указать HttpServletRequest в качестве аргумента метода, и Spring автоматически вставит его (создавая, если он еще не существует)

260. Как читать HTTP заголовки?

Существует множество способов получить доступ к заголовкам запросов, в зависимости от того, хотите ли вы только один или карту со всеми из них. В любом случае вам нужно аннотировать их с помощью @RequestHeader

261. Как получить IP-адрес пользователя?

Это вопрос с подвохом. Существует метод с именем httpServletRequest.getRemoteAddr(), который, однако, возвращает только IP-адрес пользователя или последнего прокси-сервера, отправившего запрос, в 99,99% случаев это ваш Nginx или Apache. Следовательно, вам нужно проанализировать заголовок X-Forwarded-For для получения правильного IP-адреса. Но что произойдет, если ваше приложение, кроме того, будет работать за CDN, например CloudFront? Тогда ваш X-Forwarded-For будет выглядеть так:

X-Forwarded-For: MaybeSomeSpoofedIp, realIp, cloudFrontIp

Проблема в том, что вы не можете прочитать заголовок слева направо, поскольку пользователи могут предоставить и, следовательно, подделать свой собственный заголовок X-Forwarded-For. Вам всегда нужно идти справа налево и исключать все известные IP-адреса. В случае CloudFront это означает, что вам необходимо знать диапазоны IP-адресов CloudFront и удалить их из заголовка. Ага!

Это приводит к довольно сложному коду, разрешающему IP.

Угадайте, сколько проектов сделали это неправильно!

```
package com.marcobehler.springmvcarticle;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import javax.servlet.http.HttpServletRequest;
@RestController
public class IpController {

    private static final String[] HEADERS_TO_TRY = {
            "X-Forwarded-For",
            "Proxy-Client-IP",
            "WL-Proxy-Client-IP",
            "HTTP_X_FORWARDED_FOR",
            "HTTP_X_FORWARDED",
            "HTTP_X_CLUSTER_CLIENT_IP",
            "HTTP_CLIENT_IP",
            "HTTP_FORWARDED_FOR",
            "HTTP_FORWARDED",
            "HTTP_VIA",
            "REMOTE_ADDR"};

    @GetMapping("/ip")
    public String getClientIpAddress(HttpServletRequest request) {
        for (String header : HEADERS_TO_TRY) {
            String ip = request.getHeader(header);
            if (ip != null && ip.length() != 0 &&
!"unknown".equalsIgnoreCase(ip)) {
                return getRealClientIpAddress(ip);
            }
        }
        return request.getRemoteAddr();
    }

    /**
     * Goes through the supplied ip string (could be one or multiple).
Traverses it through the right side...
     * and removes any known ip address ranges
     *
     * @param ipString
     * @return
```

```
     */
    public String getRealClientIpAddress(String ipString) {
        String[] manyPossibleIps = ipString.split(",");

        for (int i = manyPossibleIps.length - 1; i >= 0; i--) {
            String rightMostIp = manyPossibleIps[i].trim();
            if (isKnownAddress(rightMostIp)) {
                continue; // skip this ip as it is trusted
            } else {
                return rightMostIp;
            }
        }

        return ipString;
    }

    private boolean isKnownAddress(String rightMostIp) {
        // do your check here..for cloudfront you'd need to download their ip
address ranges
        // from e.g.
http://d7uri8nf7uskq.cloudfront.net/tools/list-cloudfront-ips
        // and compare the current ip against them
        return false;
    }
}
```

262. Опишите фреймворк Django.

Свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC. Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий фреймворка от некоторых других (например, Ruby on Rails). Один из основных принципов фреймворка — DRY (англ. Don’t repeat yourself).

263. Какое программное обеспечение позволяет разрабатывать django?

Django Rest Framework (DRF) — это библиотека, которая работает со стандартными моделями Django для создания гибкого и мощного API для проекта

264. Из каких частей состоит обычный проект на Django.

Веб-приложение или проект Django состоит из отдельных приложений. Вместе они образуют полноценное веб-приложение. Каждое приложение представляет какую-то определенную функциональность или группу функциональностей. Один проект может включать множество приложений. Это позволяет выделить группу задач в отдельный модуль и разрабатывать их относительно независимо от других. Кроме того, мы можем переносить приложение из одного проекта в другой независимо от другой функциональности проекта.

При создании проекта он уже содержит несколько приложений по умолчанию.

- django.contrib.admin
- django.contrib.auth
- django.contrib.contenttypes
- django.contrib.sessions
- django.contrib.messages
- django.contrib.staticfiles

265. Что такое Отображения (views) в рамках Django.

Представления (views) — центральные персонажи Web-приложений на основе Django.

В Django используются два вида представлений:

1. *Представления-функции (view functions)*,
2. *Представления-классы (class based views)*.

Рассмотрим сначала более простой (но не менее мощный) вид - обычные функции, принимающие на входе *запрос* (объект

класса HttpRequest) и возвращающие *ответ* (объект

класса HttpResponse или ему подобных).

Все представления этого вида наследуются от класса django.views.View. Метод get здесь работает точь-в-точь как view function. При этом на каждый запрос будет создан новый экземпляр этого класса, так что вы смело можете объявлять в классе методы, которые по ходу выполнения запроса будут менять его состояние. Регистрируется представление-класс с помощью метода (метода класса) as_view.

266. Что такое модель в рамках Django.

Веб-приложения Django получают доступ и управляют данными через объекты Python, называемые моделями. Модели определяют структуру хранимых данных, включая типы полей и, возможно, их максимальный размер, значения по умолчанию, параметры списка выбора, текст справки для документации, текст меток для форм и т.д

267. Как производится вывод данных с помощью Django.

return render(request, "index.html", {"people": people})

268. Как работает Административная панель в Django.

Она использует мета-данные модели чтобы предоставить многофункциональный, готовый к использованию интерфейс для работы с содержимым сайта

269. Опишите реализацию аутентификации на Django.

Внутри locallibrary/locallibrary/settings.py:

INSTALLED_APPS = [

...

'django.contrib.auth', # Фреймворк аутентификации и моделей по умолчанию.

'django.contrib.contenttypes', # Django контент-типовая система (даёт разрешения, связанные с моделями).

```
    ....

MIDDLEWARE = [
    ...
    'django.contrib.sessions.middleware.SessionMiddleware',  # Управление сессиями между запросами
    ...
    'django.contrib.auth.middleware.AuthenticationMiddleware',  # Связывает пользователей, использующих сессии, запросами.
    ....
```

270. Опишите преимущества фреймворка Django.

Когда у вас возникает определенная мысль, трансформировать ее на языке программирования и предать ей реальную форму при помощи Django займет всего несколько минут. То, что Django находится в свободном доступе, дает возможность заметно упростить процесс веб разработки, так как разработчик может сфокусироваться на процессе дизайна и разработке функционала приложения. Таким образом, Django – это идеальный инструмент для стартапов, когда веб дизайн должен отображать концепцию и цели компании.

Быстрота: Django был разработан, чтобы помочь разработчикам создать приложение настолько быстро, на сколько это возможно. Это включает в себя формирование идеи, разработку и выпуск проекта, где Django экономит время и ресурсы на каждом из этих этапов. Таким образом, его можно назвать идеальным решением для разработчиков, для которых вопрос дедлайна стоит в приоритете.

Полная комплектация: Django работает с десятками дополнительных функций, которые заметно помогают с аутентификацией пользователя, картами сайта, администрированием содержимого, RSS и многим другим. Данные аспекты помогают осуществить каждый этап веб разработки.
Безопасность: Работая в Django, вы получаете защиту от ошибок, связанных с безопасностью и ставящих под угрозу проект. Я имею ввиду такие распространенные ошибки, как инъекции SQL, кросс-сайт подлоги, clickjacking и кросс-сайтовый скриптинг. Для эффективного использования логинов и паролей, система пользовательской аутентификации является ключом.
Масштабируемость: фреймворк Django наилучшим образом подходит для работы с самыми высокими трафиками. Следовательно, логично, что великое множество загруженных сайтов используют Django для удовлетворения требований, связанных с трафиком.
Разносторонность: менеджмент контента, научные вычислительные платформы, даже крупные организации – со всем этим можно эффективно справляться при помощи Django

271. Опишите недостатки фреймворка Django.

Использование шаблона маршрутизации с указанием URL
Django слишком монолитный
Все базируется на ORM Django
Компоненты развертываются совместно
Необходимо умение владеть всей системой для работы

272. Опишите процесс масштабирования приложения с помощью какого-либо современного фреймворка.

Фреймворки имеют разную сложность и рассчитаны на компании или подразделения разного размера. При этом большинство из них рассчитано на короткие цепочки создания ценности, когда одна кроссфункциональная команда делает продукт, поставляемый потребителям. Короткие цепочки являются естественным способом организации труда, способным быстро реагировать на изменения, в отличие от стабильных условий функционирования, которые ведут к специализации и образованию длинных цепочек из функциональных подразделений.

273. Опишите ситуации, когда для масштабирования системы, написанной с помощью фреймворка требуется реинжиниринг системы.
Когда приложение было написано без учета разбиение его в будущем на микросервисы.

274. Опишите как происходит процесс перехода с одного фреймворка на другой.
Перенос страниц, перенос контроллеров, перенос моделей.

275. Когда требуется переход с одного фреймворка на другой?
Когда ограничения изначального фреймворка начинают сильно мешать, а возможности изначального фреймворка больше не являются достаточными