QA

- **Какая разница между $this и self в PHP?**
  $this – это ссылка на сам объект, а self – на текущий класс.

- **Сколько типов данных в PHP?**
  PHP поддерживает 8 базовых типов данных.
  4 скалярных типа:
  - boolean
  - integer
  - float
  - string
  2 комплексных (составных) типа:
  - array
  - object
  2 специальных типа:
  - resource
  - NULL

- **Что можно сказать про типизацию данных в PHP?**
  PHP является языком программирования с динамической типизацией, не требующим указания типа при объявлении переменных, равно как и самого объявления переменных. Преобразования между скалярными типами зачастую осуществляются неявно без дополнительных усилий.

- **Есть ли разница между одинарными и двойными кавычками в PHP?**
  В двойных кавычках данные “парсятся”, а в одинарных – нет.

- **Что такое ассоциативный массив?**
  Массивы, индексами которых являются строки, называются ассоциативными. Индексы ассоциативных массивов называются ключами. Например:

  $price = array(“яблоки” => 10, “груши” => 15, “бананы” => 24);

- **Что такое static функция и чем она отличается от “обычной” (не static)?**
  Static принадлежит классу, а не экземпляру класса. И вызывается у класса, а не у объекта, т.е. напрямую. Объявление свойств и методов класса статическими позволяет обращаться к ним без создания экземпляра класса. Атрибут класса, объявленный статическим, не может быть доступен посредством экземпляра класса (но статический метод может быть вызван). Так как статические методы вызываются без создания экземпляра класса, то псевдопеременная $this не доступна внутри метода, объявленного статическим. Доступ к статическим свойствам класса не может быть получен через оператор ->.

  Пример. Static-члены класса доступны даже если объект этого класса не создан:

```
<?php
        class A {
            public static $static_item = 'hello';
            public static function hello() {
                echo 'hello_function';
            }
        }

        echo A::$static_item; // hello
        A::hello(); // hello_function
?>
```

- **Поддерживает ли PHP множественное наследование?**

  Нет, PHP не поддерживает множественное наследование. То есть у производного класса может быть только один родительский. PHP вводит инструментарий для повторного использования кода, называемый трейтом (traits). Трейт предназначен для уменьшения некоторых ограничений единого наследования, позволяя разработчику повторно использовать наборы методов свободно, в нескольких независимых классах и реализованных с использованием разных архитектур построения классов. Также стоит отметить, что один класс может реализовывать несколько интерфейсов.

- **Что такое конструктор?**
  Конструктор – это метод __construct(), вызываемый при создании экземпляра класса (объекта) при помощи ключевого слова new.

- **Приведи пример конструктора.**

```
<?php
    class MyClass {
        public function __construct() {
            echo 'Привет из конструктора!';
        }
    }

    $myObject = new MyClass(); // Привет из конструктора!
?>
```

- **Какая разница между require(), require_once(), include() и include_once()?**
  require() подключает в сценарий дополнительный файл, в то время как require_once() делает это только в том случае, если этот файл не был включен ранее.

  Таким образом, require_once() лучше использовать, если нужно включить файл с большим количеством функций. Тогда можно быть уверенным, что файл не будет включен многократно и не возникнет ошибка “объявление функции дублируется”.

  Отличие между require() и include() следущее: require() возвращает FATAL ERROR, если файл не найден, include() же возвращает только WARNING.

  Функция include_once() работает почти так же, как и include(), а отличия те же, что и между require() и require_once().

- **Какая разница между функциями echo и print в PHP?**
  echo может принимать и выводить любое количество аргументов, а print - только один.

```
echo $a, $b; // правильно
print $a, $b; // не правильно! используйте конкатенацию
```

- **Что такое рекурсия?**
  Рекурсия – это вызов функции из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция A вызывает функцию B, а функция B – функцию A. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

- **Преинкремент и постинкремент. В чем между ними разница?**
  Преинкремент (++$i) – сначала увеличивает, потом возвращает значение.
  Постинкремент ($i++) – сначала возвращает, потом увеличивает значение.

- **Что работает быстрее: преинкремент или постинкремент?**
  Преинкремент быстрее, т.к. постинкремент создает временную переменную, в то время как преинкремент изменяет саму переменную непосредственно.

- **В чем различие между ArrayAccess и ArrayObject?**
  ArrayAccess – это просто интерфейс, требующий определения следующих методов: offsetGet, offsetSet, offsetExists, и offsetUnset.

  ArrayObject – класс, реализующий интерфейс ArrayAccess. Удобство ArrayObject заключается в том, что к его свойствам можно получать доступ таким же образом, как к обычному массиву через оператор [], например: $dogs['sound'], или через метод offsetGet('имя свойства'), например: $dogs->offsetGet('sound').

- **Что такое сериализация?**
  Сериализация это преобразования типа данных или объекта в специальную строку, с которой можно в дальнейшем работать.

  Пример: Массив
  $array = array("first" => "quizful", "two" => 2, 3 => "three");
  $s = serialize($array);
  echo $s;

  вернет:
  a:3:{s:5:"first";s:7:"quizful";s:3:"two";i:2;i:3;s:5:"three";}

  Теперь эту строку мы можем передать удобным нас способом и вернуть обратно в массив используя функцию:

  unserialize($array);

- **Поясните разницу между HTTP методами GET и POST.**
  GET передает данные серверу используя URL, когда POST передает данные, используя тело HTTP запроса.

  Длина URL'а ограничена 1024 символами, это и будет верхним ограничением для данных, которые можно отослать GET'ом.

  POST может отправлять гораздо большие объемы данных. Лимит устанавливается веб-сервером и обычно равен около 2MB.

  Передача данных методом POST более безопасна, чем методом GET, так как секретные данные (например пароль) не отображаются напрямую в web-клиенте пользователя (в отличии от URL, который виден почти всегда).

- **Что такое MVC?**
  Model-view-controller – это схема использования шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента так, что модификация одного из компонентов оказывает минимальное воздействие на остальные.

  Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области.

- **Что за что отвечает в MVC?**
  В шаблоне MVC, как следует из названия, есть три основных компонента: Модель, Представление и Контроллер.

  Модель является “сутью” системы и отвечает за непосредственные алгоритмы, расчёты и тому подобное внутреннее устройство системы.

  Представление отвечает за отображение информации, поступающей из системы или в систему.

  Контроллер является связующим звеном между “моделью” и “представлением” системы, посредством которого и существует возможность произвести разделение между ними. Контроллер получает данные от пользователя и передаёт их в “модель”. Кроме того, он получает сообщения от модели, и передаёт их в “представление”.

- **Что такое шаблоны (паттерны) проектирования?**
  Паттерн проектирования – это общее типовое решение некоторой проблемы, многократно повторяемое в процессе проектирования архитектуры программного продукта. Они показывают отношения и взаимодействия между классами, позволяют сделать систему гибкой и легко изменяемой. За счет их правильного использования повышается коэффициент использования готовых решений.

- **Какие паттерны знаешь?**

  1. Порождающие (Creational):

  Factory Method (Фабричный метод).
  Используется для определения и поддержания отношений между объектами. Фабричные методы избавляют проектировщика от необходимости встраивать в код зависящие от приложения классы.

  Singleton (Одиночка).
  Используется для создания всего одного экземпляра класса и гарантирует, что во время работы программы не появится второй. Например, в схеме MVC, зачастую этот паттерн используется для порождения главного (фронтового) контроллера.

  2. Структурные (Structural):

  Adapter (Адаптер).
  Унифицирует классы и объекты. Используется для преобразования одного интерфейса в другой, необходимый клиенту.

  Decorator (Декоратор).
  Используется для динамического расширения функциональности объекта. Является гибкой альтернативой наследованию.

  3. Поведенческие (Behavioral):

  Strategy (Стратегия).
  Предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путем определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.

- **Что такое ООП?**
  Объектно-ориентированное программирование – это парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

- **Расскажи основные принципы ООП.**
  1. Инкапсуляция.

  Это механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает их от внешнего вмешательства или неправильного использования. Когда методы и данные объединяются таким способом, создается объект. Т.е. переменные состояния объекта скрыты от внешнего мира. Изменение состояния объекта (его переменных) возможно ТОЛЬКО с помощью его собственных методов. Можно сказать, что инкапсуляция подразумевает под собой сокрытие данных, что позволяет эти данные защитить.

  2. Наследование.

  Это процесс, посредством которого, один объект может наследовать свойства другого объекта и добавлять к ним черты, характерные только для него.

  3. Полиморфизм.

  Это свойство, которое позволяет одно и тоже имя использовать для решения нескольких технически разных задач. Проще говоря, концепцией полиморфизма является идея “один интерфейс, множество реализаций”. Это означает, что можно создать общий интерфейс для группы близких по смыслу действий.

- **Что такое виртуальный метод?**
  Виртуальный метод в объектно-ориентированном программировании – это метод класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения. Виртуальные методы – один из важнейших приёмов реализации полиморфизма.

- **Зачем нужна инкапсуляция?**
  Никто не застрахован от ошибок, а человеку тем более свойственно ошибаться. Применяя инкапсуляцию, мы, как бы, возводим купол, который защищает данные, принадлежащие объекту, от возможных ошибок, которые могут возникнуть при прямом доступе к этим данным. Кроме того, применение этого принципа очень часто помогает локализовать возможные ошибки в коде программы, а это намного упрощает процесс поиска и исправления этих ошибок.

- **Как называется способность объекта скрывать свои данные и реализацию от других объектов системы?**
  Инкапсуляция.

- **Какие механизмы в ОО языках обычно позволяют обеспечить инкапсуляцию объектов?**
  Модификаторы доступа.

- **Может ли быть конструктор виртуальным?**
  Нет, конструкторы не могут быть виртуальными.

- **Что такое класс?**
  Класс – это модель ещё не существующей сущности (объекта). Класс фактически описывает устройство объекта, являясь своего рода чертежом.

- **Что такое объект?**
  Объект – это совокупность данных и методов для их обработки. Данные и методы называются членами класса. Вообще, объектом является все то, что поддерживает инкапсуляцию.

- **Чем отличается класс от объекта?**
  Класс – это тип данных, а объект – экземпляр типа класс.

- **Чем локальные переменные отличаются от глобальных?**
  Локальные доступны только конкретной подпрограмме, глобальные – всей программе.

- **Что такое область видимости переменной?**
  Область видимости переменной – это место в программе, в котором доступно значение переменной. Каждая переменная имеет свою область видимости.

  Public. Метод/переменная доступны из любого места в коде.

  Protected. Защищённые методы или переменные доступны только внутри класса, где они были объявлены и из его производных классов.

  Private. Закрытые методы или переменные доступны только внутри класса.

- **Чем отличается процедурный подход от объектно-ориентированного?**
  Процедурный подход предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Задачи разбиваются на шаги и решаются шаг за шагом. Кроме того данные лежат отдельно от функций, для каждой новой сущности приходится писать свой набор функций с немного другими именами.

  А ООП предполагает заключение внутри одного класса, как данных, так и методов их обработки. При этом создание новой сущности не вызывает необходимости переписывать все методы, а только нужные (это называется “наследование”).

- **Какие еще есть парадигмы (модели, подходы) программирования кроме ООП?**
  Из тех, что наиболее часто встречаются: функциональная, аспектно-ориентированная и процедурная.

- **Что такое абстрактный класс?**
  Абстрактный класс в объектно-ориентированном программировании – это базовый класс, который не предполагает создания экземпляров. Абстрактные классы реализуют на практике один из принципов ООП - полиморфизм. Абстрактный класс может содержать абстрактные методы и свойства. Абстрактный метод не реализуется для класса, в котором описан, однако должен быть реализован для его неабстрактных потомков. Абстрактные классы представляют собой наиболее общие абстракции, то есть имеющие наибольший объем и наименьшее содержание.

- **Можно ли создать экземпляр абстрактного класса?**
  В PHP создать экземпляр абстрактного класса нельзя.

- **Какая разница между абстрактным классом и интерфейсом?**
  Абстрактный класс – это класс, который имеет хотя бы 1 абстрактный (не определенный) метод и обозначается как abstract. Интерфейс – такой же абстрактный класс, только в нем не может быть свойств и не определены тела у методов.

  Кроме того, что абстрактный класс наследуется (etxends), а интерфейс реализуется (implements). Вот и возникает разница между ними, что наследовать мы можем только 1 класс, а реализовать сколько угодно.

- **Зачем нужен интерфейс, если есть абстрактный класс?**
  Затем, что можно унаследоваться только от одного абстрактного класса, но реализовать множество интерфейсов. Плюс, в качестве приятного довеска, ВСЕ методы, описанные в интерфейсе, ДОЛЖНЫ быть реализованы в классе, а в абстрактном классе их нужно для этой цели специально отмечать.