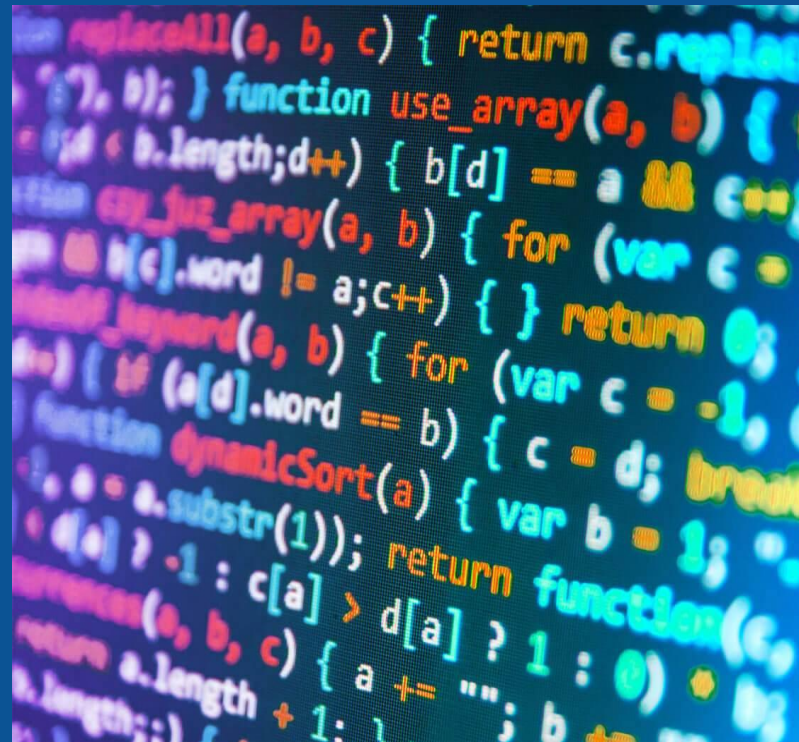


# ОСНОВЫ ЯЗЫКА PHP

Moldova State University, Nartea. N

# Установка PHP

*#основы-языка-php*



# Установка РНР

## *Введение*

Существует два способа **установки РНР**:

- ❑ Установка отдельно (минималистичный подход).
- ❑ Установка в составе веб-сервера и дополнительных инструментов.

Выбор метода зависит от потребностей разработки.

# Установка PHP

## *Отдельная установка*

*Подходит для минималистичной настройки и быстрого тестирования.*

### **Шаги установки:**

- ☐ Скачать PHP с **php.net**.
- ☐ Распаковать архив в удобное место (**C:\Program Files\php**)
- ☐ Добавить путь в системные переменные (Path)
- ☐ Проверить установку: **php -v** в командной строке.

# Установка PHP

## *Установка PHP с веб-сервером*

*Используется для комплексной разработки.*




### **Популярные сборки:**

- ❑ **XAMPP** – включает Apache, PHP, MySQL, phpMyAdmin.
- ❑ **OpenServer** – альтернатива для Windows.



# Установка PHP

## *Установка сборки XAMPP*

### **Процесс установки:**

-  Скачать и запустить установщик.
-  Выбрать компоненты (Apache, PHP, MySQL и др.).
-  Указать путь установки (C:\Program Files\xampp).

### **Запуск PHP проекта**

-  Запустить XAMPP Control Panel.
-  Включить **Apache** и проверить доступ к **http://localhost**.

# Установка PHP

## *Добавление проекта XAMPP*

- ❑ Создать папку проекта в `c:\Program Files\xampp\htdocs`.
- ❑ Разместить файлы сайта в этой папке.
- ❑ Доступ к проекту:
  - ❑ `http://localhost/имя_папки`
- ❑ Можно работать с базой данных через phpMyAdmin.

# Установка PHP

## *Использование встроенного веб-сервера PHP*

*Подходит для простых проектов и тестирования.*

- ❑ Запуск встроенного сервера:

- ❑ `cd путь_к_проекту`

- ❑ `php -S localhost:8000`

- ❑ Открыть в браузере `http://localhost:8000`

- ❑ Позволяет быстро развернуть проект без установки дополнительных компонентов.



# Установка PHP

## *Выполнение PHP-скриптов из консоли*

- ❑ Можно запускать PHP-файлы напрямую (выполнить в консоли)
  - ❑ `php <название файла>.php`
  - ❑ Например: `php index.php`
- ❑ Полезно для:
  - ❑ CLI-приложений
  - ❑ Скриптов автоматизации
  - ❑ Тестирования кода без веб-сервера

# Первая программа на PHP

*#основы-языка-php*



# Первая программа на PHP

- ❑ **PHP** — это серверный язык программирования, используемый для создания динамических веб-страниц.
- ❑ **Основное преимущество** — возможность встраивания в HTML-код.
- ❑ Сервер интерпретирует PHP-код и передает клиенту готовую HTML-страницу.

# Создание PHP-файла

- ❑ Файл должен иметь расширение `.php`
- ❑ **Для XAMPP:**
  - ❑ Разместите файл в `xampp\htdocs`.
- ❑ **Для встроенного сервера PHP:**
  - ❑ Сохраните файл в любой директории проекта.
- ❑ **Главный файл:** Обычно называется `index.php`.

# Написание PHP-кода

- ❑ Код начинается с `<?php` и заканчивается `?>`.
- ❑ Используйте `echo` для вывода данных

```
<?php
```

```
echo "Hello, world!";
```

```
echo 2 + 2;
```

- ❑ Открываем в браузере: `http://localhost/index.php`.

# Встраивание PHP в HTML

- ❑ PHP-код можно внедрять прямо в HTML.

- ❑ **Пример кода:**

```
<h1><?php echo "Привет, мир!"; ?></h1>
```

- ❑ Сервер обрабатывает PHP-код и возвращает готовый HTML.

- ❑ **Итоговой результат:**

```
<h1>Привет, мир!</h1>
```

# Обработка запроса сервером

- ❑ Браузер отправляет запрос на сервер (например, `index.php`).
- ❑ Веб-сервер передаёт запрос интерпретатору PHP.
- ❑ PHP выполняет код, формирует HTML и отправляет его обратно браузеру.
- ❑ Браузер отображает итоговую страницу.

# Переменные и константы в PHP

*#основы-языка-php*



```
function replaceAll(a, b, c) { return c.replace(
  a, b); } function use_array(a, b) {
  for ($d = 0; $d < b.length; $d++) { b[$d] == a && c++ }
} function use_fuz_array(a, b) { for (var c = 0;
  c < b[c].word != a; c++) { } return 0;
} function use_keyword(a, b) { for (var c = -1;
  c < a[d].word == b) { c = d; break }
} function dynamicSort(a) { var b = 1;
  for (var i = 0; i < a.length; i++) {
    a[i] = a.substr(1)); return function(c,
    d) { return c[a] > d[a] ? 1 : 0 }
  }
} function sort(a, b, c) { a += " "; b += " ";
  return a.length + 1; }
```



# Введение в переменные

- ❑ Переменные в PHP начинаются со знака \$.
- ❑ Должны начинаться **с буквы** или `_`, могут содержать цифры.
- ❑ Чувствительны к регистру (`$name`  $\neq$  `$Name`).
- ❑ **PHP — слабо типизированный язык**, переменная может менять тип.

# Присваивание значений

- ❑ В PHP оператор `=` используется для присваивания значения переменной.
- ❑ Переменные могут содержать:
  - ❑ числа
  - ❑ строки
  - ❑ массивы
  - ❑ другие типы данных.
- ❑ Если переменная не была объявлена, PHP выдаст предупреждение:

```
<?php
```

```
echo $undefined; // Notice: Undefined variable
```

# Проверка существования переменной

- ❑ Чтобы избежать ошибок, можно использовать `isset()`:

```
<?php  
  
if (isset($variable)) {  
  
    echo "Переменная определена!";  
  
} else {  
  
    echo "Переменная не существует."  
  
}
```

- ❑ `isset()` возвращает `true`, если переменная существует и не равна `null`.

# Константы

## *Что такое константы?*

- ❑ **Константы** — это переменные, значение которых нельзя изменить после объявления.
- ❑ Они удобны для хранения значений, которые не должны изменяться в коде (например, *математические константы, настройки*).

# Константы

## *Объявление констант с помощью define()*

- ❑ С помощью **define()**
- ❑ Функция **define()** принимает два аргумента:
  - ❑ Имя константы (строка, без \$ в начале)
  - ❑ Значение

```
<?php
```

```
define("PI", 3.14);
```

```
echo PI; // 3.14
```

# Константы

## Объявление констант с помощью *const*

- ❑ **const** используется в глобальной области видимости и внутри классов.
- ❑ Значение должно быть известно на этапе компиляции.

<?php

```
const MAX_USERS = 100;
```

```
echo MAX_USERS; // 100
```

```
const CURRENT_YEAR = date("Y"); // Нельзя использовать динамическое значение
```

# Область видимости переменных

- ❑ В PHP существуют следующие области видимости
- ❑ **Глобальная** — доступна вне функций:

```
$name = "Alice";  
  
echo $name; // OK
```

- ❑ **Локальная** — внутри функции

```
function greet() {  
  
    $message = "Hello!";  
  
    echo $message; // OK внутри функции  
  
}
```

# Использование переменных в блоках

- ❑ Переменные, объявленные внутри {} (if, while, for), доступны за пределами блока, если они не внутри функции.

```
<?php
```

```
if (true) {
```

```
    $message = "Hello!";
```

```
}
```

```
echo $message; // "Hello!"
```





# Использование переменных в блоках

- ❑ Тем не менее, **рекомендуется** заранее инициализировать переменные для читаемости и избежания ошибок.

```
<?php  
  
$message = "";  
  
if (true) {  
    $message = "Initialized inside block";  
}  
  
echo $message; // "Initialized inside block"
```

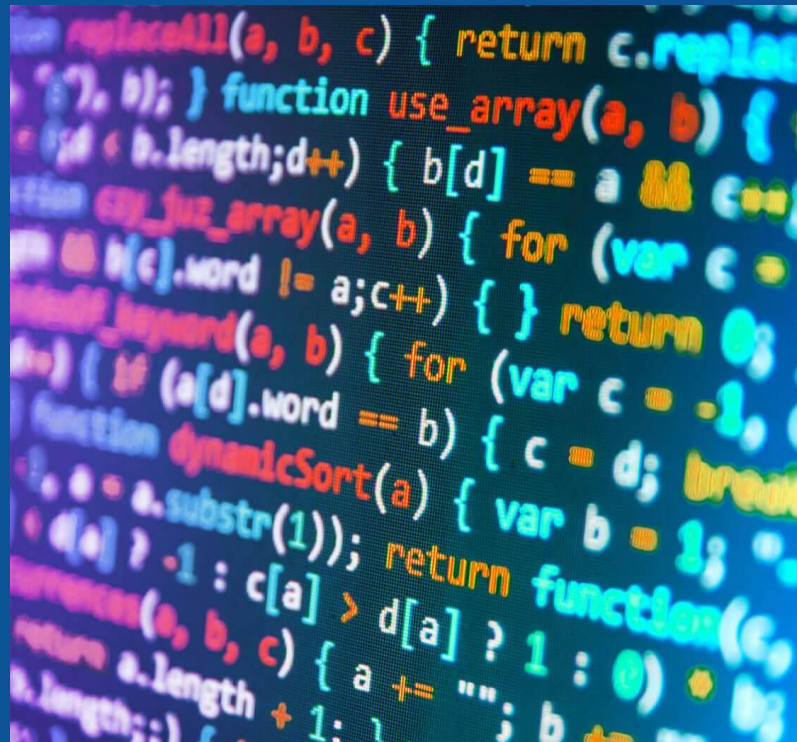
# Соглашения по именованию переменных

- ❑ camelCase для переменных: `$userName`, `$userAge`
- ❑ UPPER\_CASE для констант: `PI`, `DB_HOST`
- ❑ Выбирать понятные имена: `$totalPrice`, а не `$tp`
- ❑ Использовать английский язык в коде

|  Плохая практика            |  Хорошая практика |
|--|--|
| <pre>&lt;?php<br/><br/>\$var = "Alice"; // Слишком общее название<br/>\$a = 10; // Неинформативное имя</pre> | <pre>&lt;?php<br/><br/>\$userAge = 25;<br/>const MAX_USERS = 100;</pre>                              |

# Вывод значений на экран

*#основы-языка-php*



# Способы вывода данных

- ❑ В PHP есть несколько способов вывести данные в браузер:
  - ❑ **echo** (самый быстрый способ)
  - ❑ **print** (возвращает значение)

# Оператор echo

- ❑ Самый популярный способ вывода
- ❑ Не требует скобок, может выводить несколько значений через запятую.

```
echo "Hello, world!";
```

```
echo 2 + 2;
```

```
echo "Привет", " мир!";
```

- ❑ Результат:

```
Hello, world!
```

```
4
```

```
Привет мир!
```

# Оператор `print`

- ❑ **Работает медленнее `echo`**
- ❑ Всегда возвращает **1** (можно использовать в выражениях)
- ❑ Поддерживает только один аргумент

```
print "Hello, world!";
```

```
print 2 + 2;
```

- ❑ Также можно использовать скобки

```
print("Hello, world!");
```

# Конкатенация строк

- ❏ Оператор `.` (точка) используется для объединения строк.

```
$name = "Alice";
```

```
echo "Hello, " . $name;
```

- ❏ Результат: **"Hello, Alice"**

# Интерполяция строк

- ❑ В двойных кавычках (" "), переменные встраиваются в строку.

```
$name = "Alice";
```

```
echo "Hello, {$name}";
```

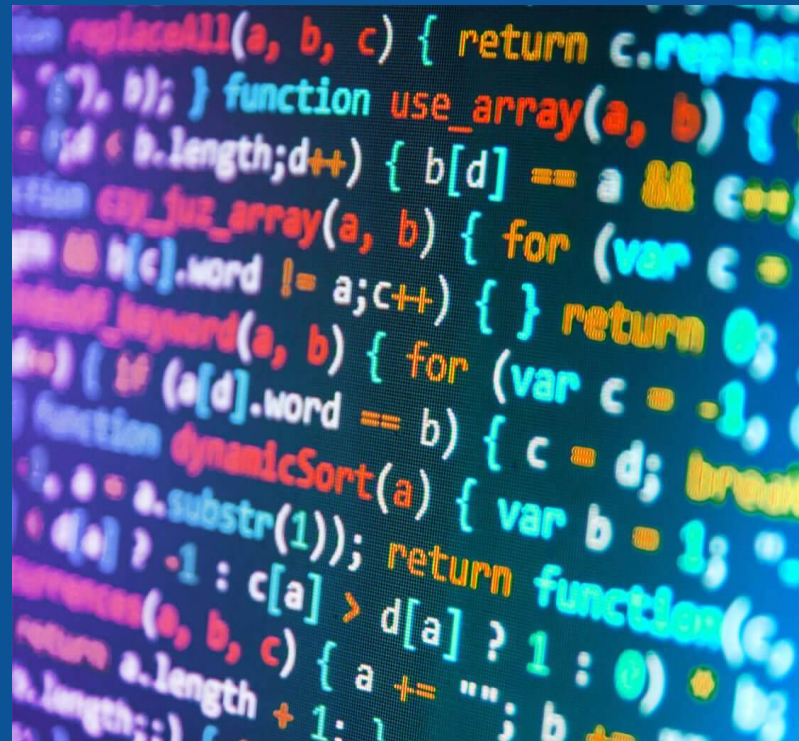
- ❑ Результат: "Hello, Alice"

- ❑ Нельзя использовать с одинарными кавычками



# Рекомендации по написанию кода в PHP

*#основы-языка-php*



```
function replaceAll(a, b, c) { return c.replace(
    a, b); } function use_array(a, b) {
    for ($d = 0; $d < b.length; $d++) { b[$d] == a && c++ }
} function use_array(a, b) { for (var c = 0;
    c < b.length; c++) { if (b[c].word != a; c++) { } } return c; }
function use_array(a, b) { for (var c = 0; c < b.length; c++) {
    if (b[c].word == a) { c++; } } return c; }
function dynamicSort(a) { var b = 1;
    for (var i = 0; i < a.length; i++) {
        a[i] = a.substr(1)); return function(c,
        d) { return c[a] > d[a] ? 1 : 0; }
    }
} function dynamicSort(a, b, c) { a += " "; b += " ";
    return a.length + 1; }
```

# Интерполяция строк

- ❑ Написание качественного кода — это не только знание синтаксиса, но и следование лучшим практикам.
- ❑ **Хороший код:**
  - ❑ Читаем и понятен;
  - ❑ Поддерживаем в долгосрочной перспективе;
  - ❑ Соответствует стандартам.

# Рекомендации по написанию PHP-кода

## *Использование PHP-тегов*

- ❑ Код всегда начинается с `<?php`
- ❑ Закрывающий тег `?>` не нужен, если файл состоит только из PHP-кода:

```
<?php
```

```
echo "Пример без закрывающего тега";
```

- ❑ Если PHP встраивается внутрь HTML, то закрывающий тэг **обязателен**

```
<div>
```

```
<?php echo "Привет, мир!"; ?>
```

```
</div>
```



# Рекомендации по написанию PHP-кода

## *Разделение PHP и HTML*

- ❑ Смешивание **PHP** и **HTML** может значительно ухудшить читаемость и усложнить поддержку кода.
- ❑ Логика должна быть отделена от представления, а PHP-код использоваться только там, где это действительно необходимо.

# Рекомендации по написанию PHP-кода

## Разделение PHP и HTML

 Плохая практика	 Хорошая практика
<ul style="list-style-type: none"><li>- смешивание PHP и HTML</li></ul>	<ul style="list-style-type: none"><li>- минимизация PHP в HTML</li></ul>
<pre>&lt;div&gt;   &lt;?php   if (\$isLoggedIn) {     echo "&lt;p&gt;Добро пожаловать, \$username!&lt;/p&gt;";   } else {     echo "&lt;p&gt;Пожалуйста, войдите.&lt;/p&gt;";   }   ?&gt; &lt;/div&gt;</pre>	<pre>&lt;div&gt;   &lt;?php if (\$isLoggedIn): ?&gt;     &lt;p&gt;Добро пожаловать, &lt;?php echo \$userName; ?&gt;!&lt;/p&gt;   &lt;?php else: ?&gt;     &lt;p&gt;Пожалуйста, войдите.&lt;/p&gt;   &lt;?php endif; ?&gt; &lt;/div&gt;</pre>
<b>Проблемы:</b> <ul style="list-style-type: none"><li><input type="checkbox"/> HTML перемешан с PHP-кодом.</li><li><input type="checkbox"/> Сложно читать и изменять</li></ul>	<b>Преимущества:</b> <ul style="list-style-type: none"><li><input type="checkbox"/> Легко читать и редактировать HTML.</li><li><input type="checkbox"/> Код остается чистым.</li></ul>

# Рекомендации по написанию PHP-кода

## *Разделение PHP и HTML. Использование циклов*

- ❑ PHP И HTML можно разделять при использовании циклов

```
<?php
$users = ["Alice", "Bob", "Charlie"];
?>
<ul>
    <?php foreach ($users as $user): ?>
        <li><?= htmlspecialchars($user) ?></li>
    <?php endforeach; ?>
</ul>
```

# Рекомендации по написанию PHP-кода

## *Короткий синтаксис echo*

- ❑ В PHP существует сокращенная форма echo, которая записывается как `<?= $переменная ?>`.
- ❑ Эквивалентно `<?php echo $переменная; ?>`, но занимает меньше места.

Стандартная форма	Короткий синтаксис
<code>&lt;?php echo "Привет, мир!"; ?&gt;</code>	<code>&lt;?= "Привет, мир!" ?&gt;</code>

# Рекомендации по написанию PHP-кода

## *Короткий синтаксис echo*

- ❑ Этот синтаксис **удобен для вставки переменных** непосредственно в HTML и часто используется для упрощения шаблонов.
- ❑ Этот подход, хотя и упрощает код, **не рекомендуется для использования в крупных проектах из-за возможных проблем с совместимостью**, так как некоторые серверы могут его не поддерживать.
- ❑ Поэтому предпочтительнее придерживаться стандартного синтаксиса  

```
<?php echo $variable; ?>
```



# Рекомендации по написанию PHP-кода

## *Стандарты PSR*

- ❑ **PSR (PHP Standard Recommendation)** — это набор стандартов кодирования, разработанных PHP-FIG (PHP Framework Interoperability Group).
- ❑ **Они обеспечивают:**
  - ❑ единый стиль кодирования
  - ❑ совместимость между разными проектами и фреймворками.

# Рекомендации по написанию PHP-кода

## *Стандарты PSR*

- ❑ На сегодняшний день PSR включает множество стандартов, таких как **PSR-1**, **PSR-2**, **PSR-3**, **PSR-4** и другие, каждый из которых охватывает определенный аспект разработки.
- ❑ Рекомендуется начать с двух основных стандартов:
  - ❑ **PSR-1: Basic Coding Standard.** Этот стандарт описывает основные правила написания кода на PHP.
  - ❑ **PSR-12: Extended Coding Style.** Данный стандарт определяет стиль написания кода, дополняя стандарт PSR-1.

# Let's go Further...

- В этой презентации были рассмотрены лишь основные аспекты темы.
- Более подробная информация доступна в официальных материалах курса.