4. Базовые конструкции PHP, процедурный подход.

Язык программирования

- Интерпретатор или компилятор? Как настраивается?
- В каких режимах можно выполнить программу?
- Есть ли интерактивная оболочка (REPL)?

- Какие подходы к программированию поддерживает?
- Какая типизация реализована?

Режимы запуска кода PHP CLI

- Выполнение скрипта из файла

```
php -f script.php
php script.php
```

- Выполнение кода из командой строки

```
php -r "echo date('Y-m-d H:i:s');"
php -r "for ($i = 1; $i <= 5; $i++) { echo $i . ' '; }"
```

- Выполнение кода из stdin

```
echo "print_r(get_defined_constants());" | php
```

Режимы запуска кода PHP CLI

- Интерактивный (REPL-оболочка) `php -a`
  - Автодополнение функций, констант, имен классов, переменных, статических методов и констант классов
  - История введенных команд
  - Можно подключить пейджер `#cli.pager=less`
  - Выход - команда `exit`
- Продвинутая оболочка и дебаггер PsySH
  - Вычисление значений
  - Подключение внешнего редактора
  - Вывод переменных, функций, классов, объектов
  - Вывод содержимого объектов, методов и функций
  - Встроенная документация по функциям, объектам, классам, методам, константам

Модульность языка

- Базовые конструкции (`echo, list, require, class, interface, …`) языка реализованы в ядре интерпретатора
- Функции, предопределенные константы и классы реализованы в модулях - расширениях.
- Расширения могут быть в составе дистрибутива или ставиться отдельно.
- Список подключенных расширений: `php -m`

Настройки

- Настройки интерпретатора PHP и расширений в конфигурационном файле php.ini
- Путь к используемому php.ini: `php --ini`
- Задать путь к нужном php.ini: `php -c`

Общая характеристика PHP

- Какие подходы к программированию поддерживает?
- Какая типизация реализована?

Парадигмы программирования в PHP

- **Императивный процедурный**
- **Функциональные возможности**
- Поддержка ООП на классах (а-ля Java)

Типизация в PHP

- Динамическая и по умолчанию слабая типизация.

```php
<?php
function add(int $a, int $b): int {
    return $a + $b;
}

echo add("1", "2"); // Выведет 3, так как строки автоматически преобразуются в числа
```

- Можно включить режим строгой типизации.
  В начале файла: `declare(strict_types=1)`

```php
<?php
declare(strict_types=1);

function add(int $a, int $b): int {
    return $a + $b;
}

// echo add("1", "2"); // TypeError: Argument 1 must be of type int, string given
```

Строгая типизация в PHP

- Влияет только на скалярные типы (int, float, string, bool)
- Работает только там, где указаны типы параметров или возвращаемых значений
- Директива `declare(strict_types=1)` действует только внутри файла, где она определена
- При возникновении ошибки несоответствия типов выбрасывается исключение `TypeError`

Процедурный (императивный) стиль

Структурный подход

- Переменные, оператор присваивания
- Последовательность, условия, циклы
- Подпрограммы, структурное кодирование
- Проектирование сверху вниз

В общем все похоже на другие универсальные скриптовые языки, со своей спецификой

Переменные и константы

Переменные

- Начинаются с $ (наследство Perl)
- Заранее не объявляются, создаются при инициализации
- Удаляется переменная функцией `unset()`
- Имена зависят от регистра
- Оператор присваивания возвращает значение, которое было присвоено переменной левого операнда

```
$a = $b = 10;

echo $a; // Выведет: 10
echo $b; // Выведет: 10
```

```
$sum = ($a = 5) + ($b = 10);

echo $sum; // Выведет: 15
echo $a;   // Выведет: 5
echo $b;   // Выведет: 10
```

```
while ($line = fgets($file)) {
    echo $line;
}
```

Ссылки

- Переменные ссылаются на свои значения.
- Ссылки - это средство доступа к содержимому одной переменной под разными именами.
- Ссылки создаются оператором &

```
$a =1; $b = &$a;
```

Области видимости

Область видимости - контекст, в котором определили переменную

- **Локальная**. Переменные, объявленные внутри функции, доступны только внутри этой функции.
- **Глобальная**. Переменные, объявленные вне всех функций и классов, доступны в любом месте скрипта. Внутри функций доступны через `global`.
- **Статическая**. Переменные, объявленные как static внутри функции, сохраняют свое значение между вызовами функции, но остаются локальными для этой функции.
- **Область видимости класса**. Переменные, объявленные внутри класса (свойства класса) доступны через экземпляр класса или в методах класса.
- **Суперглобальные переменные** доступны в любой области видимости (`$_GET, $_POST, $_SESSION, $_COOKIE, $_SERVER, $_FILES, $_REQUEST, $_ENV`)

Константы

- Имя констант без знака $
- Список доступных констант возвращает функция `get_defined_constants()`
- Константы создаются функцией `define()` или ключевым словом `const`

```php
<?php
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASS', 'password');
define('DB_NAME', 'my_database');

// Подключение к базе данных
$conn = new mysqli(DB_HOST, DB_USER, DB_PASS, DB_NAME);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

Константы

- Через const константы объявляются в глобальной области видимости или внутри класса/интерфейса

```php
<?php
const EXAMPLE = 'Value'; // Без кавычек
define('EXAMPLE', 'Value'); // С кавычками
?>
```

```php
<?php
class MyClass {
    const CONSTANT = 'Class Constant';
    // define('CONSTANT', 'Class Constant'); // Ошибка: Cannot use define() inside a class
}

echo MyClass::CONSTANT; // Выведет: Class Constant
?>
```

Типы переменных

- Базовые - `integer, float, boolean, string, null`
- Составные - `array, object, resource, enum`
- Комбинированные `enum`
- Псевдотипы - `callable, void, iterable, mixed, never`

Проверка типов

- Определение типа - функция `gettype()`. Возвращает строку
- Проверка типа - функции `is_bool()`, `is_int()`, `is_float()`, `is_string()`, `is_array()`, `is_object()`, `is_null()`

Преобразование типов

- Явное преобразование типа - конструкции `(bool)`, `(int)`, `(float)`, `(string)`, `(array)`, `(object)` **или** специальные функции

```php
<?php
$var = "123";
$intVar = (int) $var; // Используем оператор (int)

echo gettype($intVar); // Выведет: integer
echo $intVar;          // Выведет: 123

// Альтернативный способ с использованием функции intval()
$intVar2 = intval($var);
echo gettype($intVar2); // Выведет: integer
```

Преобразование типов

Неявное преобразование типов происходит при:

- операциях сравнения
- математических операциях
- работе с массивами
- конкатенации строк (оператор .)

```php
<?php
$a = "10"; // Строка
$b = 5;    // Целое число

echo $a + $b; // Выведет: 15

// PHP преобразует строку "10" в число 10 перед выполнением сложения.
?>
```

Числовые типы

При записи чисел можно использовать подчеркивание для визуального разделения разрядов (PHP 7.4).

Подчеркивания игнорируются интерпретатором.

```
$a = 100_123; // $a=100123;
```

Тип boolean

Правила неявного преобразования к boolean:

- `false`, `null`, 0 и пустая строка - ложь
- `true` и любое ненулевое выражение - истина

Тип string

- Строка = массив символов
- В "…" поддерживается интерполяция. Экранирование через \
- В '…' интерполяции нет.
- `команды` команды выполняются оболочкой, результат из stdout возвращается скрипту.
- Встроенный (многострочный) документ <<< метка...метка;
- Конкатенация оператором "точка" (.)

Тип array

- Массивы ассоциативные, можно комбинировать числовые и строковые индексы в одном массиве.
- Добавление в массив оператором `[]`, удаление элемента функцией `unset()`.
- Похожи на массивы в JavaScript и списки в Python, но с добавлением ассоциативной функциональности.

Применение массивов

- Хранение и обработка разнотипных данных
- Хранилище конфигурационных параметров
- Способ передать в функцию множество разнородных данных одним параметром

Сравнения и условия

Операторы сравнения

- ==, !=, <, >, <=, >= сравнивают значения, но не типы (идет неявное преобразование типов). Возвращает boolean
- Если один из операндов логический, то второй приводится к логическому типу.
- Если один из операндов числовой, то сравнение всегда выполняется в числовом контексте.

Операторы эквивалентности

- `===`, `!==` сравнивают значения и типы. Возвращает boolean

Помогает избежать неожиданных результатов, связанных с автоматическим преобразованием типов при обычном сравнении.

- $x <=> $y (PHP 7.0)

При равенстве $x и $y возвращает 0, если $x>$y, то возвращает 1, если $x<$y, то возвращает -1

Логические операторы

Базовый синтаксис

```
if (условие) {
   выражения;
} elseif (условие) {
   выражения;
} else {
   выражения;
}
```

Альтернативный синтаксис

```
if (условие):
   выражения;
elseif (условие):
   выражения;
else:
   выражения;
endif
```

Лучшая читаемость в шаблонах,
четкое разделение PHP-кода и HTML

Логические операторы

- Символьные операторы

  `&&`, `||`, `!`

- Словесные аналоги - имеют **меньший приоритет**

  `and`, `or`, `not`

```php
<?php

$content = file_get_contents('example.txt') or exit('Ошибка');
var_dump($content); // Содержимое файла example.txt

$content = file_get_contents('example.txt') || exit('Ошибка');
var_dump($content); // bool(true)

($content = file_get_contents('example.txt')) || exit('Ошибка');
var_dump($content); // Содержимое файла example.txt
```

Логические операторы

- Тернарный оператор

```
$x = $x < 0 ? -$x : $x;
$x = isset($x) ?: 1; //проверка существования переменной
```

- Оператор объединения с null

```
$x = $x ?? 1;    $x ??= 1;
```

Логические операторы

```
switch (выражение)
{
  case значение1:
    выражения;
    break;
  case значение2:
    выражения;
    break;
  default:
    выражения;
}
```

Сравнение оператором ==

```
$result = match ($expression) {
    $pattern1 => $result1,
    $pattern2 => $result2,
    ...
};
```

(для PHP 8)

Сравнение оператором ===

Циклы

Циклы

- while, do … while
- `for`
- `foreach` - итерирование массива

```php
<?php

$numbers = [1, 2, 3, 4, 5];

foreach ($numbers as $number) {
    echo "$number ";
}

echo PHP_EOL;

// ------------------------------------------------------
$fruits = ["apple" => "яблоко", "banana" => "банан", "orange" => "апельсин"];

foreach ($fruits as $key => $value) {
    echo "Ключ: $key, Значение: $value\n";
}


// ------------------------------------------------------
foreach ($numbers as $index => $number) {
    echo "$index : $number \n";
}
```

Функции

Функции

- Способы задания: обычные, анонимные, стрелочные
- Имена функций не зависят от регистра
- Можно задавать параметры функций по умолчанию, допускается переменное количество аргументов
- По умолчанию параметры передаются в функции по значению. Можно передать по ссылке (префикс &)
- Type hinting - можно указать типы аргументов и возвращаемого значения. Соответствие типов будет требоваться в режиме строгой типизации
- Поддерживаются именованные аргументы (PHP 8.0)

Функции

- Обычную функцию можно вызвать до ее определения в коде, анонимную - нет
- Все функции имеют глобальную область видимости. Вложенные функции после вызова внешней функции становятся глобальными
- Контекст обычных и анонимных функций замкнут: свои аргументы и свои локальные переменные.
- Для доступа к глобальным переменным из функции нужно указать модификатор `global`, при этом создается новая ссылка на значение глобальной переменной.

Функции обратного вызова

- Функции можно передавать в качестве параметров в другие функции (тип `callable`)

```
array_map(?callable $callback, array $array,
                  array ...$arrays): array

function cube(int | float $n): int|float
{
    return $n * $n * $n;
}

$arr = [1, 2, 3, 4, 5];
$cub = array_map('cube', $arr);
```

Замыкания

- Замыкание - анонимная функция, запоминающая состояние окружения в момент своего создания (захватывающая внешние переменные)
- Переменные, которые должны войти в замыкание, задаются в `use()`
- Основное назначение - замена глобальных переменных
  - Внутри замыкания нельзя изменить переданную в него переменную
  - Никакие изменения глобальной переменной не смогут повлиять на значение, переданное через замыкание
- Стрелочные функции захватывают внешние переменные автоматически (`use()` не нужен)

Модульная структура программ

Включение файлов

- `include`
- `include_once`
- `require`
- **`require_once`**
  - если файла нет, то будет ошибка
  - Если файл уже был загружен ранее, он не выполнится повторно, но его код будет доступен

Магическая константа `__DIR__` - каталог, в котором находится запущенный скрипт.

Включение файлов

- При включении файла он полностью выполняется.
- Экспортировать только определенные функции из подключаемого файла нельзя.
- В подключаемом файле не должно быть исполняемого кода, только определения функций.
- Имена переменных, констант, функций, классов в разных подключаемых файлах должны быть уникальными.

Все нужные файлы нужно подключать с помощью единственного файла, который формируется с помощью утилиты composer.
Все подключаемые файлы перечисляются в секции autoload в composer.json

Проблема именования

Код всех файлов проекта находится в одном пространстве, поэтому имена переменных, констант, функций и классов в разных файлах должны быть разными.
Можно добавлять длинные префиксы к отдельным названиям, но это неудобно.

Для изоляции кода используются ***пространства имен***

Аналоги:

- Каталог в файловой системе - в разных каталогах файлы могут иметь одни и те же имена.
- Код города в телефонном номере
- Страны, города и улицы в почтовом адресе

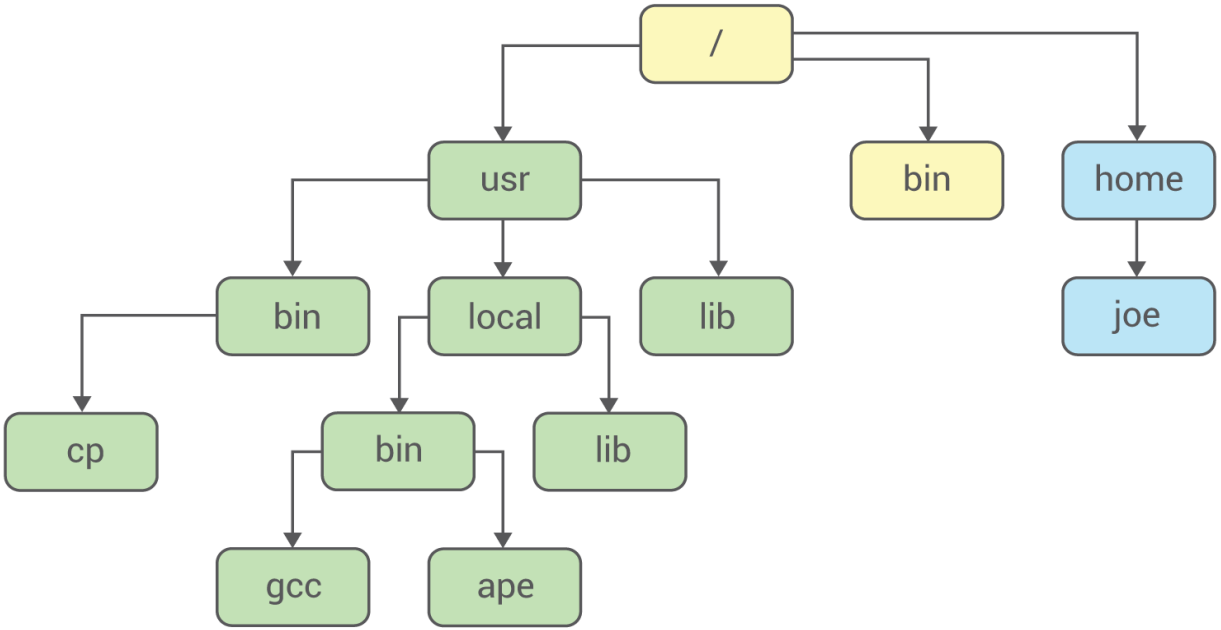
Пространства имен

Пространство имен - это обладающий именем фрагмент программы, содержащий в себе функции, переменные, константы и другие именованные элементы языка.

- Действие пространства имен распространяется только на классы, интерфейсы, функции и константы.
- Имя пространства имен не связано с именем файла.

Иерархия пространств имен

- Глобальное пространство имен \, в котором классы, функции, константы создаются по умолчанию.
- Понятие текущего пространства имен
- Абсолютные и относительные пути к пространствам имен (аналогично путям в файловой системе)



Импорт сущностей из других пространств имен

Ключевое слово `use` позволяет импортировать функции и константы из другого пространства имен.

```php
<?php
// Глобальное пространство имен
function sayHello() { echo "Global\n"; }
const APP_NAME = "Global App";

// Пространство MyNamespace
namespace MyNamespace;
function sayHello() { echo "MyNamespace\n"; }
const APP_NAME = "MyNamespace App";

// Импорт из глобального пространства
use function \sayHello as globalSayHello;
use const \APP_NAME as GLOBAL_APP_NAME;

globalSayHello(); // Global
echo GLOBAL_APP_NAME . "\n"; // Global App
sayHello(); // MyNamespace
echo APP_NAME . "\n"; // MyNamespace App
```