# ГЛАВА 1

# Переменные

Основой гибкого приложения является вариативность — способность программы служить нескольким целям в различных контекстах. Переменные — это общий инструмент для достижения такой гибкости в любом языке программирования. Такие именованные объекты ссылаются на конкретное значение, используемое программой. Это может быть число, обычная строка или даже более сложный объект со своими свойствами и методами. Суть в том, что переменные позволяют программе (и разработчику) ссылаться на эти значения и передавать их из одной части программы в другую.

Переменные не обязательно должны быть установлены по умолчанию — вполне допустимо определить пустую переменную, не присваивая ей никакого значения. Представьте стоящую на полке коробку, в которую можно положить подарок на Рождество. Вы можете легко найти эту коробку — переменную, но поскольку внутри нее ничего нет, мало что с ней можно сделать.

Предположим, что переменная называется `$giftbox`. Если бы вы попытались проверить значение этой переменной прямо сейчас, она оказалась бы пустой, так как еще не была задана. На самом деле `empty($giftbox)` вернет `true`, а `isset($giftbox)` вернет `false`. Переменная пуста и еще не определена.

Важно помнить, что любая переменная, которая не была явно определена (или задана), будет восприниматься PHP как empty(). Фактически определенная (или заданная) переменная может быть пустой или непустой в зависимости от ее значения, так как любое реальное значение, которое оценивается как false, будет считаться пустым.

В широком смысле языки программирования могут быть как строго (сильно), так и нестрого (слабо) типизированными. Строго типизированный язык требует от программиста явного указания типов всех переменных, параметров и возвращаемых функцией значений. Это обеспечивает соответствие каждого

значения ожидаемому типу. В нестрого типизированных языках, таких как PHP, значения типизируются во время исполнения программы, то есть динамически. Например, разработчики могут хранить целое число (предположим, `42`) в переменной, а затем использовать ее как строку в другом месте (скажем, `"42"`), и PHP автоматически преобразует эту переменную из целого числа в строку в нужный момент.

Главное преимущество слабой типизации заключается в простоте и удобстве разработки. Программисту не нужно заботиться о типах данных на этапе написания кода, поскольку эту задачу на себя берет интерпретатор. Основным же недостатком является то, что не всегда понятно, как будут обрабатываться определенные значения, когда интерпретатор переводит их из одного типа в другой. В табл. 1.1 показаны различные выражения, которые, начиная с PHP 8.0, оцениваются как «пустые» независимо от их базового типа.

**Таблица 1.1.** Пустые выражения в PHP

| Выражение | empty($x) |
|---|---|
| `$x = ""` | `true` |
| `$x = null` | `true` |
| `$x = []` | `true` |
| `$x = false` | `true` |
| `$x = 0` | `true` |
| `$x = "0"` | `true` |

Обратите внимание, что некоторые из этих выражений не являются по-настоящему пустыми, но PHP воспринимает их как таковые. Они считаются ложными, поскольку рассматриваются эквивалентными `false`, хотя они не идентичны `false`. Поэтому важно явно проверять ожидаемые значения типа `null`, `false` или `0` в приложении, а не полагаться на языковые конструкции вроде `empty()`, которые сделают это за вас. В таких случаях вы можете проверить, является ли переменная пустой, и явно сравнить ее с известным, фиксированным значением[1].

Рецепты в этой главе посвящены основам определения и использования переменных в PHP, а также управления ими.

[1] Операторы сравнения рассматриваются в рецепте 2.3, где приводятся как пример, так и подробное обсуждение проверок на равенство.

# 1.1. Определение констант

## Задача

Вы хотите определить в своей программе конкретную переменную с фиксированным значением, которое не может быть изменено никаким другим кодом.

## Решение

В следующем блоке кода функция `define()` используется для явного определения значения константы с глобальной областью видимости, которое не может быть изменено другим кодом:

```
if (!defined('MY_CONSTANT')) {
    define('MY_CONSTANT', 5);
}
```

В качестве альтернативного подхода следующий блок кода использует директиву `const` внутри класса для определения константы, видимой только в пределах этого класса[1]:

```
class MyClass
{
    const MY_CONSTANT = 5;
}
```

## Обсуждение

Если константа определена в приложении, функция `defined()` вернет `true` и сообщит, что вы можете обращаться к ней непосредственно в коде. Если константа еще не определена, интерпретатор PHP проанализирует, что вы делаете, и вместо этого преобразует ссылку на константу в строковый литерал.

Необязательно обозначать имена констант заглавными буквами. Тем не менее это правило, определенное в стандарте PHP Standard Recommendation 1 (PSR-1, https://oreil.ly/_rNMe), опубликованном PHP Framework Interoperability Group (PHP-FIG, https://oreil.ly/JHj-l), рекомендуется соблюдать.

[1] Подробнее о классах и объектах читайте в главе 8.

Например, в следующей строке кода переменной `$x` присвоится значение `MY_CONSTANT` только в том случае, если константа определена. До версии PHP 8.0 неопределенная константа могла привести к тому, что `$x` содержала бы строку `"MY_CONSTANT"`:

```
$x = MY_CONSTANT;
```

Если ожидаемое значение `MY_CONSTANT` не является строкой, то ответная реакция PHP на строковый литерал, вероятно, приведет к неожиданным побочным эффектам в вашем приложении. Интерпретатор не обязательно «упадет», но наличие `"MY_CONSTANT"` там, где ожидается целое число, вызовет проблемы. Начиная с PHP 8.0, обращение к еще неопределенной константе приводит к фатальной ошибке.

Наш пример демонстрирует два способа определения констант: `define()` и `const`. Первый создаст глобальную константу, которая будет доступна в любом месте вашего приложения, используя только имя самой константы. Второй привязывает константу к классу, в котором она определяется. В данном случае вместо ссылки на `MY_CONSTANT`, как в первом решении, константа будет упоминаться как `MyClass::MY_CONSTANT`.

PHP определяет несколько констант по умолчанию (https://oreil.ly/zQ40o), которые не могут быть перезаписаны пользовательским кодом. Константы в целом фиксированы и не могут быть модифицированы или заменены, поэтому всегда проверяйте, не определена ли константа, прежде чем пытаться ее задать. Попытки переопределить константу приведут к появлению предупреждения. Дополнительные сведения о работе с ошибками и уведомлениями см. в главе 12.

Константы класса по умолчанию общедоступны, то есть любой код в приложении, который обращается к `MyClass`, может ссылаться и на его общедоступные константы. Однако, начиная с версии PHP 7.1.0, можно применить модификатор видимости к константе класса и сделать ее закрытой для экземпляров класса.

## Читайте также

Документация по константам в PHP (https://oreil.ly/9WBhy), `defined()` (https://oreil.ly/jmiau), `define()` (https://oreil.ly/9iON9) и константам классов (https://oreil.ly/ggaCv).

# 1.2. Создание переменных переменных

## Задача

Вы хотите динамически ссылаться на определенную переменную, не зная заранее, какая из нескольких связанных переменных понадобится программе.

## Решение

В синтаксисе PHP имя переменной, на которую вы хотите сослаться, всегда предваряется символом `$`. Вы также можете сделать имя переменной само по себе переменной. Следующая программа выведет `#f00`, используя переменную переменной:

```
$red = '#f00';
$color = 'red';

echo $$color;
```

## Обсуждение

Когда PHP интерпретирует ваш код, он воспринимает символ `$` в начале строки как идентификатор переменной, а следующий за ним текст — как имя этой переменной. В примере выше текст после `$` является переменной. PHP будет оценивать переменные переменных справа налево, передавая результат одного вычисления в качестве имени, используемого для следующего вычисления, прежде чем выводить какие-либо данные на экран.

Говоря иначе, пример 1.1 показывает две функционально равнозначные строки кода, за исключением того, что во второй используются фигурные скобки, чтобы явно определить код, который считывается первым.

**Пример 1.1.** Определение переменных переменных

```
$$color;
${$color};
```

Крайняя правая переменная `$color` сначала расценивается как литерал `"red"`, это, в свою очередь, означает, что `$$color` и `$red` в конечном счете ссылаются на одно и то же значение. Введение фигурных скобок в качестве явных разделителей означает еще более сложное применение.

В примере 1.2 предполагается, что приложение хочет провести A/B-тестирование заголовка для SEO-оптимизации. Маркетинговая команда предоставила два варианта, и разработчики хотят показывать разные заголовки разным посетителям,

но при этом выводить тот же заголовок, когда пользователь возвращается на сайт. Вы можете сделать это, используя IP-адрес посетителя и создав переменную, которая выбирает заголовок на основе IP-адреса.

**Пример 1.2.** A/B-тестирование заголовков

```
$headline0 = 'Десять советов по написанию отличных заголовков';
$headline1 = 'Пошаговый подход к написанию ярких заголовков';

echo ${'headline' . (crc32($_SERVER['REMOTE_ADDR']) % 2)};
```

Функция `crc32()` из предыдущего примера — это удобная утилита, которая вычисляет 32-битную контрольную сумму заданной строки, детерминированно превращая строку в целое число. Оператор `%` выполняет операцию деления по модулю над итоговым целым числом, возвращая `0`, если контрольная сумма четная, и `1`, если нечетная. Затем результат конкатенируется с заголовком строки `headline` в вашей динамической переменной, чтобы функция могла выбрать тот или иной заголовок.

Массив $_SERVER — это определяемая системой суперглобальная переменная (https://oreil.ly/DtQV-), содержащая полезную информацию о сервере, на котором запущен ваш код, и о входящем запросе, который запустил PHP в первую очередь. Точное содержимое этого массива будет отличаться от сервера к серверу, особенно в зависимости от того, использовали ли вы NGINX или Apache HTTP Server (или другой веб-сервер) с PHP, но обычно он содержит такую полезную информацию, как заголовки запроса, пути запроса и имя файла текущего выполняющегося скрипта.

В примере 1.3 построчно представлено применение функции `crc32()`, чтобы проиллюстрировать, как связанное с пользователем значение, например IP-адрес, может быть задействовано для детерминированной идентификации заголовка, служащего в целях SEO.

**Пример 1.3.** Проверка контрольных сумм по IP-адресам посетителей

```
$_SERVER['REMOTE_ADDR'] = '127.0.0.1'; ❶
crc32('127.0.0.1') = 3619153832; ❷
3619153832 % 2 = 0; ❸
'headline' . 0 = 'headline0' ❹
${'headline0'} = 'Десять советов по написанию отличных заголовков'; ❺
```

❶ IP-адрес извлекается из суперглобальной переменной `$_SERVER`. Обратите внимание, что ключ `REMOTE_ADDR` будет присутствовать только при использовании PHP с веб-сервера, а не через CLI.

❷ Функция `crc32()` преобразует строку с IP-адресом в целочисленную контрольную сумму.

❸ Оператор деления по модулю (%) определяет, является ли контрольная сумма четной или нечетной.

❹ Результат деления добавляется к заголовку `headline`.

❺ Полученная строка `headline0` используется в качестве переменной переменной для определения правильного значения SEO-заголовка.

Можно также вложить переменные переменных более чем на два уровня вглубь. Использование трех символов `$` — как в случае с `$$$name` — тоже допустимо, как и `$$${some_function()}`. Однако для лучшей читаемости кода и его поддержки рекомендуется ограничивать уровни вариативности в именах переменных. В любом случае рассматриваемый инструмент используется довольно редко, но стоит иметь в виду, что многочисленные уровни косвенного обращения к переменным сделают ваш код непонятным и трудным для тестирования и сопровождения, если вдруг что-то сломается.

## Читайте также

Документация по переменным переменных (https://oreil.ly/wNBh0).

# 1.3. Обмен значениями между переменными

## Задача

Вы хотите поменять местами значения, хранящиеся в двух переменных, не вводя при этом дополнительных переменных.

## Решение

В следующей строчке кода используется языковая конструкция `list()`, которая служит для повторного присвоения значений переменным:

```
list($blue, $green) = array($green, $blue);
```

Более лаконичный вариант предыдущего решения — использовать для списка и массива короткий синтаксис, который стал доступен в PHP 7.1:

```
[$blue, $green] = [$green, $blue];
```

## Обсуждение

Ключевое слово `list` в PHP не является функцией, хотя и выглядит таковой. Это языковая конструкция, которая позволяет присваивать списку переменных значения за одну операцию. С ее помощью разработчики могут задавать значения сразу нескольким переменным из другой коллекции значений (например, из массива). Она также разрешает разбивать массивы на отдельные переменные.

В современном PHP квадратные скобки `[]` применяют для лаконичного обозначения массивов: запись `[1, 4, 5]` эквивалентна `array(1, 4, 5)`.

И list, и array в PHP относятся к языковым конструкциям. Они встроены в язык и являются ключевыми словами, которые служат для управления системой. Некоторые ключевые слова, например if, else или echo, легко отличить от пользовательского кода. В то же время такие языковые конструкции, как list, array и exit, выглядят как функции, но, как и ключевые слова, они встроены в язык и ведут себя несколько иначе, чем обычные функции. Список всех зарезервированных ключевых слов (https://oreil.ly/OJD13) можно найти в руководстве по PHP (https://oreil.ly/OJD13). Оно также содержит информацию о том, как использовать каждую из языковых конструкций.

Начиная с версии PHP 7.1, разработчики могут применять тот же лаконичный синтаксис для `list()`, создавая более читабельный код. При присвоении значений из массива массиву переменных использование подобного синтаксиса по обе стороны оператора присваивания (`=`) имеет смысл и уточняет ваше намерение.

Наш пример явно меняет местами значения, хранящиеся в переменных `$green` и `$blue`. Это то, что может сделать инженер при развертывании приложения для перехода с одной версии API на другую. При скользящем развертывании текущая активная среда часто называется «зеленым» развертыванием, а новая, потенциальная замена — «синим», указывая балансировщикам нагрузки и другим зависимым приложениям переключиться с «зеленой» на «синюю» среду и проверить подключение и функциональность, прежде чем подтвердить, что развертывание идет нормально.

В примере 1.4 рассмотрим ситуацию, когда приложение использует API с префиксом даты развертывания. Приложение отслеживает, какую версию API оно использует (`$green`), и пытается переключиться на новую среду, чтобы проверить возможность подключения. Если проверка подключения не удается, приложение автоматически переключается обратно на старое окружение.

**Пример 1.4.** Переключение синей/зеленой среды

```
$green = 'https://2021_11.api.application.example/v1';
$blue  = 'https://2021_12.api.application.example/v1';

[$green, $blue] = [$blue, $green];

if (connection_fails(check_api($green))) {
    [$green, $blue] = [$blue, $green];
}
```

Конструкцию `list()` также допускается применять для извлечения определенных значений из произвольной группы элементов. Пример 1.5 иллюстрирует, как адрес, хранящийся в виде массива, может использоваться в различных контекстах для извлечения только нужных значений.

**Пример 1.5.** Использование функции list() для извлечения элементов массива

```
$address = ['123 S Main St.', 'Anywhere', 'NY', '10001', 'USA'];

// Извлечение каждого элемента в виде именованных переменных
[$street, $city, $state, $zip, $country] = $address;

// Извлечение и именование только штата
[,,$state,,] = $address;

// Извлечение только страны
[,,,,$country] = $address;
```

Каждое извлечение в предыдущем примере является независимым и устанавливает только те переменные, которые необходимы[1]. Для более сложных приложений, обрабатывающих значительные объемы данных, установка ненужных переменных может привести к проблемам с производительностью. Хотя `list()` является мощным инструментом для разбивки коллекций вроде массивов, он подходит только для простых случаев, подобных тем, что рассмотрены в предыдущих примерах.

## Читайте также

Документация по `list()` (https://oreil.ly/bzO7i), `array()` (https://oreil.ly/tq1Z_) и PHP RFC по лаконичному синтаксису `list()` (https://oreil.ly/Ou98z).

[1] Вспомните, как во введении к этой главе объяснялось, что ссылки на переменные, которые не заданы явно, оцениваются как «пустые». Это означает, что вы можете задавать те значения и переменные, которые вам нужны.