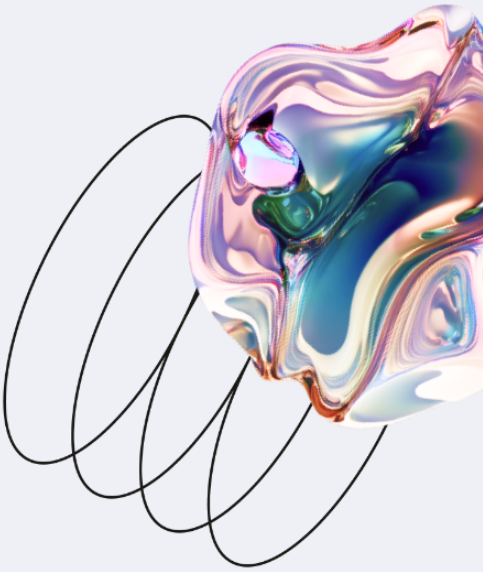
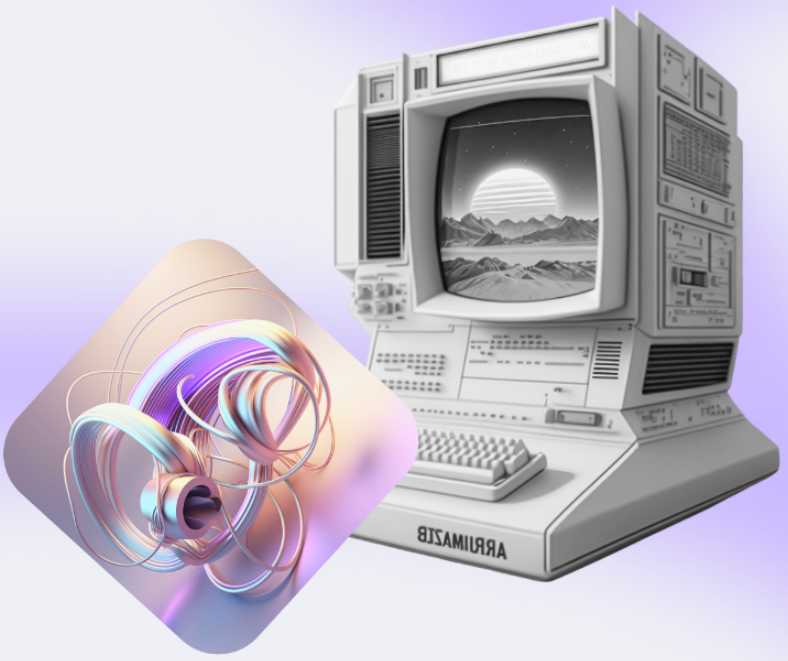
GeekBrains



Условия, массивы, циклы, функции

Основы PHP



Оглавление

Введение

Мы уже познакомились с основами основ программирования на языке PHP, имеем готовую среду для разработки.

Настало время углубляться в более сложные концепции. В этой лекции мы рассмотрим основные концепции ветвления, массивов, циклов и функций в языке PHP, которые являются необходимыми знаниями для каждого программиста.

Словарь терминов

Ветвления в PHP – это механизм, который позволяет программистам принимать решения на основе условий. Это позволяет программам принимать решения на основе данных, вводимых пользователем или получаемых из других источников.

Массивы – наборы значений, которые могут быть обработаны как один объект. Мы рассмотрим, как создавать массивы, добавлять и удалять элементы, а также как использовать массивы для решения различных задач.

Массивы – это упорядоченные списки, которые могут содержать любые типы данных: числа, строки, объекты, другие массивы и т.д. Они используются для хранения множества значений в одной переменной и обеспечивают простой и эффективный способ работы с данными.

Область применения массивов – это хранение в памяти программы однородных данных.

Одномерный массив – это простой список значений.

Многомерный массив – это массив, содержащий другие массивы в качестве элементов.

Ассоциативные массивы означают, что для каждого элемента массива можно указать свой ключ в виде строки (в индексных массивах ключом является целое число).

Функция count() – возвращает количество элементов в массиве

Функция array_push() – добавляет элемент в конец массива

Функция array_pop() – удаляет и возвращает последний элемент массива

Функция array_shift() – удаляет и возвращает первый элемент массива

Функция array_unshift() – добавляет элемент в начало массива

Функция array_slice() – возвращает выбранные элементы массива в новом массиве

Функция array_merge() – объединяет два или более массива в один

Функция array_reverse() – возвращает массив в обратном порядке

Функция array_search() – ищет значение в массиве и возвращает его ключ, если найдено

Функция in_array() – проверяет, содержится ли значение в массиве

Обход массива – это процесс перебора всех элементов массива в цикле для выполнения каких-либо операций с каждым элементом.

Циклы – используются в PHP для повторения одного и того же блока кода несколько раз. Мы рассмотрим операторы «for», «while» и «do-while» и объясним, как использовать их в своих программах.

Цикл while – цикл, который выполняет блок кода, пока заданное условие истинно.

Цикл do-while – цикл, который очень похож на цикл while, но он выполняет блок кода хотя бы один раз, даже если условие сразу ложно.

Цикл for – цикл, который выполняет блок кода заданное количество раз.

Цикл foreach – цикл, который является синтаксическим улучшением цикла for и чаще всего используется для перебора элементов массива.

Функции – набор инструкций, которые могут быть вызваны из других частей программы. Мы рассмотрим, как создавать функции, передавать им параметры и возвращать значения.

Функции – это блоки кода, которые можно вызывать из другого места в программе, для выполнения конкретной операции, указав только имя самой операции.

Параметры – это переменные, которые могут быть переданы в функцию при ее вызове.

Побочный эффект – это изменение состояния программы, которое происходит вне функции и вызывается ее выполнением.

Объявление типов – это мощный инструмент для создания более надежного и читаемого кода в PHP. Объявление типов позволяет задать тип данных, который ожидается для конкретного параметра или возвращаемого значения, что делает код более читаемым и позволяет PHP проверять правильность типов во время выполнения.

Области видимости – это когда переменные могут быть доступны только в определенных частях кода, и их значение может быть изменено только в рамках этой области видимости.

Локальные переменные создаются внутри функции и доступны только в рамках этой функции.

Символ амперсанда & – символ передачи параметра по ссылке, он используется перед именем переменной при ее объявлении в определении функции.

Условные операторы – операторы, которые позволяют проверить, является ли определенное условие истинным, и выполнить определенный код, если это условие истинно.

Оператор if – оператор, который используется для выполнения кода, если определенное условие является истинным.

Операторы сравнения — операторы, которые позволяют сравнивать значения переменных и выполнить определенный код в зависимости от результата сравнения.

Логические операторы – операторы, которые используются для соединения нескольких условий и выполнения определенного кода, если все эти условия истинны.

Оператор and («and», «&&») – оператор, который возвращает истинное значение только в том случае, если оба условия истинны.

Оператор or («or», «||») – оператор, который возвращает истинное значение, если хотя бы одно из условий истинно.

Оператор not («!») – оператор, который инвертирует результат вычисления, то есть, возвращает ложное значение, если условие истинно, и наоборот.

Оператор else if – оператор, который используется для проверки дополнительных условий, если первое условие ложно.

Оператор else – оператор, который используется для выполнения кода, если все предыдущие условия ложны.

Тернарный оператор – оператор, который позволяет выполнить определенный код в зависимости от того, выполняется ли определенное условие или нет.

Оператор switch – оператор, который используется для сравнения выражения с несколькими значениями и выполнения действий, соответствующих этим значениям.

break – оператор, который прерывает выполнение оператора switch и переходит к следующей строке кода после него.

Оператор match – оператор, который является усовершенствованным аналогом оператора switch. В отличие от switch, оператор match является более строгим и не позволяет выполнять неявные преобразования типов данных.

Ветвления

При линейном выполнении программы мы можем выполнять только простые инструкции, которые никогда не будут менять своего поведения ни при каких условиях. Они будут последовательно выполнять одни и те же шаги.

Но что, если нам нужно реагировать по-разному на разные состояния?

Например, если у пользователя есть возможность отправить нам один из нескольких вариантов ответа, нам нужно проверить, правильный ли он, и в случае ошибки сообщить пользователю.

> 💡 Таким образом, мы формируем условие «если предположение истинно, сделай шаг А, иначе сделай шаг Б»

Ветвления в PHP – это механизм, который позволяет программистам принимать решения на основе условий. Это позволяет программам принимать решения на основе данных, вводимых пользователем или получаемых из других источников.

Например, в программе можно написать условие, которое проверяет, является ли файл, отправленный пользователем, фотографией или PDF документом. И в разных ветках кода будет обозначено либо размещение фотографии в галерее, либо PDF документ будет сохранен в хранилище.

Ветвления также позволяют программам выполнять различные действия в зависимости от данных, введенных пользователем или других факторов.

Это может быть полезно, например, для написания программ:

- которые адаптируются к разным условиям;
- для написания приложений, которые реагируют на действия пользователей.

В языке программирования PHP существует несколько различных видов ветвлений, которые можно использовать для решения различных задач.

Знание этих видов ветвлений позволяет программистам писать более эффективный и читаемый код.

Условные операторы

Условные операторы позволяют проверить, является ли определенное условие истинным, и выполнить определенный код, если это условие истинно.

Оператор If

Оператор if используется для выполнения кода, если определенное условие является истинным.

Он имеет следующий синтаксис:

```
if (условие) {
  // код, который будет выполнен, если условие истинно
}
```

Здесь важно разобраться, что же такое условие.

В самом простом варианте – это может быть значением переменной.

```
$isRed = true;
if ($isRed) {
  // код, который будет выполнен, если мы работаем с красным цветом
}
```

Условием может быть выражение – сравнение значений.

Поэтому посмотрим, какие могут быть выражения

Операторы сравнения

Операторы сравнения позволяют сравнивать значения переменных и выполнить определенный код в зависимости от результата сравнения.

В PHP есть следующие операторы сравнения:

- «==» – проверка на равенство;
- «===» – проверка на равенство и соответствие типов;
- «!=» – проверка на неравенство;
- «!==» – проверка на неравенство или несоответствие типов;
- «>» – значение слева от оператора больше значения справа;
- «<» – значение слева от оператора меньше значения справа;
- «>=» – значение слева от оператора больше значения справа;
- «<=» – значение слева от оператора меньше или равно значения справа.

Пример использования:

```
$a = 5;
$b = 10;
if ($a < $b) {
  echo "a меньше, чем b";
}
```

Важно разобраться со сравнением типов:

```
$a = 5;
$b = "5";
if ($a == $b) {
  echo "a идентично b";
}

if ($a !== $b) {
  echo "a не идентично b или имеет другой тип";

}
```

В данном примере, несмотря на то, что при приведении типов строка "5" может быть приведена к целочисленному значению «5», что и происходит при простом сравнении, в случае сравнения по значению и типу не подлежит приведению.

Логические операторы

Логические операторы используются для соединения нескольких условий и выполнения определенного кода, если все эти условия истинны. В PHP есть три основных логических оператора:

- and,
- or,
- not.

And

Оператор and (его можно записывать и как «and» и как «&&») возвращает истинное значение только в том случае, если оба условия истинны.

Пример использования:

```
$a = 5;
$b = 10;
if ($a > 0 && $b > 0) {
  echo "Оба числа положительные";
}
```

Or

Оператор or (его можно записывать и как «or» и как «||») возвращает истинное значение, если хотя бы одно из условий истинно.

Пример использования:

```
$a = -5;
$b = 10;
if ($a < 0 or $b < 0) {
  echo "Как минимум одно из чисел - отрицательное";
}
```

Not

Оператор not или «!» инвертирует результат вычисления, то есть, возвращает ложное значение, если условие истинно, и наоборот.

Пример использования:

```
$a = 5;

if (!($a == 10)) {

  echo "a не равно 10";

}
```

Else If

До этого мы рассматривали только простые условия if. Но что, если нам надо создать более сложное ветвление?

Оператор else if используется для проверки дополнительных условий, если первое условие ложно.

Он имеет следующий синтаксис:

```
if (условие1) {

  // код, который будет выполнен, если условие1 истинно

} else if (условие2) {

  // код, который будет выполнен, если условие2 истинно

}
```

Else

Оператор else используется для выполнения кода, если все предыдущие условия ложны.

Он имеет следующий синтаксис:

```
if (условие1) {

  // код, который будет выполнен, если условие1 истинно
```

```
} else if (условие2) {
  // код, который будет выполнен, если условие2 истинно
} else {
  // код, который будет выполнен, если все предыдущие условия ложны
}
```

Теперь посмотрим, как эти ветвления используются в PHP:

```
$age = 25;

if ($age < 18) {
  echo "Вы несовершеннолетний";
} else if ($age >= 18 && $age < 60) {
  echo "Вы взрослый человек";
} else {
  echo "Вы пожилой человек";
}
```

В данном примере, если возраст меньше 18, будет выведено сообщение «Вы несовершеннолетний».

Если возраст больше или равен 18 и меньше 60, будет выведено сообщение «Вы взрослый человек».

Если возраст больше или равен 60, будет выведено сообщение «Вы пожилой человек».

- 🔥 Обратите внимание на то, что обязательным к наличию в ветвлении является только оператор **if**.
- 🔥 Он может иметь else, но не иметь else if.
- 🔥 Или иметь else if, но не иметь else.

Тернарный оператор

Тернарный оператор позволяет выполнить определенный код в зависимости от того, выполняется ли определенное условие или нет.

Он имеет следующий синтаксис:

```
(условие) ? (код, если условие истинно) : (код, если условие ложно);
```

Пример использования:

```
$age = 18;

$message = ($age < 18) ? "Вы несовершеннолетний" : "Вы совершеннолетний";

echo $message;
```

Оператор switch

Оператор switch в PHP по большей части улучшает синтаксис и используется для сравнения выражения с несколькими значениями и выполнения действий, соответствующих этим значениям. Улучшение синтаксиса состоит в том, что вместо большого количества if-else if-else if... есть возможность написать более лаконичную конструкцию.

Он имеет следующий синтаксис:

```
switch (выражение) {
  case значение1:
    // действия, выполняемые при совпадении со значением1
    break;
  case значение2:
    // действия, выполняемые при совпадении со значением2
    break;
  ...
  default:
     // действия, выполняемые при отсутствии совпадений
```

```
    break;
}
```

🔥 В операторе switch выражение сравнивается последовательно с каждым значением в блоках case.

Если значение совпадает с выражением, то выполняются действия, соответствующие этому значению.

Если ни одно из значений не совпадает, то выполняются действия, указанные в блоке default (если он задан).

Каждый блок case может содержать любой допустимый PHP-код, включая другие ветвления и циклы. В конце каждого блока case должен быть оператор break, который прерывает выполнение оператора switch и переходит к следующей строке кода после него.

Если оператор break не указан в блоке case, то выполнение кода продолжится в следующем блоке case.

💡 Обратите внимание на то, что **switch** с точки зрения чистоты кода стоит использовать очень аккуратно

- Не пишите внутри case сложные конструкции и другие ветвления!
- Не используйте switch, если у вас всего 3-4 ветки логики!

Автор книги «Чистый код» Роберт Мартин отмечает «Мое общее правило в отношении команд switch гласит, что эти команды допустимы, если они встречаются в программе однократно, используются для создания полиморфных объектов и скрываются за отношениями наследования, чтобы оставаться невидимыми для остальных частей системы. Конечно, правил без исключений не бывает и в некоторых ситуациях приходится нарушать одно или несколько условий этого правила.»

Пример использования оператора switch:

```
$product = "Burger";
```

```
switch ($product) {
  case "HotDog":
    echo "Выбран Хот-Дог";
    break;
  case "Cola":
    echo "Выбрана кола";
    break;
  case "Burger":
    echo "Выбран бургер";
    break;
  default:
    echo "Продукт неизвестен";
    break;
}
```

💡 Оператор switch удобен для использования в случаях, когда требуется сравнить одно выражение с несколькими значениями и выполнить соответствующие действия.

Однако его использование может привести к нечитабельному коду, если в нем содержится много блоков case. В этом случае лучше использовать другие виды ветвлений, например, оператор if.

🔥 Оператор switch неявно приводит тип переменной к тому, который используется в case. Это, как и любое неявное приведение типов, может приводить к неожиданному поведению.

В PHP 8 появился новый оператор ветвления match, который является усовершенствованным аналогом оператора switch. В отличие от switch, оператор match является более строгим и не позволяет выполнять неявные преобразования типов данных.

Оператор match имеет следующий синтаксис:

```
match (выражение) {
  значение1 => выражение1,
  значение2 => выражение2,
  ...
  default => выражение_по_умолчанию,
}
```

В выражении можно использовать различные типы данных, такие как числа, строки, массивы и объекты.

Каждое значение, переданное оператору match, сравнивается с выражением в операторе.

Если значение совпадает с выражением, то выполняется соответствующее выражение.

Если ни одно из значений не совпадает, то выполняется выражение по умолчанию (если оно задано).

Пример использования оператора match:

```
$value = "Burger";

$result = match ($value) {
  "HotDog" => "Выбран Хот-Дог",
  "Cola" => "Выбрана кола",
  "Burger" => "Выбран бургер",
  default => "Продукт неизвестен",
};

echo $result;
```

Массивы

Массивы в PHP – это упорядоченные списки, которые могут содержать любые типы данных: числа, строки, объекты, другие массивы и т.д. Они используются для хранения множества значений в одной переменной и обеспечивают простой и эффективный способ работы с данными.

Основная область применения массивов – это хранение в памяти программы однородных данных. Например, если у нас есть набор студентов, а мы хотим посчитать средний балл.

До массивов нам пришлось бы создавать для каждого студента и для каждого его свойства свою переменную.

Мы же хотим получить структуру вида и хранить её в одной переменной:

Студент 1

Имя

Балл

Студент 2

Имя

Балл

Массивы могут быть одномерными и многомерными.

Одномерный массив – это простой список значений, а многомерный массив – это массив, содержащий другие массивы в качестве элементов.

Рассмотрим простой вариант массива – индексный массив. Для создания такого массива в PHP можно использовать несколько способов. Один из самых простых способов – это создание массива с помощью функции array().

Например, следующий код создает массив, содержащий три числа:

```
$array = array(1, 2, 3);
```

Также можно создать массив с помощью квадратных скобок.

Например, следующий код создает массив, содержащий три строки:

```
$array = ['foo', 'bar', 'baz'];
```

Для доступа к элементам такого массива в PHP используется индексация. Индексы в массивах начинаются с нуля.

Например, следующий код выводит на экран элементы массива $array:

```
$array = ['foo', 'bar', 'baz'];
echo $array[0]; // выводит 'foo'
echo $array[1]; // выводит 'bar'
echo $array[2]; // выводит 'baz'
```

Массивы в PHP могут быть ассоциативными, что означает, что для каждого элемента массива можно указать свой ключ в виде строки (в индексных массивах ключом является целое число).

Например, следующий код создает ассоциативный массив, содержащий пары «ключ-значение»:

```
$student = array(
    'name' => 'Иван',
    'age' => 18,
    'email' => 'john@example.com'
);
```

Для доступа к элементам ассоциативного массива в PHP используется ключ. Например, следующий код выводит на экран значение элемента массива с ключом 'name':

```
$student = array(
    'name' => 'Иван',
    'age' => 18,
    'email' => 'john@example.com'
```

```
);

echo $array['name']; // выводит 'Иван'
```

Массивы в PHP поддерживают множество функций для работы с данными, например, для добавления и удаления элементов, для сортировки элементов, для поиска элементов и т.д.

Большинство функций можно найти в официальной документации PHP.

Некоторые из наиболее популярных функций работы с массивами в PHP:

- Функция count() – возвращает количество элементов в массиве
- Функция array_push() – добавляет элемент в конец массива
- Функция array_pop() – удаляет и возвращает последний элемент массива
- Функция array_shift() – удаляет и возвращает первый элемент массива
- Функция array_unshift() – добавляет элемент в начало массива
- Функция array_slice() – возвращает выбранные элементы массива в новом массиве
- Функция array_merge() – объединяет два или более массива в один
- Функция array_reverse() – возвращает массив в обратном порядке
- Функция array_search() – ищет значение в массиве и возвращает его ключ, если найдено
- Функция in_array() – проверяет, содержится ли значение в массиве

Кроме того, в PHP существуют многие другие функции для работы с массивами, такие как:

- array_key_exists(),
- array_diff(),
- array_intersect(),
- array_map() и многие другие.

Также в PHP есть возможность использовать многомерные массивы, которые представляют собой массивы в массивах. Например, следующий код создает

многомерный массив, содержащий два элемента-массива, каждый из которых содержит три элемента:

```
$array = array(
    array(1, 2, 3),
    array(4, 5, 6)
);
```

С точки зрения структуры это будет выглядеть так:

- **0**

  0 => 1

  1 => 2

  2 => 3

- **1**

  0 => 4

  1 => 5

  2 => 6

Для доступа к элементам многомерного массива в PHP используется индексация и ключи. Например, следующий код выводит на экран второй элемент первого элемента многомерного массива:

```
$array = array(
    array(1, 2, 3),
    array(4, 5, 6)
);
echo $array[0][1]; // выводит 2
```

Циклы

Рядом с массивами во многих языках программирования всегда идут и циклы.

Ведь в работе с массивом часто появляется потребность так называемого обхода массива. Обход массива – это процесс перебора всех элементов массива в цикле для выполнения каких-либо операций с каждым элементом.

Например, если мы хотим взять наш набор студентов и посчитать средний возраст, то без цикла не обойтись – ведь список каждый раз может быть разным.

Давайте предположим, что у нас есть массив

```
$students = [
  [
    'name' => 'Иван',
    'score' => 4.5
  ],
  [
    'name' => 'Мария',
    'score' => 5
  ],
  [
    'name' => 'Петр',
    'score' => 3.7
  ]
];
```

Поработаем с ним при помощи разных циклов.

В PHP есть несколько видов циклов, которые позволяют многократно выполнять один и тот же блок кода.

Цикл while

Цикл while выполняет блок кода, пока заданное условие истинно.

Если нам нужно обойти наш массив студентов, нам надо:

- Знать, с каким элементом массива мы сейчас работаем. Значит, нужно хранить счетчик-указатель.

- Знать, когда мы дошли до конца массива. То есть, знать его длину.
- Собирать сумму баллов по студенту

```
$counter = 0; // начинаем самого первого элемента
$summ = 0; // итоговая сумма баллов
while ($counter < count($students)) {
    $summ += $students[$counter]['score'];
    $counter++;
}
echo 'Средний балл: ' . $summ / count($students);
```

Цикл do-while

Цикл do-while очень похож на цикл while, но он выполняет блок кода хотя бы один раз, даже если условие сразу ложно.

Синтаксис цикла do-while выглядит следующим образом:

```
$counter = 0; // начинаем самого первого элемента
$summ = 0; // итоговая сумма баллов
do {
    $summ += $students[$counter]['score'];
    $counter++;
} while ($counter < count($students))
echo 'Средний балл: ' . $summ / count($students);
```

Цикл for

Цикл for выполняет блок кода заданное количество раз. Это классический вариант цикла, который встречается практически во всех языках программирования.

Синтаксис цикла for выглядит следующим образом:

for (начальное_значение; условие; шаг) {

 // выполняемый код

}

```
for ($counter = 0; ($counter < count($students); $counter++) {
    $summ += $students[$counter]['score'];
}
echo 'Средний балл: ' . $summ / count($students);
```

Как видите, это максимально универсальный и лаконичный цикл.

Цикл foreach

Цикл foreach является синтаксическим улучшением цикла for и чаще всего используется для перебора элементов массива.

Синтаксис цикла foreach выглядит следующим образом:

foreach ($массив as $ключ => $значение) {

 // выполняемый код

}

Наша задача тут решится следующим образом:

```
foreach ($students as $student) {
    $summ += $student['score'];
}
echo 'Средний балл: ' . $summ / count($students);
```

Как видите, в данной конструкции нам уже не нужен счётчик.

Операторы break и continue

Оператор break прерывает выполнение цикла, а оператор continue пропускает текущую итерацию цикла и переходит к следующей.

Например, следующий цикл выводит нечетные числа от 1 до 10:

```
for ($i = 1; $i <= 10; $i++) {
  if ($i % 2 == 0) {
    continue;
  }
echo $i;
```

Оператор break на практике может пригодиться в задачах поиска по массиву. Например, мы останавливаем цикл, если понимаем, что нашли искомый элемент.

Оператор continue помогает не тратить аппаратные ресурсы на итерацию в случае, если мы понимаем, что некое условие не выполняется. Например, если в массиве лежат email адреса, на которые надо отправлять письма, но в цикле перед отправкой мы проверяем адрес на валидность, то в случае неправильно сформированного email адреса письмо отправлять не нужно. Это сэкономит общее время работы.

Функции

До этого момента мы писали код, который выполняется один раз.

Но что, если нам надо иметь возможность выполнить расчёт несколько раз?

Конечно, можно скопировать код вызова. Но тогда при изменении логики расчёта придётся править его в нескольких местах. А также не получится адаптироваться к ситуации, когда количество вызовов заранее неизвестно. Например, когда студентов в список добавляет пользователь. И мы не знаем, сколько студентов будет добавлено.

В таком случае нам помогут функции – это блоки кода, которые можно вызывать из другого места в программе, для выполнения конкретной операции, указав только имя самой операции. Функции являются важным инструментом в программировании, потому что они помогают разбивать большие программы на более мелкие и легко управляемые части.

В PHP функции определяются с помощью ключевого слова **function**, за которым следует имя функции и список параметров в круглых скобках. Параметры – это переменные, которые могут быть переданы в функцию при ее вызове. Тело

функции заключается в фигурные скобки и содержит код, который нужно выполнить при вызове функции.

Например, в работе с подсчётом среднего балла мы можем не знать, какой массив будет сформирован. Поэтому напишем функцию, которая будет принимать массив любого размера, а возвращать уже средний балл.

```
function getAverageScore($studentsArray) {
  foreach ($students as $student) {
    $summ += $student['score'];
  }
  return $summ / count($students);
}
```

В этом примере мы определяем функцию **getAverageScore**, которая принимает в качестве параметра массив студентов. Внутри функции мы все также считаем средний балл. Но не выводим его на экран, а возвращаем это значение с помощью ключевого слова **return**.

> Если бы мы выводили на экран значение, мы бы порождали побочный эффект – это изменение состояния программы, которое происходит вне функции и вызывается ее выполнением.

Сама функция может возвращать значение, но также может иметь побочные эффекты, которые могут быть нежелательными или неожиданными.

Вывод на экран – это неожиданный для пользователя функции эффект.

Пользователь сам должен решать, что делать с полученным значением.

> 💡 Хорошим тоном в программировании считается именование функций так, чтобы было понятно, что делает функция.

Автор книги «Чистый код» Роберт Мартин отмечает «Имя переменной, функции или класса должно сообщить, почему эта переменная существует, что она делает и как используется. Если имя требует дополнительных комментариев, значит, оно не передает намерений программиста. Лучше написать, что именно измеряется и в

каких именно единицах. Пример хорошего названия переменной: daysSinceCreation; Цель: убрать неочевидность.»

> 🔥 Функции могут быть очень мощным инструментом в программировании, позволяющим повторно использовать код и создавать более читаемый и поддерживаемый код.

В PHP есть множество встроенных функций, таких как **strlen** для вычисления длины строки, **array_sum** для вычисления суммы элементов массива и т.д.

С версии PHP 7.0 была добавлена поддержка объявления типов для параметров и возвращаемого значения функций. Объявление типов позволяет задать тип данных, который ожидается для конкретного параметра или возвращаемого значения, что делает код более читаемым и позволяет PHP проверять правильность типов во время выполнения.

Модифицируем наш код:

```
function getAverageScore(array $studentsArray) : float {
  foreach ($students as $student) {
    $summ += $student['score'];
  }
  return $summ / count($students);
}

echo getAverageScore($studentsArray);
```

Объявление типов – это мощный инструмент для создания более надежного и читаемого кода в PHP. Он помогает предотвратить ошибки типизации и сделать код более ясным и понятным для других разработчиков.

Области видимости

Переменные в PHP имеют свои области видимости – это означает, что они могут быть доступны только в определенных частях кода, и их значение может быть изменено только в рамках этой области видимости.

В PHP есть две основные области видимости переменных в функциях: локальная и глобальная.

Локальные переменные создаются внутри функции и доступны только в рамках этой функции. Они не могут быть использованы вне функции, и их значение сохраняется только во время выполнения функции.

При каждом вызове функции создаются новые локальные переменные.

В нашем примере:

```
function getAverageScore(array $studentsArray) : float {
  foreach ($students as $student) {
    $summ += $student['score'];
  }
  return $summ / count($students);
}


echo $summ; // тут будет ошибка
```

Мы не сможем обратиться к переменной **$summ** вне тела функции. Также и наоборот – в теле функции нельзя обратиться к переменным, которые определены вне его.

Есть механизм, который позволяет сделать такое обращение – **глобальные переменные**

```
$counter = 0;


function incrementCounter() {
    global $counter;
    $counter++;
}
```

Но использование глобальных переменных является дурным тоном в программировании, так как непонятно, где такая переменная может меняться.

Передача параметра по ссылке

В PHP параметры функций обычно передаются по значению, то есть копия значения переменной оказывается внутри функции, но не сама переменная:

```
$counter = 0;


function incrementCounter(int $counter): int {
    return $counter++;
}


echo $counter;
incrementCounter($counter);
echo $counter; // значение никак не изменилось
```

Однако в PHP также предусмотрена возможность передачи параметров по ссылке, при которой функция получает не копию значения, а саму переменную, и может изменять ее значение.

Для передачи параметра по ссылке используется символ амперсанда & перед именем переменной при ее объявлении в определении функции.

```
$counter = 0;


function incrementCounter(int &$counter): int {
    $counter++;
}


echo $counter;
incrementCounter($counter);
echo $counter;
```

При передаче параметров по ссылке необходимо учитывать, что это может привести к неожиданным результатам и ошибкам, особенно в случаях, когда

переменная изменяется в разных частях кода. Поэтому передачу параметров по ссылке следует использовать только в тех случаях, когда это действительно необходимо и обосновано.

Заключение

В лекции мы познакомились с тем, как в PHP устроены базовые структурные компоненты, без которых невозможно представить ни одну современную программу.

Теперь мы умеем применять ветвления, циклы и функции, знаем, какие массивы, как реализуются в языке.

И теперь можем продолжать обучение на более сложных концепциях.

Домашнее задание

1. Реализовать основные 4 арифметические операции в виде функции с тремя параметрами – два параметра это числа, третий – операция. Обязательно использовать оператор return.

2. Реализовать функцию с тремя параметрами: function mathOperation($arg1, $arg2, $operation), где $arg1, $arg2 – значения аргументов, $operation – строка с названием операции. В зависимости от переданного значения операции выполнить одну из арифметических операций (использовать функции из пункта 3) и вернуть полученное значение (использовать switch).

3. Объявить массив, в котором в качестве ключей будут использоваться названия областей, а в качестве значений – массивы с названиями городов из соответствующей области. Вывести в цикле значения массива, чтобы результат был таким: Московская область: Москва, Зеленоград, Клин Ленинградская область: Санкт-Петербург, Всеволожск, Павловск, Кронштадт Рязанская область ... (названия городов можно найти на maps.yandex.ru)

4. Объявить массив, индексами которого являются буквы русского языка, а значениями – соответствующие латинские буквосочетания (‘a’=> ’a’, ‘б’ => ‘b’, ‘в’ => ‘v’, ‘г’ => ‘g’, ..., ‘э’ => ‘e’, ‘ю’ => ‘yu’, ‘я’ => ‘ya’). Написать функцию транслитерации строк.

5. *С помощью рекурсии организовать функцию возведения числа в степень. Формат: function power($val, $pow), где $val – заданное число, $pow – степень.

6. *Написать функцию, которая вычисляет текущее время и возвращает его в формате с правильными склонениями, например:

    22 часа 15 минут

    21 час 43 минуты

Что можно почитать еще?

1. https://phptherightway.com/
2. https://php.net
3. Книга. Котеров Д. В. PHP 8
4. https://prowebmastering.ru/rekursiya-php.html