Материал занятия № 2

Курс:

«Создание web–приложений, исполняемых на стороне сервера при помощи языка программирования PHP, СУБД MySQL и технологии Ajax»

Модуль 01

ТЕМА: ЦИКЛЫ И ФУНКЦИИ

План занятия

- Определение циклов
- Виды циклов
- Прерывание циклов
- Прерывание итерации
- Что такое функция?
- Типы функций
- Анатомия функций
- Возвращение значений
- Встроенные функции

Циклы

Циклы позволяют повторять некоторое количество раз команды, которые называются **телом цикла**. Проход цикла называется **итерацией**. PHP поддерживает следующие виды циклов:

- Цикл с предусловием.
- Цикл с постусловием.
- Цикл со счётчиком.
- Цикл перебора массивов.

При использовании циклов есть возможность использования операторов `break` и `continue`. Первый из них прерывает работу всего цикла, а второй — текущей итерации.

Использование циклов упрощает и укорачивает код. Также циклы незаменимы в ситуациях, когда заранее неизвестно сколько раз должен выполниться блок кода, если число зависит от множества условий и вычисляться в момент выполнения сценария.

Цикл с предусловием

Цикл с предусловием сначала вычисляет значение логического выражения, и в случае true выполняет тело цикла, иначе пропускает его.

Формы записи цикла с предусловием для классического и альтернативного синтаксиса представлены в таблице 1.

Таблица 1 — Формы записи цикла с предусловием

Наименование	Классический синтаксис	Альтернативный синтаксис
1. **Начало цикла**	`while (cond) {}`	`<? while (cond): ?>`
2. **Конец цикла**	–	`<? endwhile; ?>`

Цикл требует выполнение выражения, пока выражение истинно, а если оно ложно, то он не будет выполнен. Значение проверяется каждый раз перед началом цикла.

Если значение выражения изменится в процессе выполнения вложенных выражений в цикле, выполнение не прекратится до конца итерации.

Цикл с постусловием

В отличие от цикла с предусловием, этот цикл проверяет значение выражения после итерации. Таким образом, тело цикла выполняется хотя бы один раз.

Формы записи цикла с постусловием для классического и альтернативного синтаксиса представлены в таблице 2.

Таблица 2 — Формы записи цикла с постусловием

Наименование	Классический синтаксис	Альтернативный синтаксис
1. **Начало цикла**	`do {}`	–
2. **Конец цикла**	`while (cond)`	–

Примечание: данный тип цикла не имеет поддержки альтернативного синтаксиса.

Цикл со счётчиком

Этот цикл используется для выполнения тела определённое число раз. С помощью циклов данного типа можно создавать более сложные конструкции.

Цикл начинает работу с **инициализации** (1). Данные команды выполняются только один раз. После производится **проверка условия** (2), и при `true` выполняется тело. После выполнения последнего оператора тела происходит завершение итерации — **выполнение инкремента или декремента** (3).

Формы записи цикла со счётчиком для классического и альтернативного синтаксиса представлены в таблице 3.

Таблица 3 — Формы записи цикла со счётчиком

Наименование	Классический синтаксис	Альтернативный синтаксис
1. **Начало цикла**	`for (1, 2, 3) {}`	`<? for (1, 2, 3): ?>`
2. **Конец цикла**	–	`<? endfor; ?>`

Цикл перебора массивов

Существует также и цикл для последовательного перебора всех элементов массива (в случае использования с переменными других типов или неинициализированными переменными будет сгенерирована ошибка).

Существуют два вида данного цикла:

1. **Простой перебор массива** — переменные внутри цикла перебираются только по индексам. Использует запись формата $arrExp as $value, где $arrExp — сам массив, а $value — значение элемента массива во время итерации.
2. **Парный перебор массива** — используется для перебора элемента массива в паре с его ключом. Использует запись формата $arrExp as $key => $value, где $arrExp — сам массив, $key — индекс элемента массива, а $value — его значение.

Формы записи цикла перебора массива для классической и альтернативной форм записи представлены в таблице 4.

Таблица 4 — Формы записи цикла перебора массивов

Наименование	Классический синтаксис	Альтернативный синтаксис
1. **Начало цикла**	`foreach (exp) {}`	`<? foreach (exp): ?>`
2. **Конец цикла**	–	`<? endforeach; ?>`

Такой цикл будет продолжаться, пока он не пройдёт через каждый элемент массива.

Примечание: данный тип цикла использует лишь копию массива и любые изменения в оригинальный массив не будут применяться.

Прерывание цикла

Конструкция `break` используется для немедленного выхода из цикла. Она может задаваться с одним *необязательным* параметром — номером вложенного цикла, из которого нужно выйти (по умолчанию используется 1 — выход из текущего цикла):

- `break;`
- `break(1);`

Примечание: нумерация циклов начинается изнутри. Таким образом у первого цикла будет бóльший номер для прерывания, чем у любого вложенного.

Прерывание итерации

Конструкция `continue` используется для выхода из итерации. Как и `break`, она так же может задаваться с номером вложенного цикла для выхода. В основном, `continue` позволяет сэкономить количество фигурных скобок в коде и увеличить читаемость программного кода.

Чаще всего конструкция используется в циклах-фильтрах, где нужно выбрать только значения, что удовлетворяют заданным условиям.

Что такое функция?

Функция — важный элемент программирования, который позволяет группировать и использовать один и тот же код множество раз. Функции также называют *подпрограммой*, поэтому она имеет свою точку входа и свой «выход».

С точки зрения внешней программы функция может быть названа *чёрным ящиком*[1]. Функция определяет свою локальную область видимости, куда входят входные параметры, и переменные, что объявляются в теле самой функции. Главная особенность функции — это её *вызов*.

Таким образом, функция — это средство проектирования, что позволяет сделать **декомпозицию** программы, то есть, разделить её на более простые части. Решение маленьких задач будет создано по отдельности, благодаря чему в случае ошибки её достаточно будет исправить лишь в одном месте.

Типы функций

Существуют два типа функций — *встроенные* и *пользовательские*:

- **Встроенные функции** уже написали создатели языка программирования. Одна из хорошо знакомых функций — функция, которая выводит переданный ей текст на экран — `print()`.
- **Пользовательские функции** создаёт разработчик. Они используются только внутри одного проекта или сценария.

---

[1] **Чёрный ящик** — термин из *поведенческого тестирования*, полное название — *тестирование по стратегии чёрного ящика*, способ проверки работы программы без знаний о внутреннем (программном) устройстве такой программы или её кода. Иначе говоря, при такой проверке у тестировщиков нет доступа к исходному коду приложения.

Анатомия функции

Работа с функциями состоит из *объявления* и *определения:*

- **Объявление** (`declaration`) функции содержит список входных аргументов.
- **Определение** (`definition`) функции содержит исполняемый код функции.

Любая функция может состоять из имени и аргументов. **Имя функции** используется для вызова и должно быть уникальным, когда **аргументы функции** — это переменные, которые могут быть получены из внешнего кода, но не являются обязательными при создании новой функции.

Перед вызовом функции её необходимо объявить:

```
function <имя функции> (<аргументы функции>) {
    // Код функции
}
```

Внутри функций может быть выполнен любой код, включая вызовы других функций. Все переменные, которые были определены внутри, не будут доступны за её пределами. Такие переменные будут локальны по отношению к функции.

При этом значения переменных могут быть также заданы по умолчанию, из-за чего передача этого аргумента будет являться необязательно, так как он уже определён в коде. Аргументы передаются строго в порядке, определённом при объявлении функции.

Возвращение значений

Функция может *возвращать* некоторое значение — число, строку, логический оператор (`true` и `false`) и так далее. Такое

значение называется **результатом функции**. Для возвращения значения применяется ключевое слово `return`.

Пример функции, которая принимает два числа типа `Integer` и возвращает их сумму:

```
function add (int $a, int $b = 2) {
    return $a + $b;
}
```

Для того, чтобы использовать возвращаемое значение, результат функции необходимо будет сохранить в переменную, либо использовать напрямую, например, вместе с оператором `echo`:

- `$result = add(1, 2);`
- `echo add(1, 2);`

Если в функции используется конструкция выбора, то ключевое слово `return` можно использовать для закрытия всего кейса вне зависимости от его содержания:

```
function example ($var) {
    switch ($var) {
        case "value":
            // Код кейса
            return true;
    }
}
```

Если после `return` в функции идут другие инструкции, то они не будут выполняться, а среда разработки (например, Visual Studio Code), уведомит о наличии недосягаемого участка кода:

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

Unreachable code detected
PHP(PHP0419)

Функции, которые не возвращают значений, иногда называют *процедурами.*

Встроенные функции

Среди существующих встроенные функций можно выделить следующие[2]:

- `rand(int $min, int $max)` — генерирует случайное число. При указании минимального и максимального значения работает в переданном диапазоне.

Например, вызов `rand(1, 12)` вернёт случайное число от 1 до 12.

- `strlen(string $string)` — возвращает длину строки.
- `substr_count(string $haystack, string $needle)` — возвращает число вхождений подстроки.

Например, вызов `substr_count("This is a function", "is")` вернёт число 2, так как в строке дважды повторяется `is` (подчёркнуто в примере вызова).

[2] Больше о стандартных функциях для работы со строками и математических модулях написано в официальной документации по PHP.