Операторы и выражения (1)

- Введение
- Выражения в PHP
- Арифметические операторы
- Логические операторы
- Строковые операторы

Введение

Прежде чем начать разберемся с основной его терминологией (дочитать до конца!).

Выражение – комбинация операндов и операций, задающая порядок вычисления некоторого значения.

Операнды — это объекты, **над которыми** или при помощи которых выполняются действия. Операнд в простейшем случае является константой или идентификатором. В общем случае каждый **операнд выражения** также представляет собой **выражение**, имеющее некоторое значение.

Операции определяют действия, выполняемые над операндами. Возвращают некоторое значение.

Оператор — это некоторая конструкция, **присущая данному конкретному языку**, изменяющая состояние памяти компьютера, но ничего не возвращающая. Оператором называется нечто, состоящее из одного или более **выражений**, которое можно вычислить как новое значение (таким образом, вся конструкция может рассматриваться как выражение).

Существует тонкая грань между оператором и операцией и между оператором и выражением.

> ***К сведению***. **Оператор** - это символ в языке программирования, а **операция** - это действие, которое выполняется с помощью этого символа.
> Например: Оператор + выполняет операцию сложения. Оператор * выполняет операцию умножения и т.п.

Сложнее с оператором и выражением. В некоторых источниках указано - оператор может состоять из одного или нескольких выражений, в других - выражение состоит из операторов. Верно и то и другое. Но для ясности предлагаю остановиться на таком подходе.

> ***К сведению***. **Выражение** - это некоторая инструкция, может содержать обычные числа, переменные, строки или функции, а также один или несколько **операторов**. Для использования в выражениях предусмотрены следующие операторы: сложение, вычитание, умножение и т.д...

И теперь повторим кратко все то же самое, но в обратном порядке.

Выражение состоит из **операторов** - оператор объединяет **операнды** – **каждый операнд** также представляет собой **выражение**. Круг замкнулся.

> ***Совет***. Не стоит слишком серьёзно относиться к некоторым фактам. До тех пор, пока не начал преподавать, я много лет спокойно программировал и даже не представлял себе, что все это так замысловато.

Выражения в PHP

Выражения — это краеугольный камень PHP. Почти все, что мы пишем в PHP, является выражением. Выражения являются "кирпичиками", из которых состоят PHP-программы.

Под выражением в PHP понимается то, что имеет значение. И обратно: если что-то имеет значение, то это "что-то" и есть выражение.

Основными формами выражений являются **константы** и **переменные**. Например, если вы записываете "$a = 100", вы присваиваете "100" переменной $a:

```
$a = 100;
```

В приведенном примере:

- $a - это переменная,
- = - это оператор присваивания,
- 100 - это и есть выражение, его значение 100.

Выражением может быть и переменная, если ей сопоставлено определенное значение:

```
$x = 7;

$y = $x;
```

В приведенном примере:

- в первой строке выражением является константа 7,
- во второй строке - переменная $x, т.к. ранее ей было присвоено значение 7; $y = $x также является выражением.

Немного более сложными **примерами выражений** являются **функции**. Если вы не знакомы с концепцией функций, то следующий пример может показаться не совсем понятным, но достаточно уловить саму суть идеи.

Пусть **мы имеем функцию** myFunc, принимающую в качестве аргумента дату рождения и вычисляющую по принятой дате возраст:

```
$age = myFunc("25.06.1990");
```

И пусть **мы имеем инструкцию** вида:

```
$age = 31;
```

Две приведенные записи **абсолютно эквиваленты**.

Поскольку myFunc возвращает 31, значением выражения myFunc("25.06.1990") является число 31 (на 04.01.2022 данное утверждение является верным).

> ***К сведению***. **Функции — это выражения**, значением которых является то, что **возвращает функция**. Как правило, функции возвращают не статическое значение, а некоторое вычисленное.

PHP поддерживает три типа **скалярных значений** (скалярными являются значения, которые вы не можете *разбить* на меньшие части, в отличие, например, от массивов):

1. **Целочисленные**.
2. **С плавающей точкой**.
3. **Строковые значения**.

PHP поддерживает также два **комбинированных типа** (не скалярных):

1. **Массивы**.
2. **Объекты**.

> ***Важно.*** Каждый из этих типов значений может **возвращаться функцией** или **присваиваться переменной**.
>
> Написанное **чрезвычайно важно** и будет использоваться нами при

> проектировании приложений!

Следующие нарезки кода демонстрируют написанное. В **каждом листинге** четыре действия:

1. **Описание** функции.
2. Возвращение функцией значения **определенного типа**.
3. Присваивание значения **переменной**.
4. **Вывод** на экран значения переменной.

Что НЕ нужно:

- Разбираться построчно с кодом (функции, массивы, объекты - отдельные большие темы).

Что нужно:

- Понять парадигму написанного. Уловить общий смысл, увидеть связь между написанным текстом и кодом.
- Набрать код самостоятельно, запустить, проверить работоспособность.

example_1. Функция возвращает целое число

```php
<?php
    // описание функции
    function myFun() {
        // функция возвращает скалярное значение -> Число
        return 100;
    }
    // вызов функции
    // переменная принимает возвращаемое функцией число
    $var = myFun();
    // проверяем, что находится в переменной
```

```
    var_dump($var);
?>
```

example_2. Функция возвращает число с плавающей точкой

```
<?php
    // описание функции
    function myFun() {
        // функция возвращает скалярное значение -> Число с плавающей точкой
        return 12.543;
    }
    // вызов функции
    // переменная принимает возвращаемое функцией число с плавающей точкой
    $var = myFun();
    // проверяем, что находится в переменной
    var_dump($var);
?>
```

example_3. Функция возвращает строку

```
<?php
    // описание функции
    function myFun() {
        // функция возвращает скалярное значение -> Строку
        return "Тьма, пришедшая со Средиземного моря, накрыла ненавидимый прокуратором город.";
    }
```

```
    // вызов функции
    // переменная принимает возвращаемое функцией строку
    $var = myFun();
    // проверяем, что находится в переменной
    var_dump($var);
?>
```

example_4. Функция возвращает массив

```
<?php
    // описание функции
    function myFun() {
        $arr = array("Pink Floyd", "Jethro Tull", "The Doors",
        "The Rolling Stones");
        // функция возвращает комбинированное значение ->
        Массив
        return $arr;
    }
    // вызов функции
    // переменная принимает возвращаемый функцией массив
    $var = myFun();
    // проверяем, что находится в переменной
    // для просмотра комбинированных значений желательно
    использовать тег <pre>,
    // лучший способ узнать почему, - закомментируйте тег и
    посмотрите что будет
    echo "<pre>";
    var_dump($var);
    echo "</pre>";
```

```
?>
```

example_5. Функция возвращает объект

```
<?php
    // описание функции
    function myFun() {
        // определяем Объект
        class Movie {
            public $name = "Пролетая над гнездом кукушки";
            public $director = "Милош Форман";
            public $producer = "Майкл Дуглас";
            public $favorite = "Джек Николсон";
        }
        // создать объект
        $obj = new Movie();
        // функция возвращает комбинированное значение ->
        Объект
        return $obj;
    };
    // вызов функции
    // переменная принимает возвращаемый функцией объект
    $var = myFun();
    // проверяем, что находится в переменной
    // для просмотра комбинированных значений желательно
    использовать тег <pre>,
    // лучший способ узнать почему, - закомментируйте тег и
    посмотрите что будет
```

```
    echo "<pre>";
    var_dump($var);
    echo "</pre>";
?>
```

Арифметические операторы

Описанные ниже **арифметические операторы** PHP работают по законам основ арифметики.

Пример	Название	Результат
-$a	Отрицание	Смена знака $a
$a + $b	Сложение	Сумма $a и $b
$a - $b	Вычитание	Разность $a и $b
$a * $b	Умножение	Произведение $a и $b
$a / $b	Деление	Частное от деления $a на $b
$a % $b	Деление по модулю	Целочисленный остаток от деления $a на $b
$a ** $b	Возведение в степень	Результат $a в степени $b

Операция деления ("/") всегда возвращает вещественный тип, даже если оба значения были целочисленными.

Операция вычисления остатка от деления "%" работает только с целыми числами, так что применение ее к дробным может привести к нежелательному результату. Остаток $a % $b будет негативным, для негативных значений $a.

Возможно использование скобок. Приоритет одних математических операций над другими и изменение приоритетов при использовании скобок в арифметических выражениях соответствуют обычным математическим правилам.

example_6. Арифметические операторы

```php
<?php
    echo "8 + 2 = ", 8 + 2; // 10, сложение
    echo "<p>";
    echo "10 - 2 = ", 10 - 2; // 6, вычитание
    echo "<p>";
    echo "6 * 2 = ", 6 * 2; // 16, умножение
    echo "<p>";
    echo "9 / 2 = ", 9 / 2; // 4, деление
    echo "<p>";
    echo "11 % 2 = ", 11 % 2; // 0, деление по модулю
    echo "<p>";
    echo "8 ** 2 = ", 8 ** 2; // 64, возведение в степень
?>
```

Определения:

- **Арифметическая операция** - сложение, вычитание, умножение и деление.
- **Унарная операция** - операция с одним операндом. Например, -3 — унарная операция для получения числа, противоположного числу три.
- **Бинарная операция** - операция с двумя операндами. Например, 3 + 9.

Логические операторы

Логические операторы предназначены исключительно для работы с **логическими выражениями** (подробнее в теме "Управляющие конструкции") и возвращают **false** или **true**.

Приведем таблицу логических операторов PHP:

Пример	Название	Результат
$a and $b	Логическое 'и'	**TRUE** если и $a, и $b **TRUE**
$a or $b	Логическое 'или'	**TRUE** если или $a, или $b **TRUE**
! $a	Отрицание	**TRUE** если $a не **TRUE**
$a && $b	Логическое 'и'	**TRUE** если и $a, и $b **TRUE**
$a || $b	Логическое 'или'	**TRUE** если или $a, или $b **TRUE**

Смысл двух разных вариантов для операторов "and" и "or" в том, что они работают с **различными приоритетами**.

Следует заметить, что вычисление логических выражений, содержащих такие операторы, идет всегда слева направо, при этом, если результат уже очевиден (например, **false && что-то** всегда дает **false**), то **вычисления обрываются**.

```php
<?php
    $res = (true && false); // false
    // аналогично
    $res = (true and false); // false
    // ----------
    $res = (true || false); // true
    // аналогично
    $res = (true or false); // true
    // ----------
```

```
$res = (false || 21); // true

$res = (false && 21); // false

$res = !true; // false
```

Строковые операторы

В PHP есть два оператора для работы со строками:

- оператор **конкатенации** ('.'), который возвращает объединение левого и правого аргумента.
- оператор **присвоения с конкатенацией**, который присоединяет правый аргумент к левому.

Приведу конкретный пример:

example_7. Строковые операторы

```
<?php

    $a = "Hello ";

    $b = $a . "World!<p>";

    // $b будет содержать строку "Hello World!"

    echo '$b = ' . $b;

    // $a будет содержать строку "Hello World!"

    $a .= "World!<p>";

    echo '$a = ' . $a;

    // $b будет содержать строку "Hello World!<p>Stop the World
    - I Want to Get Off..."

    $b .= "Stop the World - I Want to Get Off...";

    echo '$b = ' . $b;

?>
```

Задание № 8

==========

Реализуйте программу, которая вычисляет значение выражения:

8 / 2 + 5 - -18 / 6 / 2

Постарайтесь предварительно рассчитать результат без помощи современных гаджетов).

Определите **тип данных** выводимого значения. Для этого вывод организовать через функцию **var_dump**.

Задание № 9

==========

Реализуйте программу, которая вычисляет и выводит на экран результат выражения:

1 + 2 - 3 + 4

где:

1. "5 в степени 3"
2. "12 разделить на 3"
3. остаток от деления "30 на 4"
4. "минус 7 умножить на 2"

Результат **каждого** действия сохраните в **отдельной переменной**.

Задание № 10

==========

Рассчитайте значение каждого приведенного выражения:

- (952 / 4) * 3 - (476 / 7) + 196 **||** 0
- (952 / 4) * 3 - (476 / 7) + 196 **&&** 0

Напишите программу, которая вычисляет и выводит на экран результат выражения. Сверьте программный вывод результата с расчетным значением.

Если вывод и расчетное значение совпадают – **вы молодец**!

Если не совпадают, вы все равно молодец!

Операторы и выражения (2)

- Операторы присвоения
- Операторы сравнения
- Операции инкремента и декремента
- Операторы эквивалентности
- Оператор управления ошибками

Операторы присвоения

Базовый оператор присвоения обозначается как **=**.

На первый взгляд может показаться, что это оператор "равно". На самом деле это не так. В действительности, оператор присвоения означает, что левый операнд получает значение правого выражения.

Результатом выполнения оператора присвоения является само присвоенное значение. Таким образом, результат выполнения $a = 3 будет равен 3. Это позволяет использовать конструкции вида:

```
<?php
      $a = ($b = 4) + 5;
      // результат:
      // $a = 9
      // $b = 4
```

В дополнение к базовому оператору присвоения имеются **комбинированные операторы** для всех бинарных арифметических и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения.

Сложно? Нет, посмотрите примеры:

- **+=** Сложение с последующим присвоением результата:

```
$a = 12;
$a += 5;
echo $a; // равно 17
```

- **-=** Вычитание с последующим присвоением результата:

```
$a = 12;
$a -= 5;
echo $a; // равно 7
```

- ***=** Умножение с последующим присвоением результата:

```
$a = 12;
$a *= 5;
echo $a; // равно 60
```

- **/=** Деление с последующим присвоением результата:

```
$a=12;
$a /= 5;
echo $a; // равно 2.4
```

- **.=** Объединение строк с присвоением результата. Применяется к двум строкам. Если же переменные хранят не строки, а, к примеру, числа, то их значения преобразуются в строки и затем проводится операция:

```
$a = 12;
$a .= 5;
echo $a; // равно 125
// идентично
$b = "12";
$b .= "5"; // равно 125
```

- **%=** Получение остатка от деления с последующим присвоением результата:

```
$a = 12;
$a %= 5;
echo $a; // равно 2
```

- ****=** Получение результата от возведения в степень:

```
$a = 8;
$a **= 2;
echo $a; // равно 64 (8 в степени 2)
```

Обратите внимание, что присвоение копирует оригинальную переменную в новую (присвоение по значению), таким образом все последующие изменения одной из переменных на другой никак не отражаются.

example_1. Комбинированные операторы

```php
<?php
    $a = 3;
    $a += 5; // устанавливает $a = 8
    echo $a, "<p>";
    // аналогично $a = $a + 5;
    $b = 10;
    $b *= 2; // устанавливает $b = 20
    echo $b, "<p>";
    // аналогично $b = $b * 2;
    $c = "Hello ";
    $c .= "There!"; // устанавливает $c = "Hello There!;
    echo $c;
    // аналогично $c = $c . "There!";
?>
```

Операторы сравнения

Операторы сравнения, как это видно из их названия, позволяют сравнивать между собой два значения.

Это в своем роде уникальные операции, потому что независимо от типов своих аргументов они всегда возвращают одно из двух: **false** или **true**.

> ***К сведению***. Операции сравнения позволяют сравнивать два значения между собой и, если **условие выполнено**, возвращают **true**, а если нет — **false**.

В PHP разрешается сравнивать только скалярные переменные.

> ***К сведению***. Массивы и объекты в PHP **сравнивать нельзя**.

Так что удивившись как-то раз, почему два совершенно разных массива при сравнении их с помощью == оказываются вдруг одинаковыми, вспомните, что перед сравнением оба операнда преобразуются в слово **array**, которое потом и сравнивается.

Операторы сравнения:

Пример	Название	Результат
$a == $b	Равно	**TRUE** если $a равно $b
$a === $b	Тождественно равно	**TRUE** если $a равно $b и имеет тот же тип данных
$a != $b	Не равно	**TRUE** если $a не равно $b
$a <> $b	Не равно	**TRUE** если $a не равно $b
$a !== $b	Тождественно не равно	**TRUE** если $a не равно $b или в случае, если они разных типов
$a < $b	Меньше	**TRUE** если $a строго меньше $b
$a > $b	Больше	**TRUE** если $a строго больше $b
$a <= $b	Меньше или равно	**TRUE** если $a is меньше или равно $b
$a >= $b	Больше или равно	**TRUE** если $a больше или равно $b

example_2. Операторы сравнения

```php
<?php

    echo "<pre>";

    var_dump(3 == "2"); // false

    var_dump(2 == '2'); // true

    var_dump(3 != 2); // true

    var_dump(5 <> 5); // false

    var_dump(4 < 6); // true

    var_dump(9 > 2); // true

    var_dump(5 <= 6); // true

    var_dump(7 >= 9); // false

?>
```

Операции инкремента и декремента

Операции **инкремента** и **декремента** являются арифметическими операциями, но производятся над одним операндом.

> ***К сведению***.
>
> - **Инкремент** - (операция ++) увеличивает число на единицу.
> - **Декремент** - (операция --) уменьшает число на единицу.

PHP поддерживает **префиксные** и **постфиксные** операторы инкремента и декремента.

Пример	Название	Действие
++$a	Префиксный инкремент	Увеличивает $a на единицу и возвращает значение $a

$a++	Постфиксный инкремент	Возвращает значение $a, а затем увеличивает $a на единицу
--$a	Префиксный декремент	Уменьшает $a на единицу и возвращает значение $a
$a--	Постфиксный декремент	Возвращает значение $a, а затем уменьшает $a на единицу

Постфиксные операторы:

1. сначала **возвращают** значение переменной,
2. затем **увеличивают** или **уменьшают** значение этой переменной.

Пример:

```php
<?php
    $a = 10;
    $b = $a++;
    echo "a = $a, b = $b"; // выводит a=11, b=10
```

Сначала переменной $b присвоилось значение переменной $a, а уже затем переменная $a была инкрементирована (увеличена на единицу).

Префиксные операторы:

1. **увеличивают** или **уменьшают** значение переменной,
2. **возвращают** значение этой переменной.

Пример:

```php
<?php
    $a = 10;
    $b = --$a;
    echo "a = $a, b = $b"; // выводит a=9, b=9
```

Сначала переменная $a была декрементирована (уменьшена на единицу), а уже затем ее новое значение было присвоено переменной $b.

Операции инкремента и декремента на практике применяются очень часто. Например, они встречаются практически в любом цикле **for**.

example_3. Операции инкремента и декремента

```
<?php
      echo "<h3>Постфиксный инкремент</h3>";
      $a = 5;
      echo "Должно быть 5: " . $a++ . "<br />\n";
      echo "Должно быть 6: " . $a . "<br />\n";
      echo "<h3>Префиксный инкремент</h3>";
      $a = 5;
      echo "Должно быть 6: " . ++$a . "<br />\n";
      echo "Должно быть 6: " . $a . "<br />\n";
      echo "<h3>Постфиксный декремент</h3>";
      $a = 5;
      echo "Должно быть 5: " . $a-- . "<br />\n";
      echo "Должно быть 4: " . $a . "<br />\n";
      echo "<h3>Префиксный декремент</h3>";
      $a = 5;
      echo "Должно быть 4: " . --$a . "<br />\n";
      echo "Должно быть 4: " . $a . "<br />\n";
?>
```

Операторы эквивалентности

В PHP есть оператор **тождественного сравнения** — тройной знак равенства ===, или оператор проверки на эквивалентность.

PHP довольно терпимо относится к тому, что строки неявно преобразуются в числа, и наоборот.

Например, следующий код выведет, что значения переменных равны:

```
<?php
	$a = 10;
	$b = "10";
	if($a == $b) echo "a и b равны"; // выводит "a и b равны"
```

И это несмотря на то, что переменная $a представляет собой число, а $b — строку.

Теперь рассмотрим несколько другой пример:

```
<?php
	$a = 0; // ноль
	$b = ""; // пустая строка
	if($a == $b) echo "a и b равны"; // выводит "a и b равны"
```

Хотя $a и $b явно не равны даже в обычном понимании этого слова, скрипт заявит, что они совпадают. Почему так происходит?

Дело в том, что если один из операндов логического оператора может трактоваться как число, то оба операнда трактуются как числа. При этом пустая строка превращается в 0, который затем и сравнивается с нулем. Неудивительно, что оператор echo срабатывает.

Проблему решает **оператор эквивалентности ===** (тройное равенство).

> ***К сведению***. Оператор эквивалентности сравнивает не только значения, но также их **типы**.

Перепишем наш пример с использованием этого оператора:

```php
<?php
    $a = 0; // ноль
    $b = ""; // пустая строка
    if($a === $b) echo "a и b равны"; // ничего не выводит
```

Вот теперь ничего выведено не будет.

Для оператора === существует и его противоположность — оператор !==.

Оператор управления ошибками

PHP поддерживает **оператор управления (контроля) ошибками**: знак **@**. В случае, если он предшествует какому-либо **выражению** в PHP-коде, любые **сообщения об ошибках**, генерируемые этим выражением, будут проигнорированы.

До PHP версии 8.0.0 оператор **@** мог отключить критические ошибки, которые прекращали выполнение скрипта. Например, добавление @ к вызову несуществующей функции приводил к завершению сценария без указания причины.

Таким образом, следующий фрагмент кода ошибку бы не генерировал:

```php
<?php
    // пытаемся сделать то, что делать нельзя (сложить число и строку)
    // ошибка критическая
    $res = @(2 + 'z6');
    echo '$res = ', $res;
```

С версии 8.0 дела обстоят несколько иначе. Выполняем пример, разбираемся.

example_4. Подавление ошибки

```php
<?php
    // error_reporting(0);
    // пытаемся подключить файл которого не существует
    // ошибка не критическая
    $content1 = @include "content1.txt";
    // пытаемся подключить файл которого не существует
    // ошибка критическая
    $content2 = @require "content2.txt";
    // пытаемся сделать то, что делать нельзя (сложить число и строку)
    // ошибка критическая
    $res = @(2 + 'z6');
    echo '$res = ', $res;
?>
```

Критические ошибки можно *подавить* только инструкцией **error_reporting**().

> ***Примечание***. **Оператор @** работает только с **выражениями**. Простое практическое правило таково, если можно взять значение чего-то, то можно добавить к нему оператор **@**.

Например, оператор **@** можно добавлять к переменным, вызовам функций, вызовам определенных языковых конструкций (например, include) и т. д. Его нельзя добавлять к определениям функций или классов, а также к условным структурам, таким как if, foreach и т. д.

> ***Важно***. Подавлять ошибку, происхождение которой неизвестно, очень **плохая идея**. Использовать оператор можно, либо в отладочных

> целях, либо в ситуации, когда физика происхождения ошибки известна, и ею можно пренебречь.

Задание № 11

==========

Дан фрагмент кода.

```
$x = 3;

$result = 8 / 2 + 5 - -9 / $x++;
```

Рассчитайте **значение выражения** без **использования калькулятора**.

Напишите программу, выведите результат.

Задание № 12

==========

Дан фрагмент кода.

```
$x = 6;

$y = 19;

$result = 8 / 2 + 5 - -(--$y) / $x++;
```

Рассчитайте **значение выражения** без **использования калькулятора**.

Напишите программу, выведите результат.

Задание № 13

==========

Дан фрагмент кода.

```
$a = 5;

$b = 10;

$c = ++$a + ($a / $b);

$d = ($c++ - $b) + $a++;
```

Рассчитайте **значение выражения устно**.

Приоритет операций

- Коммутативная операция
- Композиция операций
- Приоритет операций

Коммутативная операция

От перемены мест слагаемых сумма не меняется. Это один из базовых и интуитивно понятных законов арифметики, он называется **коммутативным законом**.

Бинарная операция считается коммутативной, если, поменяв местами операнды, вы получаете тот же самый результат. Очевидно, что сложение — коммутативная операция: 3 + 2 = 2 + 3.

А вот является ли коммутативной операция вычитания? Конечно, нет: 2 - 3 ≠ 3 - 2. В программировании этот закон работает точно так же, как в арифметике.

Более того, большинство операций, с которыми мы будем сталкиваться в реальной жизни, не являются коммутативными. Отсюда вывод: **всегда обращайте внимание на порядок того, с чем работаете**.

Это кажется простым. Но когда вы пишите более менее сложную программу и при этом подключаете файл с функциями, открываете соединение с базой данных, создаете дескриптор на файл и инициализируете сокет-соединение, порядок выполнения операций имеет значение.

> ***Важно***. **Коммутативность** - свойство операции, когда изменение порядка операндов не влияет на результат.

Композиция операций

А что, если понадобится вычислить такое выражение: 3 * 5 - 2? Именно так мы и запишем:

```
<?php

    print(3 * 5 - 2); // 13
```

Интерпретатор производит арифметические вычисления в **правильном порядке**: сначала деление и умножение, потом сложение и вычитание. Иногда этот порядок нужно изменить — об этом в следующем разделе.

Или другой пример:

```
<?php

    // 2 * 4 * 5 * 10 = 8 * 5 * 10 = 40 * 10 = 400

    print(2 * 4 * 5 * 10);
```

Операции можно соединять друг с другом, получая возможность вычислять все более сложные составные выражения. Такое свойство операций называется **композицией**. Композиция арифметических операций в программировании аналогична композиции из школьной программы. Композиция операций распространяется на вообще все операции, а не только арифметические.

> ***К сведению***. **Компози́ция** (лат. compositio — составление, связывание, сложение, соединение) — составление целого из частей.

Приоритет операций

В школьной математике мы изучали понятие **приоритет операции**. Приоритет определяет то, в какой последовательности должны выполняться операции. Например, умножение и деление имеют больший

приоритет, чем сложение и вычитание, а приоритет возведения в степень выше всех остальных арифметических операций:

2 ** 3 * 2 = 16.

Но нередко вычисления должны происходить в порядке, отличном от стандартного приоритета. В сложных ситуациях приоритет можно (и нужно) задавать круглыми скобками, точно так же, как в школе, например: (2 + 2) * 2.

Скобки можно ставить вокруг любой операции. Они могут вкладываться друг в друга сколько угодно раз:

```
<?php
    print(3 ** (4 - 2)); // 9
    print(7 * 3 + (4 / 2) - (8 + (2 - 1))); // 14
```

Главное при этом соблюдать парность, то есть закрывать скобки в правильном порядке. Это, кстати, часто становится причиной ошибок не только у новичков, но и у опытных программистов. **Для удобства ставьте сразу открывающую и закрывающую скобку**, а потом пишите внутреннюю часть. Это касается и других парных символов, например, кавычек.

Иногда выражение сложно воспринимать визуально. Тогда можно расставить скобки, **не повлияв на приоритет**. Например, следующее выражение можно сделать немного понятнее, если расставить скобки.

```
<?php
    // было
    print(8 / 2 + 5 - -3 / 2); // 10.5
    // стало
    print(((8 / 2) + 5) - (-3 / 2)); // 10.5
```

Код пишется для людей, потому что код будут читать люди, а машины будут только исполнять его. Для машин код — или **корректный**, или **не**

корректный, для них нет "более" понятного или "менее" понятного кода. Явная расстановка приоритетов упрощает чтение вашего кода другими разработчиками (или вами, но через некоторый промежуток времени).

Задание № 14

==========

Дано выражение:

5 ** 9 - 6 + 8 / 4 % 2

Создайте **приоритет**, чтобы действия:

- **9 – 6**
- **8 / 4**

высчитывались в первую очередь.

Предварительно рассчитайте результат выражения **без использования** калькулятора.

Напишите сценарий, выведите результат.

Задание № 15

==========

Дан **фрагмент кода**. **Рассчитайте** вывод **каждого оператора echo** без использования калькулятора и прочих подручных средств.

Напишите программу, сверьте выводы программы с **расчетными** значениями.

```
$x = 10;

$y = $r = $x--;

$t = ( ++$r + $y-- ) % --$x;

echo "x = $x </br>";

echo "y = $y </br>";

echo "r = $r </br>";

echo "t = $t </br>";
```

Задание № 16

==========

Дан **фрагмент кода**. **Рассчитайте** вывод **каждого оператора echo** без использования калькулятора и прочих подручных средств.

Напишите программу, сверьте выводы программы с **расчетными** значениями.

```
$x = 5;
$y = 2;

$r = --$x / $y;

echo "r = $r </br>";
echo "x = $x </br>";
echo "y = $y </br>";
echo "<p>";

$t = ( $r++ / $y ) / $x--;

echo "t = $t </br>";
echo "r = $r </br>";
echo "x = $x </br>";
echo "y = $y </br>";
echo "<p>";

$s = ( ( $x - $y-- ) / $t++ ) + $r;

echo "s = $s </br>";
echo "t = $t </br>";
echo "r = $r </br>";
echo "x = $x </br>";
echo "y = $y </br>";
```