연산자

연산자는 데이터와 데이터를 가지고 또 다른 데이터를 만들어내는 역할을 한다. 흔히 볼 수 있는 사칙연산 `+ - * /` 등의 연산자가 대표적으로 존재한다.

산술연산자

산술연산자는 숫자와 숫자 데이터를 가지고 사칙연산을 하는 연산자다.

```
1 + 1; // 2
1 * 5; // 5
3 - 2; // 1
10 / 5; // 2
```

연산자로 결합된 데이터는 그 계산 결과에 따라 새로운 데이터를 만들어낸다. 이 데이터는 당연하지만 변수에 저장할 수 있다.

```
$sum = 1 + 2 + 3 + 4;
```

논리연산자

논리연산자는 결과값으로 `true` `false`를 만들어낸다. 숫자 크기의 비교 혹은 문자열이 같은지 다른지 비교를 할 때 주로 사용된다.

```
10 > 5; // true
10 < 5; // false
```

크기를 비교하는 `> < >= <=` 는 설명이 없어도 잘 알 수 있을 것이다.

php에서는 같음, 다름을 비교하는 `== === != !==`가 존재하는데, 개인적으로는 `===` 와 `!==`만 쓰기를 권장한다.

`==`와 `!=`는 느슨한 데이터 타입으로 비교를 하기 때문에, 예를 들어 `1 == '1'`과 같이 숫자 1과 문자열 1이 같은지 비교를 하는 경우, `true`값이 리턴된다. `===`를 사용하면 데이터 타입까지 비교를 하기 때문에, 좀 더 예측가능한 결과를 얻을 수 있다.

부정연산자

부정연산자 `!`는 데이터 앞에 붙는 경우, 해당 데이터가 `truthy`하면 `false`를, `falsy`하면 `true`를 리턴한다.

php의 모든 데이터는 `true` 또는 `false`로 평가될 수 있다. 이를 `truthy`하다, `falsy`하다 라고 표현을 하는데, 이는 논리를 평가하는 `if`문이나 `while`문 등에서 사용된다. 예를 들어서

```
if (0) {
  echo 'true';
} else {
  echo 'false';
}
```

위의 경우, 숫자 0은 `true` `false`는 아니지만 `if`문 안에서 `true`또는 `false`로 평가가 된다. 숫자 `0`의 경우, `falsy`하므로 `else`구문이 실행될 것이다.

거의 모든 데이터는 `true`로 평가되지만, `false`로 평가되는 몇몇 데이터들이 존재한다.

- 0
- false
- '' (빈 문자열)
- "" (빈 문자열)
- null

대표적인 `falsy`한 데이터는 위와 같다.

증감연산자

증감연산자는 `++`와 `--`가 존재한다. 이는 해당 데이터 값에 1을 더하거나 1을 빼는 작업을 편하게 해주는데, 이게 데이터 앞에붙냐 뒤에붙냐에 따라 코드 동작이 달라진다.

전위연산자

전위연산자는 앞에 붙는 연산자로, 일단 데이터에 값을 빼거나 더한 후 코드가 실행된다.

```
$i = 10;

echo --$i; // 9
```

일단 `$i`변수에 들어있는 값을 1 뺀 다음에, 코드가 실행되어 `echo 9;` 가 찍힌다.

후위연산자

전위연산자와는 반대로, 일단 코드가 실행된 후 데이터에 값을 빼거나 더한다.

```
$i = 10;

echo $i--; // 10
echo $i; // 9
```

위의 경우, `echo $i--`는 전위연산자인 `echo --$i`와는 다르게, 코드가 먼저 실행되므로 `$i`의 기존값이 출력된다. 그 다음 비로소 1이 줄어들기 때문에, 그 다음에 echo를 찍어야 비로소 9라는 값이 나오는 것이다.

문자열 결합 연산자

문자열 결합연산자는 문자열을 만드는 연산자이다.

```
echo 'Hello' . ' World'; // Hello World
```

두 문자열인 Hello와 World를 연결하였다.

원래 echo는 단 하나의 문자열만 출력할 수 있다. 즉, 하나의 문자열 데이터만 출력할 수 있다. 그런데 보통 변수가 여러개가 존재하고, 이 변수의 값들을 이용해서 문자열을 출력하고자 하는 경우가 많으므로 이런 문자열 결합 연산자인 .을 이용해서 문자열을 출력하는 것이다.

이렇게 문자열 결합 연산자를 이용할 때, 데이터가 문자열이 아닌 숫자인 경우엔, 해당 숫자가 문자열 숫자로 변환된다. 즉,

```
$num1 = 1;
$num2 = 2;

$result = $num1 . $num2; // '12'
```

위의 경우, $num1의 1과 $num2의 2가 합쳐져서 12라는 문자열이 된 것이다.

이렇게 문자열 연산자를 이용하면 굳이 새로운 문자열 데이터를 직접 하드코딩하지 않아도 된다.

```
$name = 'John';
$age = 100;

$greeting1 = 'John님은 나이가 100살 이십니다.';
$greeting2 = $name . '님은 나이가 ' . $age . '살 이십니다.';

echo $greeting2;
// 또는
echo $name . '님은 나이가 ' . $age . '살 이십니다.';
```

이렇게 그냥 변수 자체로 데이터를 만들면 된다. 데이터는 보통 프로그래머가 정하는게 아니라, 사용자의 입력값에 따라 달라지므로, 이런 데이터를 변수에 담아서 코딩해서 표현하는 것이다.

표현식

코드는 크게 표현식인 expression과 구문인 statement로 구분되어 있다.

표현식은 결과가 데이터이다. 구문은 제어구조나 변수 대입 등과 같은 것이다.

```
1 + 1; // 표현식

function returnOne() {
    return 1;
}

returnOne(); // 함수를 호출하는 것도 표현식이다.
```

함수를 호출하는 것도 표현식이다. 보통 함수는 데이터를 리턴하기 때문이다.

표현식은 보통 단독으로 쓰이지 않는다. 당연히 구문과 함께 쓰인다.

대표적으로, `echo`는 응답데이터에 문자열을 프린트하는 구문이다.

구문

구문은 표현식이 아닌 것들인데, 변수를 대입하거나 제어구조, 함수 선언 등을 뜻한다. 즉, 결과 값이 딱히 존재하지 않는 것들이 구문이다.

변수 대입

```
$a = 10;
```

변수 `$a`에 10을 대입했다. 변수에 값을 넣는 것은 딱히 어떤 결과값이 존재하지 않는다.

함수 선언

```
function hello() {
    return 'hello';
}
```

함수를 선언했다. 함수를 선언했다고 딱히 어떤 결과값이 존재하는 것은 아니다.

if

`if`는 조건문이다.

```
if (condition expression) {
    // truthy
} else {
    // falsy
}
```

보면 알겠지만, 괄호 안에는 어떤 조건 표현식이 들어간다. 이 때, 이 표현식의 데이터는 `truthy` 혹은 `falsy`하게 평가가 된다.

보통 대부분의 값은 `truthy`하게 평가가 되지만, 일부 값은 `falsy`하게 평가가 된다. 대표적인 값들을 보자면

- false
- 0
- ''
- ""
- null

위와 같은 값들이 존재한다.

조건에 따라 블록 안의 구문들이 실행된다.

이 때, 조건 표현식에 변수를 대입하는 구문을 사용할 수도 있는데, 예를 들어

```
if ($a = true) {
    // truthy
}
```

위의 경우, 우선 `$a = true` 구문이 실행된다. 그리고 `$a`값이 평가가 된다. 즉, 결국 `true`이므로 실행될 것이다.

while

`while`은 기본적인 반복문이다.

```
while (condition expression) {
    // truthy
}
```

`while`의 블록 역시 `if`와 마찬가지로 조건 표현식이 들어간다. 이 조건 표현식이 `truthy`하면, 블록이 실행된다. 블록 내부에서 `break`를 하거나, 또는 조건 표현식이 `falsy`한 값이 되지 않는한 `while`문은 계속 돌아간다. 이는 잘못하면 무한루프를 돌게해서, 시스템이 멈추게 될 수도 있다. 따라서 `while`문을 사용할 때에는 적절하게 조건 표현식에 들어가는 값을 `false`하게 바꿔줘서 루프를 빠져나오게 하거나, 블록 안에서 `break`를 사용해서 빠져나오게 구성해야 한다.

```
while ($row = mysqli_fetch_assoc($result)) {
    echo $row['id'];
}
```

위의 예제의 경우, `mysqli_fetch_assoc` 함수가 실행될 때마다 어떤 데이터가 `$row`변수로 리턴되고, `$row`값이 평가가 된다. 이 때, `$row`값이 `falsy`한 값으로 평가가 되는 경우 `while`문은 작동을 멈출 것이다. 위의 경우, 모든 데이터를 꺼내온 경우 `while`문은 작동을 그만둘 것이다.

for

for문은 while문의 진화형태로, 반복문은 보통 몇 번 반복하느냐를 결정하는게 중요한데, while문은 이를 적용하기가 조금 까다롭다. 보통은 블록 안에서 항상 if문으로 break를 시키냐 마냐로 무한루프를 피하는데, 이럴 때 대개 어떤 변수 값 하나를 가지고 증감을 시키면서 평가한다. 예를 들어, 1부터 10까지 while문으로 echo를 찍는다고 해보자.

```
$i = 1;
while (true) {
    echo $i;
    $i++;
    if ($i === 10) {
        break;
    }
}
```

이렇게 구성하거나, 혹은

```
$i = 1;
while ($i <= 10) {
    echo $i;
    $i++;
}
```

이렇게 구성할 것이다. for문으로 구성하면 이를 좀 더 깔끔하게 구성할 수 있다.

```
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
```

for문에는 초기화 영역, 조건 영역, 증감영역을 정의할 수 있게 되어있으므로 while문처럼 코드가 지저분해지지도 않고, 알아보기도 쉬워진다.

```
for (초기화 영역; 조건 영역; 증감문 영역) {
    // truthy
}
```

초기화 영역은 for문이 실행될 때 단 한번만 실행된다. 그리고 바로 조건 영역으로 조건을 테스트한뒤, 해당 조건이 truthy하면 블록이 실행되고, 그 다음 증감문 영역이 실행된다. 다시 정리해보면

1. 초기화 영역에서 변수 값을 초기화한다.
2. 조건 영역에서 조건을 판단한다.
3. truthy한 경우, 블록이 실행된다.
4. 블록 실행이 끝나고 나면, 증감문 영역이 실행된다.

5. 2번부터 다시 반복한다.

foreach

`foreach`는 `array`나 `object`에 사용되는 반복문으로, `for`문과는 다르게, `array`나 `object`가 갖고 있는 데이터 만큼 반복문을 돌린다.

Normal array

기본적인 배열, 즉 숫자 인덱스로 되어 있는 배열을 `for`문과 `foreach`를 이용해 데이터를 뽑아와보자.

```
$arr = [0, 1, 2, 3, 4, 5];

for ($i = 0, $length = count($arr); $i < $length; $i++) {
    echo $arr[$i];
}
```

`count($arr)`로 우선 `$arr`의 배열의 길이를 얻어왔다. 그리고 배열의 길이인 6만큼 블록을 실행하면서, 배열 인덱스로 배열의 값을 출력하게 했다.

```
$arr = [0, 1, 2, 3, 4, 5];

foreach ($arr as $value) {
    echo $value;
}
```

`foreach`의 경우, 배열이 갖고 있는 데이터 갯수만큼 알아서 반복해서 데이터를 가져온다. 한번 반복될 때마다, 배열의 인덱스 0부터 존재하는 데이터를 `$value`로 가져와서 출력한다.

Association array

연관 배열의 경우, `key`값과 `value`값이 존재한다.

```
$arr = [
  'key1' => 1,
  'key2' => 2,
];

foreach ($arr as $key => $value) {
    echo "{$key} 안에 {$value} 값이 들어 있습니다.";
}
```

연관배열의 경우에는, `for`문으로 가져오기가 쉽지 않다. `for`문은 대개 숫자값을 돌리는데, 연관배열은 숫자 인덱스로 값이 저장되어 있는게 아니기 때문이다. 따라서 이렇게 `foreach`를 이용해 값을 가져와야 한다.

Object

객체의 경우, 객체의 프로퍼티만큼 foreach가 순회된다.

```
$obj = new stdClass();

$obj->key1 = 1;
$obj->key2 = 2;

foreach ($obj as $key => $value) {
    echo "{$key} 프로퍼티 안에 {$value} 값이 들어 있습니다.";
}
```

연관배열과 비슷하다.

함수

함수는 반복되는 코드를 줄여주는, 프로그래밍에서 아주 중요한 요소다. 미리 정의된 코드를 실행하는데, 이 때 필요한 데이터들을 받아서 코드 실행의 결과 데이터를 반환해준다.

함수 선언은 다음과 같이 한다.

```
function plus($arg1, $arg2) {
    $result = $arg1 + $arg2;
    return $result;
}

echo '1 + 1은 ' . plus(1, 1) . ' 입니다.';
```

함수는 function 키워드로 선언을 하며, 그 뒤에 함수의 이름을 정의한다. 그리고 괄호를 열고, 함수에서 사용되는 데이터가 할당되는 변수들을 선언한다. 그리고 내부에서 어떤 코드를 실행하고, return 키워드로 데이터를 리턴하면 된다.

변수의 범위

언어마다 함수가 존재하는데, 이 때 함수에서 사용되는 변수의 범위개념이 있다. 전역변수, 지역변수라고 보통 많이들 부르는데, php는 기본적으로 php에서 정의되는 슈퍼글로벌 변수들, 즉 $_로 시작되는 변수들을 제외하고는 특별한 키워드가 없는 이상, 함수 내부는 함수 외부와 완전히 격리되어있다.

예를 들어, 자바스크립트의 경우

```
var a = 10;

function callA() {
    console.log(a);
}
```

```
callA(); // 10
```

위와 같이 함수 내부에서 사용되는 변수값이 함수 내부에 존재하지 않으면, 외부 범위에서 다시 한번 찾는데 이 때 이 변수 a는 전역변수라고 볼 수 있다. 만일 함수 내부에서 사용되는 변수값이 존재하면, 당연히 그 변수 값이 사용될 것이다.

```
var a = 10;

function callA() {
    var a = 20;
    console.log(a);
}

callA(); // 20
```

이렇게 함수 내부에서 선언되는 변수는 지역변수라고 한다. php는 그런데 특별한 키워드가 없는 이상, 함수 내부는 외부와 완전히 격리되어 있으므로

```
$a = 10;

function callA() {
    echo $a;
}

callA(); // ERROR! 선언되지 않은 변수
```

이렇게 변수 $a를 외부에서 찾지 않는다. 함수 내부에 존재하지 않은 변수이므로 바로 에러가 발생한다.

만일 외부 변수를 참조하려면

```
$a = 10;

function callA() {
    global $a; // 전역변수 $a를 가져온다.
    echo $a;
}

callA(); // 10
```

위와 같이 global이라는 키워드를 사용해야 한다.

매개변수

매개변수는 위에서도 설명했듯이, 함수를 선언할 때 ()괄호 안에 선언되는 변수를 매개변수라고 한다.

```
function myFunc($arg1, $arg2) {
    return $arg1 + $arg2;
}
```

위의 myFunc함수는 매개변수로 $arg1 $arg2가 존재한다. 정의된 함수를 호출할 때, 매개변수에 값을 넘겨주면 함수 내부에서는 해당 변수에 전해진 값이 들어있을 것이므로 그 값을 이용해 코드를 실행한다.

call by value & call by reference

직역하면 값으로써의 호출, 참조로써의 호출인데 프로그래밍 언어의 데이터 타입마다 값 그 자체인지, 혹은 참조값인지 구분된다.

자바스크립트의 경우 Number String Boolean 등을 Primitive Value라고 표현을 하는데, 즉 원시값이라고 표현을 하는데, 이런 값들의 특징은 값 자체가 존재한다는 점에 있다. 이게 무슨 말이냐면

```
// 변수 a를 선언한다.
var a = 0;

// 변수 b를 선언한다.
var b = 1;

b = 100;

// a에 b 값을 대입한다.
a = b;

// b의 값을 바꾼다.
b = 200;

console.log(a); // 100
```

위는 당연하겠지만, a = b가 실행되었을 때 a에는 그 당시 b에 있던 값이 들어간다. 따라서 나중에 b의 값을 바꾼다고 하더라도, a의 값은 그대로이다. 하지만

```
// 배열로 선언하고, 0번째 인덱스에 0을 넣는다.
var a = [0];

var b = [1];

// 변수 a에 변수 b의 값을 대입한다.
a = b;

b[0] = 100;

console.log(a[0]); // 100
```

분명 b의 값에 100을 대입했는데, a의 값도 바뀌어있다. 이것이 바로 call by reference 참조로써의 호출이다.

즉 이렇다. 어떤 데이터 타입들은 값 자체가 존재하므로, 변수에 값을 할당하면 값 자체가 복사되어 들어간다고 볼 수 있다. 이런 값들을 call by value라고 표현한다.

그런데 어떤 데이터 값들은, 값이 생성될 때 이 값이 고유하게 작동하고, 변수에 대입하면 변수들은 그저 이 값을 가리키게 된다. 즉, 데이터 자체를 참조한다는 말이다.

```
var a = [0]; // 여기에서 배열 [0]이 생성되고, 이 데이터 값은 어떤 메모리에 생성되었다. 그리고 변수는 단지 이 메모리값을 참조할 뿐이다.
var b = a; // 변수 b는 변수 a가 참조하는 값을 동시에 참조한다. 즉, 현재 변수 a와 b는 같은 데이터를 참조하고 있는 것이다!
```

자바스크립트의 경우, Object들은 모두 참조로써 작동을 하는데, 자바스크립트 특성상 Array Function 역시 객체이므로 배열, 함수는 모두 참조로써 작동을 하게 된다. 그렇다면 php는 어떨까?

결론부터 말하면 php에서 참조로써 작동하는 값은 객체인 object 즉 new 연산자로 생성되는 객체들만 그렇고, 자바스크립트와는 다르게 배열 역시 call by value로써 작동한다.

이는 함수 호출에도 아주 지대한 영향을 끼치는데, call by value로 작동하는 값들은 매개변수에 값 자체가 복사되므로, 외부의 변수값이 내부 함수에 의해 변경되지 않는다. 즉,

```
function changeValue($value) {
    $value = 100;
}

$a = 10;

chagneValue($a);

echo $a; // 10
```

당연하지만 $a의 값은 함수에 의해 바뀔리가 없다. 하지만 참조로써 작동하는 객체를 넘기면 어떻게 될까?

```
function changeValue($value) {
    $value->a = 100;
}

$obj = new stdClass();
$obj->a = 10;

changeValue($obj);

echo $obj->a; // 100
```

이렇게 php에서는 객체가 `call by reference`로 작동하기 때문에, 함수에 의해 변수 값이 변경이 된다는 것이다.

&연산자

php에서는 객체 외에는 전부 `call by value`로 작동한다고 했다. 그런데 `&`연산자를 이용하면, 강제로 `call by reference`로 만들 수 있다.

```
function changeValue(&$value) {
  $value = 100;
}

$a = 10;

changeValue($a);

echo $a; // 100
```

이는 변수가 참조로써 전달되었기 때문에 가능한 일이다.