개요

PHP (PHP: Hypertext Preprocessor)는 웹 개발 용으로 널리 사용되는 범용 오픈 소스 스크립트 언어로서 웹 페이지(HTML) 내부에 삽입될 수 있습니다. PHP의 문법은 C언어의 문법과 매우 유사합니다. PHPoC (PHP on Chip)는 이러한 PHP를 기반으로 솔내시스템(주)에서 자체적으로 개발한 스크립트 언어입니다.

PHPoC는 임베디드 시스템의 특성에 따른 제한사항들을 제외하고는 기본적으로 PHP 문법과 동일합니다. 따라서 전문 개발자들은 물론 누구나 쉽게 사용할 수 있습니다.

※ PHPoC는 PHP와 유사하게 만들어졌지만 모든 문법이 완전히 호환되지는 않습니다. 이 문서에서는 PHPoC와 PHP 사이의 차이점을 다루지 않으므로 이에 관한 내용은 PHPoC vs PHP에서 참고하십시오.

문법

- 기본 문법
- 자료 형 (Types)
- 변수 (Variable)
- 상수 (Constants)
- 연산자 (Operators)
- 제어 구조 (Control Structures)
- 함수 (Functions)
- 클래스와 객체 (Classes and Objects)
- 네임스페이스 (Namespace)

기본 문법

PHPoC 태그

PHPoC 스크립트는 스크립트의 시작을 의미하는 시작태그(<?php 또는 <?)와 종료를 의미하는 종료태그(?>)로 구성됩니다. 이 태그 밖의 모든 텍스트는 PHPoC 파서(parser)에 의해 무시되어 표준 출력 포트로 전송됩니다. 웹 페이지의 경우 웹 브라우저로 전송됩니다.

```
<?php                   // 시작태그
  echo "Hello PHPoC!";  // 스크립트
?>                      // 종료태그
```

웹 페이지(HTML)에 PHPoC 스크립트 삽입하기

PHPoC 파서가 시작태그와 종료태그 밖의 모든 텍스트를 무시하므로 PHPoC 스크립트는 웹 페이지(HTML)에 삽입해서 사용할 수 있습니다.

```
<p>This will be ignored by PHPoC and displayed by the browser. </p>
<?php echo "While this will be parsed."; ?>
<p>This will also be ignored by PHPoC and displayed by the browser. </p>
```

단, 조건절을 이용하면 PHPoC 태그 밖의 코드에 대해서도 처리할 수 있습니다. 아래의 예제를 보시기 바랍니다.

```
<?php if(true){ ?>
This will show if the expression is true. <!-- 이것은 출력 됩니다 -->
<?php } else { ?>
Otherwise this will show.                 <!-- 이것은 출력 되지 않습니다 -->
<?php } ?>
```

명령 행 구분하기

PHPoC의 각 명령 행은 세미콜론으로 구분합니다. 단, PHPoC 스크립트 블록의 맨 마지막 줄 또는 단일 행 명령인 경우 세미콜론을 생략할 수 있습니다.

```
<?php
  echo "the first statement.₩r₩n";          // 첫 번째 줄, 세미콜론 사용
  echo "the last statement.₩r₩n"             // 마지막 줄, 세미콜론 생략 가능
?>
<?php echo "single line statement.₩r₩n" ?>  // 단일 행, 세미콜론 생략 가능
```

※ 세미콜론 생략은 문법적으로 올바르지 않으므로 항상 사용할 것을 권장합니다.

주석

PHPoC는 단일 행 주석과 다중 행 주석을 모두 지원합니다.

```
<?php
  echo "the first statement.₩r₩n";          // 이것은 단일 행 주석입니다.
  /* 이것은
  다중 행
  주석입니다. */
  echo "the last statement.₩r₩n";
?>
```

※ PHP 및 유닉스 등에서 사용되는 단일 행 주석인 #은 PHPoC에서 지원하지 않습니다.

자료 형 (Types)

PHPoC는 진리 값, 정수형, 부동소수점 수, 문자열 그리고 배열의 5가지 자료 형을 지원합니다.

- 진리값 (Boolean)
- 정수 (Integer)
- 부동소수점 수 (Floating Point Numbers)
- 문자열 (String)
- 배열 (Array)
- 자동 형 변환 (Type Juggling)

※ 객체(Objects) 및 NULL은 지원하지 않습니다.

진리값 (Boolean)

가장 간단한 자료 형으로 참과 거짓 두 가지 값을 표현합니다. 참은 TRUE로, 거짓은 FALSE로 표현되며 대소문자의 구별은 하지 않습니다.

다른 자료형에서 진리 값으로의 명시적 형 변환을 위한 캐스트 연산자는 (bool) 또는 (boolean) 이며 역시 대소문자의 구별은 하지 않습니다.

```
<?php
  $bool_true = TRUE;              // 진리 값의 사용
  $int_test = 3;                 // 정수 3
  $bool_test = (bool)$int_test;  // 정수를 진리 값으로 형 변환
?>
```

진리 값으로의 형 변환에서 다음 값은 FALSE로 변환됩니다.

- 정수 0 (zero)
- 부동소수점 수 0.0 (zero)
- 빈 문자열 ("")

위 값을 제외한 모든 값은 TRUE로 변환됩니다. (문자열 "0" 포함)

정수 (Integer)

정수는 부호(+ 또는 -)를 포함하여 2진수, 8진수, 10진수, 16진수 네 가지 형태로 표현할 수 있습니다.

다른 자료 형에서 정수로의 명시적 형 변환을 위한 캐스트 연산자는 (int) 또는 (integer) 이며 대소문자의 구별은 하지 않습니다.

```
<?php
$octal = 010;                // 8 - 8진법
$decimal = 10;               // 10 - 10진법
$hexadecimal = 0x10;         // 16 - 16진법
$binary = 0b10;              // 2 - 2진법
$str_test = "10";            // 문자열 "10"
$int_test = (int)$str_test;  // 문자열을 정수로 형 변환
?>
```

부동소수점 수 (Floating Point Numbers)

부동소수점 수는 실수를 표현하는 방식으로 다음과 같이 표현될 수 있습니다.

```
<?php
$float0 = 3.14;             // 3.14
$float1 = 3.14e3;           // 3140
$float2 = 3.14E-3;          // 0.00314
?>
```

e3(또는 E3)은 10의 3제곱인 1000을 곱하는 것을 의미하고, e-3(또는 E-3)은 10의 -3제곱 즉, 10의 3제곱분의 1을 곱하는 것을 의미합니다.

다른 자료 형에서 부동소수점 수로의 명시적 형 변환을 위한 캐스트 연산자는 (float) 이며 대소문자의 구별은 하지 않습니다.

- 정확도
  부동소수점 수는 계산 시에 유의할 점이 있습니다. 컴퓨터는 0.1 또는 0.3 등의 간단해 보이는 수를 2진수로 계산하기 때문에 정확한 값을 표현할 수 없습니다.

```
<?php
$a = 0.1 / 0.3;
printf("%.20e₩r₩n", $a);   // 결과를 소수점 이하 20자리까지 출력
?>
```

```
[출력 결과]
3.33333333333333370341e-1
```

위 출력 결과에서 볼 수 있듯이 소수점 이하 15자리 밑으로는 값이 정확하지 않습니다. 이러한 한계점 때문에 부동소수점 수의 직접적인 비교 연산은 사용하지 않는 것이 좋습니다.

- NAN
  때때로 산술연산의 결과값으로 상수 NAN이 나올 수 있습니다. 이것은 부동소수점연산에서 확실하지 않거나 표현할 수 없는 수를 의미합니다. 이 상수와 자신을 포함한 다른 어떤 수의 값이 같은지를 비교하는 연산의 결과는 항상 FALSE가 됩니다.

```
<?php
$float0 = acos(2);
if($float0 == $float0)
    echo "True₩r₩n";          // if문 결과: FALSE
else
    echo "$float0₩r₩n";
?>
```

```
[출력 결과]
NAN
```

- INF
  상수 INF는 부동소수점 연산에서 표현할 수 있는 범위를 넘어가는 수를 의미합니다.

```
<?php
$float0 = 1.8E+309;
echo "$float0";
?>
```

```
[출력 결과]
INF
```

문자열 (String)

문자열은 일련의 문자들이 연속적으로 나열된 형태입니다. 문자열은 큰 따옴표 또는 작은 따옴표를 사용해 표기합니다.

다른 자료 형에서 문자열로의 명시적 형 변환을 위한 캐스트 연산자는 (string) 이며, 대소문자의 구별은 하지 않습니다.

```
<?php
$int_test = 10;                         // 정수 10
$str_test = (string)$int_test;           // 정수를 문자열로 형 변환
?>
```

- 작은 따옴표를 사용한 문자열
  가장 간단하게 문자열을 표기하는 방법은 작은 따옴표 안에 표현하는 것입니다. 작은 따옴표로 된 문자열 안에서 작은 따옴표(')' 또는 역 슬래시(₩)를 처리하기 위해서는 앞에 역 슬래시를 사용합니다. 이 두가지를 제외한 나머지 문자는 문자 그대로 표현됩니다.

```
<?php
echo 'This is a simple string';
echo "₩r₩n";
echo 'insert
newlines';
echo "₩r₩n";
echo 'specify ₩' (single quotation)';
echo "₩r₩n";
echo 'specify ₩₩ (back slash)';
echo "₩r₩n";
echo 'specify ₩ (back slash)';
echo "₩r₩n";
echo 'nothing happened ₩r₩n';
echo "₩r₩n";
echo 'nothing $a happened';
?>
```

```
[출력결과]
This is a simple string
insert
newlines
specify ' (single quotation)
specify ₩ (back slash)
specify ₩ (back slash)
nothing happened ₩r₩n
nothing $a happened
```

- 큰 따옴표를 사용한 문자열
  문자열을 큰 따옴표 안에 표현하는 방법입니다. 큰 따옴표 방식은 더 많은 특수문자를 처리할 수 있습니다. 특수문자의 처리는 역 슬래시(₩)를 사용합니다. 사용할 수 있는 특수문자는 다음과 같습니다. 이들을 제외한 나머지 문자는 문자 그대로 표현 됩니다.

특수문자	의미
₩n	개행문자(줄 바꿈)
₩r	캐리지 리턴
₩t	탭 간격 띄움
₩₩	역 슬래시
₩"	따옴표
₩$	달러 표시
₩[0-7]{8진수}	문자의 8진수 표기
₩x[0-9][A-F][a-f]{16진수}	문자의 16진수 표기

```
<?php
echo "This is a simple string";
echo "₩r₩n";
echo "insert ₩r₩n newlines";
echo "₩r₩n";
echo "Specify ₩" (Double quotation)";
?>
```

```
[출력결과]
This is a simple string
insert
newlines
Specify " (Double quotation)
```

※ 특수문자 ₩e, ₩v, ₩f에 대한 처리는 지원하지 않습니다.

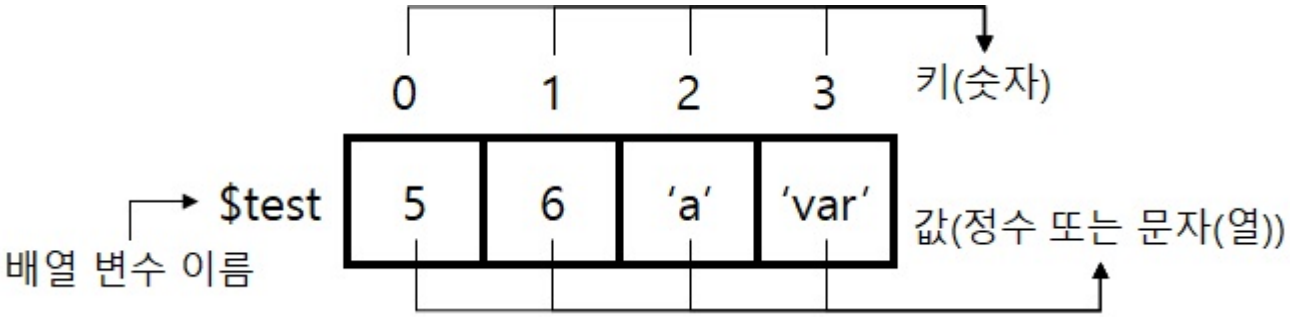
뿐만 아니라 이 방식에서는 문자열 내부에 변수를 처리할 수 있습니다.

```
<?php
$a = "a variable";
echo "Process $a";
?>
```

```
[출력결과]
Process a variable
```

배열 (Array)

배열은 정수, 문자 또는 배열 등이 순차적으로 나열 된 일련의 집합입니다. PHPoC에서의 배열은 값(Value)과 키(Key)로 구성됩니다. 키 값은 숫자만 사용이 가능하고 0부터 순서대로 배정됩니다.



- 선언
  배열을 선언할 때에는 반드시 초기 값을 지정해야 합니다. 초기 값은 값을 직접 입력하거나 변수 형태로 입력하는 것도 가능합니다.

```
<?php
    $int1 = 1;
    $char1 = 's';
    $str1 = 'sollae';
    $array1 = array(1, 2, 3);                 // 값이 모두 정수인 배열
    $array2 = array('a', 'b', 'c');           // 값이 모두 문자인 배열
    $array3 = array($int1, $char1, $str1);    // 정수와 문자가 혼합된 배열
?>
```

- 사용
  배열은 대괄호('[' 와 ']')에 키 값을 명시하여 사용할 수 있습니다.

```
<?php
    $array1 = array(1, 2, 3);                 // 배열 array1의 선언 및 초기화
    $array1[0] = 5;                           // 배열 array1의 키 0번의 값을 5로 변경
    echo $array1[0];                          // 배열 array1의 키 0번의 값을 출력
?>
```

```
[출력결과]
5
```

- 문자열과 배열
  문자열은 다음과 같이 배열처럼 사용할 수 있습니다.

```
<?php
    $str = "test";                          // 문자열 변수 선언
    $str[0] = 'T';                          // 문자열 변수의 첫번째 문자를 T로 변경
    echo $str;
?>
```

```
[출력결과]
Test
```

- 다차원 배열
  PHPoC는 다차원 배열을 지원합니다.

```
<?php
    $array0 = array(0, 1, 2);                           // 1차원 배열
    $array1 = array(3, 4, 5);
    $array2 = array($array0, $array1, array(6, 7, 8));    // 2차원 배열
?>
```

자동 형 변환 (Type Juggling)

- 산술 연산자: 더하기(+), 빼기(-), 곱하기(), 나누기(/)

구분	진리 값	정수	실수	문자열
진리 값	X	X	X	X
정수	X	O	O	X
실수	X	O	O	X
문자열	X	X	X	X

- 산술 연산자: 나머지(%)

구분	진리 값	정수	실수	문자열
진리 값	X	X	X	X
정수	X	O	X	X
실수	X	X	X	X
문자열	X	X	X	X

- 비트 연산자: AND(&), OR(|), XOR(^), 왼쪽시프트(<<), 오른쪽시프트(>>)

구분	진리 값	정수	실수	문자열
진리 값	X	X	X	X
정수	X	O	X	X
실수	X	X	X	X
문자열	X	X	X	X

- 비트 연산자: 보수(~)

진리 값	정수	실수	문자열
X	O	X	X

- 비교 연산자: ~보다 작음(<), ~보다 큼(>), ~보다 작거나 같음(<=), ~보다 크거나 같음(>=)

구분	진리 값	정수	실수	문자열
진리 값	X	X	X	X
정수	X	O	O	X
실수	X	O	O	X
문자열	X	X	X	O

- 비교 연산자: 같음(==), 같지 않음(!=), 같지 않음(<>)

구분	진리 값	정수	실수	문자열
진리 값	O	X	X	X
정수	X	O	X	X
실수	X	X	O	X
문자열	X	X	X	O

- 증감 연산자: 증가(++), 감소(--)

진리 값	정수	실수	문자열
X	O	X	X

- 논리 연산자: AND(&&), OR(||)

구분	진리 값	정수	실수	문자열
진리 값	O	O	X	O
정수	O	O	X	O
실수	X	X	X	X
문자열	O	O	X	O

- 논리 연산자: NOT(!)

진리 값	정수	실수	문자열
O	O	X	O

- 부호 연산자: 양수(+), 음수(-)

진리 값	정수	실수	문자열
X	O	O	X

- 제어구조의 표현식: if문, for문, (do) while문

진리 값	정수	실수	문자열
O	O	X	O

- printf 함수 출력 포맷

구분	진리 값	정수	실수	문자열
%b, %o, %x	X	O	X	X
%d, %u	X	O	X	X
%c	X	O	X	X
%e, %f, %g	X	X	O	X
%s	X	X	X	O

변수 (Variable)

변수

변수는 변수 기호와 변수 이름으로 구성됩니다.

변수 기호	변수 이름	
	첫 글자	나머지 글자
$	알파벳 또는 _(밑줄)	알파벳, 숫자 또는 _(밑줄)

사용 예는 다음과 같습니다.

올바른 사용	$_var = 0; $var1 = 0; $var_1 = 0;

잘못된 사용	$123 = 0; //변수 이름이 숫자로 시작
	$var_#% = 0; // 변수 이름에 특수문자(#, %) 사용

변수를 선언할 때 반드시 초기 값을 지정해 주어야 합니다. 또한 단일 행에 두개 이상의 변수를 선언할 수 없습니다.

올바른 사용	$var1 = 0; $var2 = 1; $var3 = 2;

잘못된 사용	$var1; // 초기 값 없음
	$var2 = 0, $var3 = 1; // 단일 행에 변수 2개 선언

※ 변수 이름의 최대 크기는 31 바이트 입니다. 변수 이름이 31 바이트를 넘는 경우에는 나머지 부분이 무시됩니다.

미리 정의 된 변수

$GLOBALS

이 변수는 스크립트 슈퍼글로벌 변수입니다. 슈퍼글로벌 변수는 global 키워드 없이 항상 모든 범위에서 사용할 수 있습니다. 이 변수는 정수형이지만 다음과 같이 배열 형태로 변경할 수 있습니다.

```
if(!is_array($GLOBALS))
    $GLOBALS = array(0, 0, 0, 0);
$GLOBALS[0] = 1;
$GLOBALS[1] = "abc";
$GLOBALS[2] = 3.14;
$GLOBALS[3] = array("a", "b", "c", "d");
```

변수의 사용 범위

PHPoC에서의 변수는 기본적으로 변수가 선언 된 해당 영역 안에서만 사용이 가능합니다.

```
<?php
```

```
    $var1 = 0;       // $var1은 test() 함수 밖에서만 사용 가능
    function test()
    {
        $var2 = 1;     // $var2는 test() 함수 안에서만 사용 가능
    }
?>
```

- global 키워드
  global 키워드를 사용하면 변수의 사용 범위를 다른 영역으로 확장할 수 있습니다.

```
<?php
    $var1 = 0;
    function test()
    {
        global $var1;  // $var1은 test() 함수 안에서도 사용 가능
    }
?>
```

가변 변수이름

PHPoC는 가변 변수이름을 지원하지 않습니다.

상수 (Constants)

상수는 스크립트 실행 중에 변하지 않는 값(숫자 또는 문자열)을 저장하는 식별자(이름)입니다. 상수는 define 키워드를 이용해 다음과 같이 선언 후 사용합니다.

```
<?php
  define("TEST_CONST", 16);                 // 정수형 상수
  define("TEST_NAME", "constant");          // 문자열 형 상수
?>
```

※ PHPoC는 const 키워드를 사용하는 상수의 선언을 지원하지 않습니다.

※ PHPoC는 마법 상수(Magic Constants)를 지원하지 않습니다.

연산자 (Operators)

연산자는 숫자나 문자열 등의 값 또는 변수들을 사용하여 연산을 하는 기호를 말합니다. PHPoC는 대입, 산술, 증감, 비교, 논리, 문자열, 비트, 삼항 연산자를 제공합니다.

- 연산자 우선순위
- 산술 연산자
- 대입 연산자
- 비트 연산자
- 비교 연산자
- 증감 연산자
- 논리 연산자
- 문자열 연산자
- 삼항 연산자

※ PHPoC는 에러 제어 연산자(Error Control Operators), 실행 연산자 (Execution Operators) 및 배열 연산자 (Array Operators) 등은 지원하지 않습니다.

연산자 (Operators)

연산자 우선순위

하나의 표현에 연산자가 2개 이상인 경우, 연산의 순서를 결정하는 것이 연산자 우선순위 입니다.
PHPoC에서 연산자 우선순위는 다음과 같습니다.

우선순위	연산자 기호	연산자 구분
높음	[(	괄호
	++ -- ~ (int) (string) (bool)	증감/캐스트
	!	논리
	* / %	산술
	+ - .	산술
	<>	비트
	< <= > >=	비교
	== != === !== <>	비교
	&	비트
	^	비트
	|	비트
	&&	논리
	||	논리
	? :	비교(삼항)
낮음	= += -= *= /= .= %= &= |= ^= <>=	대입

우선순위가 같은 연산자가 중복되어 사용되는 경우에는 왼쪽에 있는 항부터 먼저 연산이 이루어집니다.
단, 대입, 증감, 캐스트 및 논리 연산자 '!'는 예외적으로 오른쪽부터 연산이 시작됩니다.

- 연산자 우선순위 적용 예

```
<?php
    $var0 = 3 * 3 % 5;          // (3 * 3) % 5 = 4 (왼쪽부터)

    $var1 = 1;
    $var2 = 2;
    $var1 = $var2 += 3;         // $var1 = ($var2 += 3), $var1, $var2 = 5 (오른쪽부터)
?>
```

연산자 (Operators)

산술 연산자

연산자	기호	사용 예	비고
더하기	+	$var1 + $var2	-
빼기	-	$var1 - $var2	-
곱하기	*	$var1 * $var2	-
나누기	/	$var1 / $var2	정수 연산에서는: $var1을 $var2로 나눈 몫 (정수 부분)
나머지	%	$var1 % $var2	$var1을 $var2로 나눈 나머지

연산자 (Operators)

대입 연산자

대입 연산자는 대입연산자 우측의 값 또는 수식의 결과를 왼쪽의 변수에 대입합니다.

연산자	기호	사용 예	비고
기본	=	$var = 1	$var에 1을 대입
더하기	+=	$var += 1	$var에 $var + 1의 결과를 대입
빼기	-=	$var -= 1	$var에 $var - 1의 결과를 대입
곱하기	*=	$var *= 2	$var에 $var * 2의 결과를 대입
나누기	/=	$var /= 2	$var에 $var / 2의 결과를 대입
나머지	%=	$var %= 2	$var에 $var % 2의 결과를 대입
문자열 연결	.=	$var .= "string"	문자열 $var의 마지막에 "string" 추가
AND(비트)	&=	$var &= 0x02	$var에 $var & 0x02의 결과를 대입
OR(비트)	|=	$var |= 0x02	$var에 $var | 0x02의 결과를 대입
XOR(비트)	^=	$var ^= 0x02	$var에 $var ^ 0x02의 결과를 대입
비트시프트(좌)	<<=	$var <<= 4	$var에 $var << 4의 결과를 대입
비트시프트(우)	>>=	$var >>= 4	$var에 $var >> 4의 결과를 대입

연산자 (Operators)

비트 연산자

연산자	기호	사용 예	비고
AND	&	$b1 & $b2	$b1과 $b2를 bit AND
OR	|	$b1 | $b2	$b1과 $b2를 bit OR
보수	~	~$b1	$b1을 반전(0은 1로 , 1은 0으로)
XOR	^	$b1 ^ $b2	$b1과 $b2를 bit XOR
왼쪽시프트	<<	$b1 << 8	$b1을 왼쪽으로 8자리만큼 시프트
오른쪽시프트	>>	$b1 >> 8	$b1을 오른쪽으로 8자리만큼 시프트

- 비트연산의 예

```
<?php
    $b1 = 0x11;              // 0001 0001
    echo "$b1₩r₩n";
    $b2 = 0x23;              // 0010 0011
    echo "$b2₩r₩n";
    $b3 = $b1 & $b2;           // 0000 0001, bit AND
    echo "$b3₩r₩n";
    $b3 = $b1 | $b2;           // 0011 0011, bit OR
    echo "$b3₩r₩n";
    $b3 = ~$b1;               // 1110 1110, NOT
    echo "$b3₩r₩n";
    $b3 = $b1 << 1;            // 0010 0010, 왼쪽으로 1비트 이동 (곱하기 2)
    echo "$b3₩r₩n";
    $b3 = $b1 >> 1;            // 0000 1000, 오른쪽으로 1비트 이동 (나누기 2)
    echo "$b3₩r₩n";
?>
```

```
[출력결과]
17
35
1
51
-18
34
8
```

- 왼쪽시프트
  왼쪽시프트 연산에 의해 생성되는 비트들은 항상 값이 0으로 채워지게 됩니다.

```
<?php
   $b1 = 0xFFFFFFFFFFFFFFFF;  // -1
   $b2 = $b1 << 1;            // 0xFFFFFFFFFFFFFFFE (0이 채워짐)
   echo "$b2";
?>
```

```
[출력결과]
-2
```

- 오른쪽시프트
  오른쪽시프트 연산에 의해 생성되는 비트들은 부호비트(Sign bit)와 같은 값으로 채워지게 됩니다.

```
<?php
   $b1 = 0xFFFFFFFFFFFFFFFF;  // -1
   $b2 = $b1 >> 1;            // 0xFFFFFFFFFFFFFFFF (1이 채워짐)
   echo "$b2";
?>
```

```
[출력결과]
-1
```

연산자 (Operators)

비교 연산자

비교 연산자의 결과는 항상 진리 값 형태입니다.

연산자	기호	사용 예	비고
같음(값)	==	$var1 == $var2	$var1과 $var2의 값이 같음
같음(값, 자료 형)	===	$var1 === $var2	$var1과 $var2가 같음
같지 않음(값)	!=	$var1 != $var2	$var1과 $var2의 값이 다름
같지 않음(값, 자료 형)	!==	$var1 !== $var2	$var1과 $var2가 다름
같지 않음	<>	$var1 <> $var2	$var1과 $var2가 같지 않음
~보다 작음	<	$var1 < $var2	$var1이 $var2보다 작음
~보다 큼	>	$var1 > $var2	$var1이 $var2보다 큼
~보다 작거나 같음	<=	$var1 <= $var2	$var1이 $var2보다 작거나 같음
~보다 크거나 같음	>=	$var1 >= $var2	$var1이 $var2보다 크거나 같음

- 비교 연산자의 사용 예

```
<?php
    $var1 = 1;
    $var2 = 2;
    $var3 = $var1 == $var2;    // $var3 = False (0 - 거짓)
    $var4 = $var1 != $var2;    // $var4 = True (1 - 참)
    $var5 = $var1 <> $var2;    // $var5 = True (1 - 참)
    $var6 = $var1 < $var2;     // $var6 = True (1 - 참)
    $var7 = $var1 > $var2;     // $var7 = False (0 - 거짓)
    $var8 = $var1 <= $var2;    // $var8 = True (1 - 참)
    $var9 = $var1 >= $var2;    // $var9 = False (0 - 거짓)
?>
```

연산자 (Operators)

증감 연산자

연산자	기호	사용 예	비고
증가 연산자	++	$var++	연산을 수행하고 $var을 1 증가
		++$var	$var을 1 증가하고 연산을 수행
감소 연산자	--	$var--	연산을 수행하고 $var을 1 감소
		--$var	$var을 1 감소하고 연산을 수행

- 증감 연산자 사용 예

```
<?php
    $var = 3;

    echo $var++;       // echo 수행 후 $var 값 1 증가
    echo $var;

    echo ++$var;       // $var 값 1 증가 후 echo 수행
    echo $var;

    echo $var--;       // echo 수행 후 $var 값 1 감소
    echo $var;

    echo --$var;       // $var 값 1 감소 후 echo 수행
    echo $var;
?>
```

```
[출력 결과]
34555433
```

연산자 (Operators)

논리 연산자

연산자	기호	사용 예	비고
AND	&&	(expr1) && (expr2)	(expr1)과 (expr2)가 모두 참일 때만 참
OR	||	(expr1) || (expr2)	(expr1)과 (expr2) 둘 중 하나만 참이면 참
NOT	!	!(expr1)	(expr1)이 참이면 거짓, 거짓이면 참

- 논리 연산자 사용 예

```
<?php
    $var1 = true;
    $var2 = false;

    $var3 = $var1 && $var2;
    $var4 = $var1 || $var2;
    $var5 = !$var1;

    echo (int)$var3, "₩r₩n";     // 거짓
    echo (int)$var4, "₩r₩n";     // 참
    echo (int)$var5, "₩r₩n";     // 거짓
?>
```

```
[출력결과]

1
```

연산자 (Operators)

문자열 연산자

연산자	기호	사용 예	비교
문자열 연결	.	$str1 . $str2	$str1과 $str2를 연결
	.=	$str1 .= $str2	$str1에 $str2를 연결한 값을 $str1에 대입

- 문자열 연산자 사용 예

```
<?php
    $str1 = "Hel";
    $str2 = "lo";

    $str3 = $str1 . $str2;  // $str3 = "Hello"
    $str3 .= " PHPoC!";     // $str3 = "Hello PHPoC!"

    echo $str3;
?>
```

```
[출력결과]
Hello PHPoC!
```

연산자 (Operators)

삼항 연산자

연산자	기호	사용 예	비고
삼항 연산자	? :	(expr) ? $a : $b	expr이 참이면 $a, 거짓이면 $b

- 삼항 연산자의 사용 예

```
<?php
    $var1 = $var2 = 1;
    $var3 = ($var1 == $var2) ? true : false;
    $var4 = ($var1 != $var2) ? true : false;
    echo (int)$var3, "₩r₩n";
    echo (int)$var4;
?>
```

```
[출력 결과]
1
```

※ PHPoC에서 하나의 명령 줄에 삼항 연산자의 중복 사용은 의도하지 않은 결과가 나올 수 있으므로 권장하지 않습니다.

제어 구조 (Control Structures)

PHPoC 스크립트는 일련의 명령문으로 이루어집니다. 명령문에는 할당, 함수 호출, 반복문, 조건문 등이 올 수 있으며 아무 내용이 없는 경우도 있습니다. 각 명령문은 세미 콜론(;)으로 구분됩니다. 이러한 명령문들은 중괄호({, })로 둘러 싸 그룹화 할 수 있습니다.

PHPoC의 모든 명령문 또는 명령문 그룹들은 기본적으로 위에서부터 차례로 실행되지만 제어 구조를 활용하면 특정 위치의 명령 행을 조건에 따라서 건너뛰거나 반복적으로 실행하는 등의 제어가 가능합니다.

PHPoC가 제공하는 대부분의 제어구조는 C언어를 비롯한 다른 프로그래밍 언어와 매우 유사합니다.

- if문
- else문
- elseif / else if문
- while문
- do-while문
- for문
- break
- continue
- switch문
- return
- include
- include_once

※ PHPoC는 PHP에서 제공하는 제어 구조 중 foreach, declare, require, require_once 및 goto는 지원하지 않습니다.

제어 구조 (Control Structures)

if 문

if 문은 임의의 표현 식의 결과에 따라서 정해진 명령 행을 실행할지 또는 건너뛸지를 결정하는 가장 기본적인 제어 구조 중 하나입니다.

- if 문의 구조

문법 구조	설명
if (표현식) 명령문;	표현식의 결과가 참이면 명령문을 실행하고, 표현식의 결과가 거짓이면 명령문을 실행하지 않음

- 단일 행 if 문의 사용 예

```
<?php
    $var1 = $var2 = 1;
    if($var1 == $var2)                  // 표현식의 결과가 참
      echo "var1 and var2 are equal";  // if 문 내부의 명령문을 실행
?>
```

```
[출력 결과]
var1 and var2 are equal
```

- 다중 행 if 문의 사용 예

```
<?php
    $var1 = 1;
    $var2 = 2;
    if($var1 < $var2)
    {                                   // 중괄호로 명령문을 그룹화
        echo "var1 is smaller than var2";
        echo "₩r₩nbye!";
    }                                   // 중괄호로 명령문을 그룹화
?>
```

```
[출력 결과]
var1 is smaller than var2
bye!
```

- 중첩 if 문의 사용 예

```
<?php
    $var1 = $var2 = 1;
    $var3 = 2;
    if($var1 == $var2)                  // 표현식의 결과가 참
    {
      if($var1 < $var3)                 // 표현식의 결과가 참
        echo "var1 and var2 are equal";  // if 문 내부의 명령문을 실행
    }
?>
```

제어 구조 (Control Structures)

else 문

else 문은 if 문의 표현식의 결과가 거짓일 때 실행할 명령문을 정의합니다. 따라서 if-else 문을 사용하면 표현식의 결과가 참일 때와 거짓일 때의 명령문을 모두 정의할 수 있습니다. else 문은 표현식이 따로 없으며, if 문 없이 단독으로 사용할 수 없습니다.

- if-else 문의 구조

문법구조	설명

if(표현식)
명령문1;
else
명령문2;
	1) 표현식의 결과가 참이면 명령문1을 실행
	2) 표현식의 결과가 참이 아니면 명령문2를 실행

- if-else문의 사용 예

```
<?php
    $var1 = $var2 = 2;
    if($var1 == $var2)                    // 표현식의 결과가 거짓
      echo "var1 and var2 are equal";
    else
      echo "var1 and var2 are not equal";  // else 문 내부의 명령문을 실행
?>
```

```
[출력 결과]
var1 and var2 are equal
```

- 중첩 if-else 문의 사용 예

```
<?php
    $var1 = $var2 = 1;
    $var3 = 2;
    if($var1 > $var2)                  // 표현식의 결과가 거짓
      echo "var1 and var2 are equal";
    else
    {
      if($var1 > $var3)                // 표현식의 결과가 거짓
        echo "good";
      else
        echo "bad";                    // else 문 내부의 명령문을 실행
    }
?>
```

```
[출력 결과]
bad
```

제어 구조 (Control Structures)

elseif / else if 문

elseif 문은 if와 else를 합쳐놓은 것입니다. 이 제어구조는 else와 마찬가지로 if 문을 연장하여 if 문의 표현식이 거짓일 때 수행할 명령문을 정의할 때 사용합니다. 그러나 else 와는 달리 표현식을 가지며 이 표현식이 참일 때에만 명령문을 수행합니다.

elseif 문은 if 문 없이 단독으로 사용할 수 없지만, if 문 하나에 여러 개의 elseif 문을 사용할 수 있습니다.

- elseif 문의 구조

문법구조	설명

if(표현식1)
  명령문1;
elseif(표현식2)
  명령문2;
elseif(표현식3)
  명령문3;
else
  명령문4;
	
	1)표현식1의 결과가 참이면 명령문1을 실행
	2)표현식2의 결과가 참이면 명령문2를 실행
	3)표현식3의 결과가 참이면 명령문3을 실행
	4) 표현식 1~3의 결과가 모두 거짓이면 명령문4를 실행

- elseif 문의 사용 예

```
<?php
    $var1 = 1;
    $var2 = 2;
    $var3 = 3;
    if($var1 == 0)                // 표현식의 결과가 거짓
      echo "var1 = 0";
    elseif($var2 == 0)            // 표현식의 결과가 거짓
      echo "var2 = 0";
    elseif($var3 == 0)            // 표현식의 결과가 거짓
      echo "var3 = 0";
    elseif($var3 == 3)            // 표현식의 결과가 참
      echo "var3 = 3";            // elseif 문의 명령문이 실행 됨
    else
      echo "No Result";
?>
```

```
[출력 결과]
var3 = 3
```

제어 구조 (Control Structures)

while 문

while 문은 가장 간단한 반복문입니다. 반복문은 일정 조건을 만족하는 동안 특정한 명령문을 반복적으로 실행하고자 할 때 사용합니다.

- while 문의 구조

문법 구조	설명
while(표현식) 명령문;	표현식의 결과가 참인 동안 명령문을 반복 실행

- while 문의 사용 예

```
<?php
    $var = 0;
    while($var < 3)     // $var이 3보다 작으면 아래의 명령문 반복 실행
    {
      echo "$var₩r₩n";  // 최종적으로 3번의 echo가 실행 됨
      $var++;            // $var을 1 증가
      sleep(1);         // 프로그램 실행을 1초 동안 대기
    }
?>
```

```
[출력 결과]

1
2
```

- 무한루프
  반복문에 사용된 표현식의 결과가 언제나 참인 경우 반복문이 무한히 실행될 수 있습니다. 반복문을 빠져나오지 못하는 이러한 상태를 무한루프라고 합니다. 개발자가 의도하지 않은 무한루프에 빠지게 되면 프로그램이 다음 단계로 진행되지 않으므로 프로그래밍 할 때 특히 유의해야합니다.

```
<?php
    $var = 0;
    while(1)              // 표현식의 결과가 항상 참 (1 = true)
    {
      echo "$var₩r₩n";
      $var++;              // $var을 1 증가, $var = 1, 2, 3, ...
      sleep(1);           // 프로그램 실행을 1초 동안 대기
    }
?>
```

제어 구조 (Control Structures)

do-while 문

do-while 문은 표현식의 검사를 반복문의 시작이 아닌 끝에 수행하는 것을 제외하고는 while 문과 거의 유사합니다.

- do-while 문의 구조

문법 구조	설명

do
{
  명령문;
}
while(표현식);
	1) 명령문을 먼저 실행하고 표현식을 검사
	2) 표현식의 결과가 참인 동안 명령문을 반복 실행

- do-while 문의 사용 예

```
<?php
    $var = 0;
    do
    {
      echo "$var₩r₩n";  // 아래 표현식과 상관 없이 무조건 한 번은 실행
      $var++;
      sleep(1);
    }while($var < 3);
?>
```

```
[출력 결과]

1
2
```

제어 구조 (Control Structures)

for 문

for 문은 특정 횟수만큼 명령문을 반복하기 위해 사용하는 제어 구조입니다.

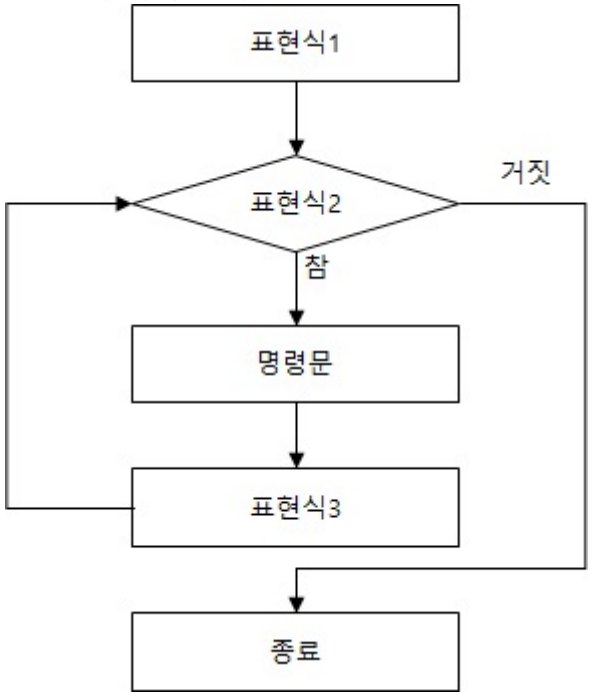
- for 문의 구조

문법 구조	설명

for(표현식1;표현식2;표현식3)
{
  명령문;
}
	1) 표현식1을 먼저 수행하고 표현식2를 검사
	2) 표현식2의 결과가 참이면 명령문을 실행
	3) 표현식3 실행

일반적으로 표현식1에는 for 문에서 사용될 변수의 초기 값을 지정해주고, 표현식2에는 명령문을 실행할지를 결정하는 조건문을 설정합니다. 표현식3에는 증감연사자를 이용해 반복 횟수를 카운팅하도록 설정합니다.

- for 문의 처리 순서도



- for 문 사용 예

```
<?php
   for($i = 0; $i < 5; $i++)  // $i가 0부터 1씩 증가하면서 5보다 작을 동안
   {
     echo $i;                 // $i가 0, 1, 2, 3, 4인 경우에 명령문 실행
   }
?>
```

```
[출력 결과]
01234
```

for 문의 각 표현식들은 아래와 같이 생략이 가능합니다.

- 표현식의 생략 예1

```
<?php
    for($i = 1; ; $i++)  // 표현식2 생략
    {
      if($i > 10)
        break;           // 반복문을 빠져 나옴
      echo $i;
    }
?>
```

```
[출력 결과]
12345678910
```

- 표현식의 생략 예2

```
<?php
    $i = 0;
    for( ; ; )     // 표현식 모두 생략 - 무한루프
    {
      if($i > 10)
        break;     // 반복문을 빠져 나옴
      echo $i;
      $i++;
    }
?>
```

```
[출력 결과]
012345678910
```

- for 문과 배열의 궁합
  for 문은 배열의 각 요소를 순차적으로 처리할 때 사용하면 매우 편리합니다.

```
<?php
    $arr = array(1, 2, 3);      // arr[0] = 1, arr[1] = 2, arr[2] = 3
    for($i = 0; $i < 3; $i++)  // $i가 0, 1, 2일 때 명령문 실행
    {
      echo $arr[$i];            // 배열의 각 요소를 출력
    }
?>
```

```
[출력 결과]
123
```

제어 구조 (Control Structures)

break

break는 반복문(while, do-while 및 for) 또는 switch 문에서 명령문의 실행을 멈추고 빠져나오기 위한 제어 구조입니다.

- break의 구조

문법 구조	설명

for(; ;)
{
  if(표현식)
  {
  명령문;
  break;
  }
}
	for 문을 무한 반복하다가 if 문의 표현식이 참이면
	명령문을 실행하고 break에 의해 for 문을 빠져 나옴

- break의 사용 예

```
<?php
  for($i = 0; ; $i++)  // 무한 루프
  {
    if($i > 10)
      break;           // 반복문을 빠져 나옴
    echo $i;
  }
?>
```

```
[출력 결과]
012345678910
```

- break의 옵션
  break는 뒤에 숫자 옵션이 올 수 있습니다. 이 숫자는 중첩된 제어구조에서 빠져나올 제어구조 레벨의 수를 의미합니다.

```
<?php
  $j = 1;
  for($i = 0; ; $i++)  // 무한 루프 (레벨 1)
  {
    while($j != 0)     // 무한 루프 (레벨 2)
    {
      if($j > 10)
        break 2;       // while 문(레벨 2)은 물론 for 문(레벨 1)까지 빠져 나옴
      echo $j;
      $j++;
    }
  }
?>
```

```
[출력 결과]
12345678910
```

제어 구조 (Control Structures)

continue

continue는 반복문에서 명령문의 실행을 멈추고 반복문의 맨 처음으로 돌아가 조건절을 다시 평가하게 하는 제어 구조입니다.

- continue의 구조

문법 구조	설명

for(; ;)
{
  if(표현식)
  {
  명령문1;
  continue;
  }
  명령문2;
}
	for 문을 무한 반복하다가 if 문의 표현식이 참이면 명령문1을 실행하고 continue에 의해 for 문의 맨 처음으로 이동

- continue의 사용 예

```
<?php
  for($i = 1; ; $i++)  // 무한 루프
  {
    if($i % 5)
      continue;         // for 문의 맨 처음으로 이동
    echo "$i₩r₩n";      // echo는 $i가 5의 배수일 때 마다 실행
    sleep(1);
  }
?>
```

```
[출력 결과]
5
10
15
... (반복)
```

- continue의 옵션
  continue는 뒤에 숫자 옵션이 올 수 있습니다. 이 숫자는 중첩된 제어구조에서 건너뛸 제어구조 레벨의 수를 의미합니다.

```
<?php
  $j = 0;
  for($i = 0; ; $i++)                          // 무한 루프 (레벨 1)
  {
    sleep(1);
    if($i)
      echo "This is for statement ₩$i = $i₩r₩n";  // continue 2에 의해 재실행됨
    while(1)                                   // 무한 루프 (레벨 2)
    {
      $j++;
      if(($j % 5) == 0)
        continue 2;                            // for문의 처음으로 이동
      echo "$j, ";
      sleep(1);
    }
  }
?>
```

```
[출력 결과]
1, 2, 3, 4, This is for statement $i = 1
6, 7, 8, 9, This is for statement $i = 2
11, 12, 13, 14, This is for statement $i = 3
... (반복)
```

제어 구조 (Control Structures)

switch 문

switch 문은 switch-case 문이라고도 하며, 구조상 if 문과 매우 유사합니다. switch 문은 하나의 표현 식을 여러 개의 값과 비교하여 각각의 값에 대한 다른 명령문을 수행하고 싶을 때 주로 사용합니다. default를 사용하면 switch 문의 표현 식이 모든 case의 값과 다를 때 수행할 기본 명령문을 지정할 수 있습니다.

- switch 문의 구조

문법 구조	설명

switch(표현식)
{
case 값1:
명령문1;
break;
case 값2:
명령문2;
break;
default:
명령문3;
break;
}
	1) 표현식의 결과가 값1과 같은지 비교
	2) 1)의 값이 같으면 명령문1을 수행하고 빠져 나옴
	3) 1)의 값이 다르면 표현식의 결과를 값2와 비교
	4) 3)의 값이 같으면 명령문2를 수행하고 빠져 나옴
	5) 3)의 값이 다르면 명령문3을 수행함

- switch 문의 사용 예

```
<?php
  $var = 1;
  switch($var)
  {
    case 1:
      echo "var is 1";
      break;
    case 2:
      echo "var is 2";
      break;
    default:
      echo "Error";
      break;
  }
?>
```

```
[출력 결과]
var is 1
```

- default의 사용 예
  switch 문을 사용할 때 default는 생략할 수 있습니다.

```
<?php
  $var = 1;
  switch($var)
  {
    case 1:
      echo "var is 1";
      break;
    case 2:
      echo "var is 2";
      break;
  }
?>
```

```
[출력 결과]
var is 1
```

break는 각각의 case문에 반드시 사용해야 하는것은 아닙니다. 만약 현재 수행중인 case문에 break가 없으면 break가 나올때까지 다음 case문들이 순차적으로 수행됩니다.

- break가 생략된 case문

```
<?php
  $var = 1;
  switch($var)
  {
    case 1:
      echo "3";
    case 2:
      echo "3";
    case 3:
      echo "3";
      break;
    case 4:
      echo "4";
  }
?>
```

```
[result]
333
```

※ case 표현식 뒤에 콜론(:) 대신 세미콜론(;)을 사용할 수 없습니다.

제어 구조 (Control Structures)

return

return을 함수 안에서 사용하면 PHPoC는 즉시 함수의 실행을 멈추고 인자 값을 함수를 호출한 곳으로 반환합니다. 또한 함수가 아닌 include된 스크립트 파일의 글로벌 영역에서 사용하면 스크립트의 실행이 종료되고, 실행 순서가 include한 파일로 되돌아갑니다. 이 경우에도 반환 값이 있다면 include한 파일에 해당 값을 넘겨줍니다.

만약 init.php에서 return을 사용하면 스크립트의 실행이 종료됩니다.

문법 구조	설명
return 인자 값;	함수 또는 스크립트의 실행을 종료하고 인자 값을 반환 함 인자 값은 생략 가능

- 함수 내부에서의 반환 예

```
<?php
function func()           // 사용자 정의함수 func() 선언
{
  $var1 = 1;
  return $var1;           // $var1의 값(1)을 반환
}

$var2 = 2;
$var3 = func();           // func() 호출로 $var3에는 $var1의 값이 대입 됨
$result = $var2 + $var3;  // 2 + 1 = 3
echo $result;
?>
```

```
[출력 결과]
3
```

- 파일에서 파일로의 반환 예

test.php

```
<?php
$var2 = 2;
$var3 = 3;
return ($var2 + $var3); // 값 5 반환
?>
```

init.php

```
<?php
$var1 = include_once "test.php";
echo $var1;
?>
```

```
[출력 결과]
5
```

- init.php에서의 사용 예

```
<?php
$var1 = 1;
echo ++$var1;  // 실행 됨
echo ++$var1;  // 실행 됨
return;        // 스크립트 종료
echo ++$var1;  // 실행 안 됨
?>
```

```
[출력 결과]
23
```

제어 구조 (Control Structures)

include

include는 지정된 스크립트 파일의 내용을 현재 스크립트에 포함시키는 데 사용됩니다.

문법 구조	설명
include 파일명;	파일명에 해당하는 파일을 현재 스크립트에 포함함 파일명은 문자열 변수 형태로도 사용 가능 파일명은 대/소문자를 구분 함

- include 사용 예

test.php

```
<?php
$var1 = 1;
$var2 = 2;
?>
```

init.php

```
<?php
$var1 = $var2 = 0;
echo $var1 + $var2;
include "test.php";  // test.php를 포함
echo $var1 + $var2;
?>
```

```
[출력 결과]
03
```

- function 내부에서의 사용 예
  function 내부에서 include한 파일에 사용 된 변수를 글로벌 영역에서 사용하기 위해서는 반드시 해당 변수를 글로벌 변수로 선언해야 합니다.

test.php

```
<?php
$var1 = 1;
$var2 = 2;
?>
```

init.php

```
<?php
$var1 = $var2 = 0;
function func()
{
  global $var1;         // $var1만 글로벌 변수 선언
  include "test.php";  // test.php를 포함
  echo $var1 + $var2;
}
func();
echo $var1 + $var2;
?>
```

```
[출력 결과]
31
```

- include의 return 사용 예
  include한 파일의 return 인자가 없는 경우에는 파일 포함 성공시 1을 반환하고, 실패시에는 PHP 에러가 발생합니다.

test1.php

```
<?php
// $var를 반환
$var = 3;
return $var;
?>
```

test2.php

```
<?php
// 반환 없음
$var = 3;
return;
?>
```

init.php

```
<?php
$var1 = include "test1.php";
echo $var1;
$var2 = include "test2.php";
echo $var2;
?>
```

```
[출력 결과]
31
```

제어 구조 (Control Structures)

include_once

include_once는 include와 마찬가지로 지정된 스크립트 파일의 내용을 현재 스크립트에 포함시키는 데 사용됩니다. 그러나 include와는 달리 이미 포함된 파일이 있는 경우 해당 파일을 다시 포함시키지 않습니다.

문법 구조	설명
include_once 파일명;	파일명에 해당하는 파일을 현재 스크립트에 포함함 파일명은 문자열 변수 형태로도 사용 가능 파일명은 대/소문자를 구분 함

- include_once 사용 예

test.php

```
<?php
echo "Hello₩r₩n";
?>
```

init.php

```
<?php
include "test.php";       // test.php를 포함시킴
include "test.php";       // test.php를 다시 포함시킴
include_once "test.php";  // test.php를 포함시키지 않음
?>
```

```
[출력 결과]
Hello
Hello
```

함수 (Functions)

- 사용자 정의 함수
- 함수의 인자
- 함수의 반환 값
- 내부 함수

※ PHPoC는 가변 함수이름을 지원하지 않습니다.

※ PHPoC는 이름 없는 함수를 지원하지 않습니다.

함수 (Functions)

사용자 정의 함수

사용자 정의 함수는 자주 사용되는 코드를 함수로 정의하여 필요할 때마다 함수 이름만으로 호출함으로써 소스코드의 크기를 줄이고, 분석을 쉽게 하는데 사용됩니다. 함수는 함수 이름, 인자 값, 명령문 및 반환 값 등으로 구성되며, 함수 이름 생성 규칙은 변수이름 생성 규칙과 동일합니다.

사용자 지정 함수 이름	
첫 글자	나머지 글자
알파벳 또는 _(밑줄)	알파벳, 숫자 또는 _(밑줄)

- 함수 선언의 기본 구조

문법 구조	설명

function 함수이름(인자 값)
{
  명령문;
  return 반환 값;
}
	지정된 함수 이름의 사용자 정의 함수를 생성
	인자 값은 여러 개 사용 또는 생략 가능
	return 또는 반환 값 또한 생략 가능

- 함수 호출의 기본 구조

문법 구조	설명
함수이름(인자 값1, 인자 값2, ...);	인자 값은 변수 형태로도 사용 가능 함수이름은 대/소문자를 구분 함

- 함수의 선언 및 사용 예
  사용자 정의 함수는 반드시 선언 된 이후에 호출할 수 있습니다.

```
<?php

function func()         // func() 함수 선언
{
  echo "Hello PHPoC";
}
func();                 // func() 함수 호출

?>
```

- 반환 값 사용 예

```
<?php

function func()         // func() 함수 선언
{
  return 5;
}
$var = func();          // func() 함수 호출
echo $var;

?>
```

```
[출력 결과]
5
```

- 인자 값 사용 예
  함수의 처리를 위한 값을 함수 외부로부터 전달하고자할 때 사용하는 것이 인자입니다. 함수의 인자는 쉼표(,)로 구분된 값, 변수 또는 표현식을 나열하여 사용합니다.

```
<?php

function func($arg)    // func() 함수 선언 및 $arg를 인자로 받음
{
  return $arg + 1;     // 받은 인자 $arg에 1을 더한 값을 반환
}
$var = func(2);        // func() 함수에 인자로 2를 넘기고 3을 반환 받음
echo $var;
$var = func($var);     // 인자 $var(= 3)의 값을 넘김
echo $var;
$var = func($var+1);   // 인자 $var+1(= 5)의 값을 넘김
echo $var;

?>
```

```
[출력 결과]
346
```

- 함수의 재귀적 호출 예
  함수의 선언문 안에서 함수를 다시 호출할 수 있습니다. 이 때 호출할 수 있는 함수에는 자기 자신도 포함됩니다.

```
<?php

function func($arg)   // func() 함수 선언 및 $arg를 인자로 받음
{
  if($arg < 6)
  {
    echo "$arg₩r₩n";  // $arg 출력
    $arg++;           // $arg에 1을 더한 값을 $arg에 대입
    func($arg);       // func() 함수를 호출하여 인자로 $arg를 전달
  }
}
func(1);              // func() 함수를 호출하여 인자로 1을 전달

?>
```

```
[출력 결과]
1
2
3
4
5
```

함수 (Functions)

함수의 인자

PHPoC는 인자 전달 방식으로 값에 의한 인자 전달과 참조에 의한 인자 전달 그리고 기본 인자 값을 지원합니다.

- 값에 의한 인자 전달
  인자 전달의 기본적인 형태로 함수 안에서 인자의 값을 변경해도 함수 밖에서는 적용되지 않습니다.

```
<?php

function func($arg1, $arg2)  // 값에 의한 전달
{
  $temp = $arg1;
  $arg1 = $arg2;
  $arg2 = $temp;
  return $arg1 + 1;
}
$var1 = 1;
$var2 = 2;
func($var1, $var2);          // 함수 호출
echo "$var1, $var2";         // 출력 결과 $var1과 $var2 실제 값은 바뀌지 않음

?>
```

```
[출력 결과]
1, 2
```

- 참조에 의한 인자 전달
  전달하는 변수 자체(실제로는 변수가 존재하는 메모리 주소)를 함수 내부로 전달하는 방식입니다. 변수 자체를 전달하기 때문에 함수 내부에서 변수의 값이 변경되면 함수 밖에 있는 실제 변수에 적용됩니다. 참조에 의한 전달을 위해서는 함수 선언시 인자 값 이름 앞에 엠퍼샌드(&)를 붙입니다.

```
<?php

function func(&$arg1, &$arg2)  // 참조에 의한 전달
{
  $temp = $arg1;
  $arg1 = $arg2;
  $arg2 = $temp;
  return $arg1 + 1;
}
$var1 = 1;
$var2 = 2;
func($var1, $var2);             // 함수 호출
echo "$var1, $var2";            // 출력 결과 $var1과 $var2 실제 값이 바뀜

?>
```

```
[출력 결과]
2, 1
```

- 기본 인자 값
  함수를 정의할 때 인자의 기본 값을 정의할 수 있습니다.

```
<?php

function print_str($str = "Hello PHPoC!₩r₩n")  // 기본 인자 값 설정
{
  echo $str;
}
print_str();                                  // 인자 값 없이 함수 호출

?>
```

```
[출력 결과]
Hello PHPoC!
```

함수 (Functions)

함수의 반환 값

함수 내부에서 return을 이용하여 값을 반환할 수 있습니다. 기본적으로 함수의 반환 값은 하나입니다. 여러개의 값을 반환하려면 배열 형태로 반환해야합니다.

- 배열 형태의 반환 예

```
<?php

function func()
{
  $var1 = 1;
  $var2 = 2;
  $var3 = 3;
  $arr = array($var1, $var2, $var3);
  return $arr;
}

$arr = func();
printf("%d, %d, %d₩r₩n", $arr[0], $arr[1], $arr[2]);

?>
```

```
[출력 결과]
1, 2, 3
```

※ PHPoC는 함수의 반환 값이 없을 때 NULL이 아닌 0을 반환합니다.

※ PHPoC는 참조에 의한 반환을 지원하지 않습니다.

함수 (Functions)

내부 함수

PHPoC는 다양한 내부 함수를 제공합니다. 내부함수 사용에 관한 내용은 PHPoC Internal Functions를 참조하시기 바랍니다.

클래스와 객체 (Classes and Objects)

※ PHPoC는 클래스 및 객체를 지원하지 않습니다.

네임스페이스 (Namespaces)

개요

네임스페이스는 개체를 구분할 수 있는 범위를 나타냅니다. 하나의 네임스페이스 안에서는 단 하나의 이름만이 존재할 수 있습니다.

네임스페이스의 공유

PHPoC는 키워드, 함수 그리고 상수가 하나의 네임스페이스를 공유합니다. 따라서, 사용자는 함수나 상수를 선언할 때 동일한 이름이 없도록 유의해야 합니다.

부록

- 미리 정의된 상수 (Predefined Constants)
- 키워드 (Keyword)
- 메모리 관련
- 오류 메시지 (Error Messages)

부록

미리 정의된 상수 (Predefined Constants)

PHPoC는 다음의 미리 정의된 상수를 제공합니다.

미리 정의된 상수	설명
COUNT_NORMAL	일반적 카운팅 (1차원 배열 요소 카운팅)
COUNT_RECURSIVE	반복적 카운팅 (다차원 배열 요소 카운팅)
EPIPE	파이프 종료(Broken Pipe), TCP 데이터 송신 중 접속이 끊어지면 반환됨
EBUSY	디바이스 사용중 (Device or Resource Busy)
ENOENT	파일 엔트리에 없음
FALSE	거짓
M_PI	원주율 (≒ 3.1415926535898)
M_E	자연대수 (≒ 2.718281828459)
MAX_STRING_LEN	최대 문자열 변수 길이
O_NODIE	스크립트 종료 회피: 파일 열기
PHP_VERSION_ID	PHPoC 버전 정보
SEEK_SET	파일 포인터 위치: 파일의 시작
SEEK_CUR	파일 포인터 위치: 현재 위치
SEEK_END	파일 포인터 위치: 파일의 끝
SSL_CONNECTED	SSL 접속상태: 접속 완료
SSL_CLOSED	SSL 접속상태: 접속 끊김
SSL_LISTEN	SSL 접속상태: 접속 대기
TRUE	참
TCP_CLOSED	TCP 접속상태: 접속 끊김
TCP_LISTEN	TCP 접속상태: 접속 대기
TCP_CONNECTED	TCP 접속상태: 접속 완료

부록

키워드 (Keyword)

다음은 PHPoC에서 미리 정의된 키워드입니다. 키워드, 함수 그리고 상수는 하나의 네임스페이스를 공유하므로 함수 및 상수를 선언할 때 아래의 키워드와 중복되지 않도록 유의하시기 바랍니다.

알파벳	키워드
a	array
b	bool boolean break
c	case const continue
d	default define die do
e	echo else elseif exit
f	float for function
g	global goto
i	if int integer include include_once
p	print
r	return
s	static string switch
w	while

부록

메모리 관련

- 선언 가능한 최대 변수 개수
  PHPoC는 메모리 용량이 제한적이므로 변수의 사용에 필요한 메모리 용량 또한 제한되어 있습니다. 따라서 변수의 개수와 문자열 변수의 길이는 제한되며, 문자열 변수 크기가 늘어날수록 이와 반비례하여 사용 가능한 개수는 줄어듭니다.

부록

오류 메시지 (Error Messages)

PHPoC는 사용자 디버깅을 위해 다양한 오류 메시지를 콘솔로 출력 합니다.

에러 메시지
address already in use
argument count mismatch
cannot break/continue N level(s)
'case' or 'default' expected
device or resource busy
divided by zero
duplicated name
expression syntax error
file name too long
file not found
function not implemented
integer number too large
invalid argument
invalid constant name
invalid device or address
maximum execution time exceeded
missing operator
missing terminating character ''' or '"'
modifiable value required
only variable can be passed by reference
operation not permitted
out of memory
string too long
syntax error
syntax error, unexpected array [, expecting 'token']
syntax error, unexpected character
syntax error, unexpected 'character' [, expecting 'character']
syntax error, unexpected end of file
syntax error, unexpected 'name' [, expecting 'character']
syntax error, unexpected number [, expecting 'character']
syntax error, unexpected 'operator' [, expecting 'token']
syntax error, unexpected string [, expecting 'character']
syntax error, unexpected 'token' [, expecting 'token']
syntax error, unexpected variable [, expecting 'character']
too many open files
undefined name
undefined offset
unsupported argument type
unsupported operand type
unsupported operator
unsupported pid
unsupported type juggling
'while' expected

변경내역

210115 (F/W: 2.3.1)

- 미리 정의된 변수 추가: $GLOBALS
- 일부 오류 정정 및 표현 개선