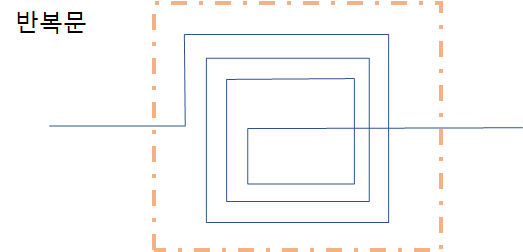
jinyPHP 1권 - 9. 반복문

**9. 반복문**

일상생활과 비슷하게 프로그램 소스를 작성하면서 반복되는 작업들을 자주 접하게 됩니다. 만일 “hello world!”을 10번을 출력한다거나, 1부터 10까지의 숫자를 출력할 때 우리는 프로그램에서 echo “hello world”; 또는 echo “1”; echo “2”; … echo “10”; 처럼 코드를 10번 작성해야만 합니다. 대부분의 프로그램 언어들은 반복되는 작업들을 쉽게 묶어 그룹화하여 처리할 수 있는 반복 문법을 지원하고 있습니다.

반복문은 프로그램 소스상에서 비슷한 내용을 여러 번 반복 처리를 수행합니다. 반복 기능을 통해 같은 코드를 중복해서 나열하지 않고 반복 문법으로 코드를 반복 수행합니다.

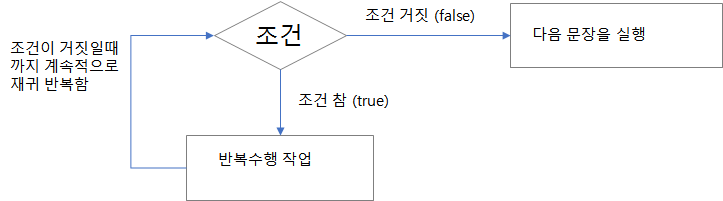


프로그램 언어에서 반복문을 사용하는 이유는 연속하여 발생한 중복 코드의 양을 줄이고 소스의 가독성을 향상하는 데 목적이 있습니다.

PHP언어 에서는 반복 처리를 위해 네 가지의 명령 및 문법을 제공합니다.

* for
* foreach
* while
* do~while

반복 처리 문법은 약간의 작성 규칙과 구조를 가지고 있습니다. 반복 조건을 검사하는 제어부와 반복을 수행하는 소스의 블록으로 구성됩니다.



또한 반복문을 작성할 때는 반복되는 작업들이 하나 또는 여러 작업들이 순차적으로 하나의 블록으로 묶여 있어야 합니다. 만일 소스 코드가 여러 블록으로 분산되어 있다고 한다면 반복 작업을 수행할 수 없습니다.

반복문은 프로그램에서 if 조건문 다음으로 가장 많이 사용을 하는 명령 기능입니다. 반복문을 잘 이용하면 보다 구조적이고 간략하게 프로그램 코드를 작성할 수 있습니다.

**9.1 for**

for 반복문은 대부분의 언어에서 가장 기본적이고 많이 사용하는 반복 문법입니다. for 키워드를 이용한 반복문은 C 언어 스타일로 고급 언어에서는 대부분 적용이 가능한 문법 구조입니다. PHP에서도 대중적인 반복 문법인 for문을 제공합니다.

다음은 for 반복 문법의 기존 구조 표현입니다.

**다중 반복 문법**

|  |
| --- |
| <?php  for (초기값 설정; 조건; 증가 연산자) {  반복되는 수행문 1;  반복되는 수행문 2;  }  ?> |

for 문법은 for 키워드와 중괄호의 조건 블록과 대괄호의 반복 수행 코드로 작성합니다. 조건 블록은 세 개의 인자값이 세미콜론(;)으로 구분하여 입력됩니다. 또한 반복되는 수행은 중괄호 { }를 이용하여 실행할 코드의 블록을 지정할 수 있습니다.

만일 반복되는 수행문이 한 개일 때는 다음과 같이 중괄호 { }를 생략할 수 있습니다.

**단일 반복 문법**

|  |
| --- |
| <?php  for (초기값 설정; 조건; 증가 연산자) 반복되는 수행문;  ?> |

for 반복 문법은 조건 블록은 세 개의 조건 인자값이 필요합니다. 첫 번째는 초기값, 두 번째는 반복 조건, 세 번째는 증가 연산자입니다. 첫 번째 초기값과 증가 연산자는 생략할 수도 있지만 두 번째 반복 조건은 생략할 수 없습니다.

**9.1.1 초기값**

반복 문법 for문장은 중괄호로 묶여 있는 반복 코드 블록을 지정한 수만큼 반복 수행합니다. 만일 우리가 다섯 번의 반복 수행을 한다고 할 때 사람들은 추상적으로 다섯 번을 쉽게 이해할 수 있습니다. 하지만 1회, 2회, 3회, 4회, 5회 등 횟수를 카운트하기 위해서는 초기의 시작하는 값이 필요합니다. 1부터 다섯 번을 세어 5까지 셀 수도 있고, 3부터 다섯 번을 세어 7까지도 셀 수도 있습니다. 그러므로 반복문을 실행하기 위해서는 반복 횟수를 카운트를 하기 위한 초기값을 반드시 필요로 합니다.

반복문에서는 카운트를 할 때 카운트용으로 사용할 하나의 변수가 필요로 합니다. 즉, 한 번, 두 번, 세 번 등 반복되는 횟수를 저장해야 할 임시변수입니다. 반복문의 초기값이란 이 반복 횟수를 지정하는 변수에 초기값을 지정하는 것입니다.

for (**$i=0**; 조건; 증가 연산자) 반복되는 수행문;

위의 예는 초기값을 0부터 카운트한다는 표현입니다. 반복문은 초기값이 반드시 필요하지만 꼭 for 반복 조건문 안에 써야 하는 것은 아닙니다. for 반복 조건문의 첫 번째 인자인 초기값은 생략도 가능합니다.

**$i=1;**

for (; 조건; 증가 연산자) 반복되는 수행문;

위의 예처럼 조건 블록에서는 생략한 경우에는 초기값은 for반복문이 실행되기 이전에 미리 정의해주면 상관 없습니다.

초기값을 외부에서 설정할 때는 외부에서 설정한 값부터 시작해서 카운트합니다. 이러한 외부초기값 설정 방법은 for 반복문을 동적으로 수행하거나, 하나의 반복문을 나누어 실행할 수 있는 테크닉으로 사용할 수 있습니다.

for (**$i=0**; $i<5; $i++) 반복되는 수행문1;

for (;$i<10; $i++) 반복되는 수행문2;

위의 예는 두 개의 반복문을 연결하는 예입니다. 첫 번째 반복문은 초기값을 0으로 시작해서 수행문1을 다섯 번을 실행합니다. 두 번째 반복문은 초기값이 없습니다. 앞에 수행한 반복문의 초기값 임시변수 $i 를 이어서 계속 사용합니다.

따라서 수행문2가 두 번째 반복문을 통해 다섯 번을 실행하고 종료하게 됩니다. 즉, 수행문1 과 수행문 2를 각각 다섯 번씩 동작하게 됩니다.

반복 수행에 필요한 초기값 변수는 하나 이상의 다수의 변수를 한 번에 초기화할 수 있습니다.

위의 예는 초기값 변수 $i, $j, $k 세 개를 한 번에 for문에서 초기화할 수 있습니다. 다수의 변수를 초기화 값을 지정하고 싶다면 콤마(,)를 이용하여 구분하면 됩니다.

초기화된 변수는 꼭 반복문 안에서 사용해야만 하는 것은 아닙니다. 다른 처리 로직에서 필요로 하는 변수들도 for 반복 문법을 통해 초기화하여 사용할 수도 있습니다.

|  |
| --- |
| for ($i=1,$sum=0;$i<=10;$i++) {  $sum += $i;  } |

위의 예처럼 $sum 한수는 반복 수행에 관련된 변수는 아니지만, 반복 수행 변수의 합계를 구하기 위한 임시변수로 초기화 조건에서 같이 0으로 초기화할 수 있습니다.

**9.1.2 증가 연산자**

반복문이 여러 번의 코드를 수행할 때 반복 횟수는 어떻게 카운트할까요? 바로 for문의 세 번째 인자값을 증가 연산자 형태를 이용하여 반복을 수행하면서 카운트의 값을 한 번씩 변경합니다. 대부분 반복 수행을 할 때 카운트의 숫자를 1씩 증가를 합니다. 꼭 증가만 하는 것은 아닙니다. 감소할 때는 감소 연산자를 사용하면 됩니다.

for ($i=0; 조건; **$i++**) 반복되는 수행문;

for ($i=0; 조건; **$j--**) 반복되는 수행문;

단순한 반복 작업 이라면 증가 연산자를 이용하여 한 번에 1씩 더하거나 빼면서 카운트를 합니다. 하지만 꼭 1씩 써야만 하는 것은 아닙니다. 만일 증가되는 카운트 값이 1이 아닌 값을 처리해야 될 경우,

**$i + = 증가 값**

형태로 바꿔서 값을 변경할 수 있습니다. $i+=2 형태로 쓴다고 하면 짝수 값 형태로 초기값 임시변수의 값이 증가될 것입니다.

초기값 변수가 생략이 가능한 것처럼 증가 연산자 조건도 필요에 따라서 생략이 가능합니다. 대신에 증가 연산자의 값은 반복 수행하는 코드 블록 안에 삽입해야 합니다.

for ($i=0; 조건;) {

**$i++;**

반복되는 수행문;

}

본문 안에 증가 연산자를 넣어서 작성하면 반복의 수행 횟수를 반복문 내부에서 제어할 수 있는 장점이 있습니다. 특정 조건 등을 넣어서 일부러 조건 값 이상의 값으로 설정한다면 반복문을 조기에 마치고 탈출할 수도 있을 것입니다.

만일 증가 연산자를 for문 조건 인자값, 본문 코드 블록 모두 사용하지 않으면 프로그램은 for문을 탈출하지 못하고 무한적으로 반복하기 때문에 컴퓨터에 부하를 주거나 프로그램이 동작을 하지 않는 것처럼 보일 수 있습니다.

**9.1.3 조건**

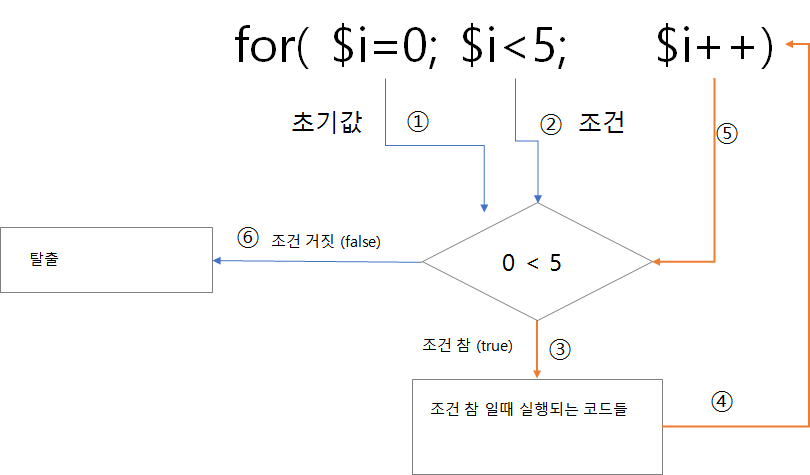
for 반복 문법의 조건인자는 for문의 소스를 반복하는 조건 횟수를 판별하는 if와 같습니다. 첫 번째 인자의 초기값으로 시작하여, 세 번째 인자의 설정값처럼 증가/감소하면서 반복하다가 두 번째 인자의 조건이 되면 반복문을 종료하게 됩니다.

for ($i=0; **$i<5**; $i++) 반복되는 수행문;

반복 조건은 등호(=)와 크기를 비교하는 부등호(<, >)와 같은 연산 기호를 사용하여 반복 횟수를 비교할 수 있습니다. 또한 조건이 거짓일 경우에 반복문을 종료하고 다음 소스 코드 실행으로 이동합니다.

**9.1.4 반복 플로우**

지금까지 배운 for 반복 문법의 조건 블록을 순차적으로 한번 그림으로 정리해 봅니다.



① for 반복문이 실행될 때 사용자가 설정한 초기값을 가지고 옵니다.

② 초기화 변수의 초기값과 조건을 비교합니다

③ 처음 시작 시 초기 조건이 만족하면 { } 블록 코드를 실행합니다.

④ 초기화 변수의 값을 변경합니다.

⑤ 변경된 초기화 변수의 증가 값과 조건을 비교합니다.

③ 증가된 조건이 만족하면 다시 한번 { } 블록 코드를 재실행합니다.

④ 초기화 변수의 증가 값을 변경합니다.

⑤ 변경된 증가 값과 조건을 비교합니다.

⑥ 증가된 조건이 거짓이면 반복문을 종료합니다.

**9.1.5 예제**

몇 가지 예제를 통하여 for 반복문의 실행과 동작을 숙지하도록 하겠습니다. 반복문은 많이 사용해야만 익숙해지고 쉽게 소스 코드를 이해할 수 있습니다.

**예제 파일** for-01.php

|  |
| --- |
| <?php  for ($i = 0; $i < 5; $i++) {  echo "The number is: $i <br>";  }  ?> |

**결과**

The number is: 0

The number is: 1

The number is: 2

The number is: 3

The number is: 4

가장 전형적인 for 구문입니다. 첫 번째 인자로 초기화 변수 $i 를 0으로 초기화합니다.

두 번째 인자로 조건 $i가 5보다 작을 때까지 해당 코드가 반복 수행됩니다.

세 번째 인자로 매번 반복될 때마다 임시변수 $i 값을 1씩 증가시킵니다.

이런 세 가지 조건을 통해 본문이 5번 반복되어 내용을 출력합니다.

두 번째 예는 초기값 변수와 증가 연산을 제외한 방법의 예제입니다.

**예제 파일** for-02.php

|  |
| --- |
| <?php  // for문 밖에서 초기값을 지정합니다.  $i=1;  // for문에서는 초기값. 증가 값 설정 부분을 생략합니다.  for ( ; $i < 5; ) {  echo "The number is: $i <br>";  // 한 번 반복할 때마다 1씩 값을 증가합니다.  $i++;  }  ?> |

**결과**

The number is: 1

The number is: 2

The number is: 3

The number is: 4

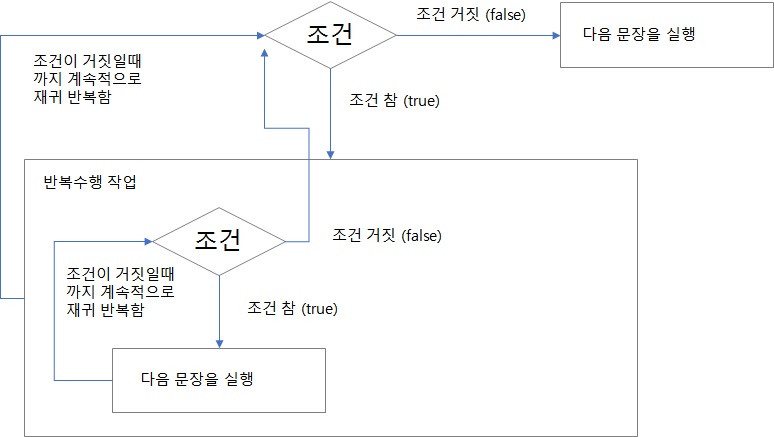
위의 예제는 초기값을 for문 실행 전에 반복문 상단 밖에 지정합니다. 증가 연산 값은 본문 중괄호 { } 블록 안에 넣은 방법입니다.

for 반복 문장은 프로그램에서 반복문의 가장 전형적인 문법이고 표준으로 자리잡고 있습니다. for 반복 문법 하나만 잘 숙지해도 대부분의 언어에서 반복 문법을 쉽게 적용하여 사용할 수 있습니다.

**9.2 다중 반복문**

반복문은 여러 소스코드를 묽어 반복처리를 수행합니다. 반복문 또한 소스코드 이기 때문에 다른 반복문의 반복 수행되는 코드가 될 수 있습니다. 다중 반복문이란 하나의 반복문이 또 다른 반복문을 포함하는 형태를 말합니다.

그림으로 이중 반복문을 표현하면 다음과 같습니다.



반복문의 중복의 횟수는 제한이 없습니다. 반복문이 중첩이 될수록 반복문1 x 반복문2 x 반복문3 곱샘 형태로 수행의 횟수는 증가합니다.

다중 반복문을 작성할 때는 각각의 반복문의 초기화 임시변수에 대해서 신경을 많이 써야 합니다. 대부분 다중 반복문 각각의 서로 다른 초기화 변수명을 사용합니다. 만일 초기화 변수값이 같다면 서로 영향을 미쳐서 생각 이외의 결과가 출력될 수 있습니다.

**9.2.1 일차원 반복**

일반적으로 기존 한 개의 반복문은 일차원적인 반복을 처리합니다. 이를 일차원 반복문이라고 합니다.

예를 들어 “헬스장에서 코치가 윗몸 일으키기 50번 하세요”라고 한다면 윗몸 일으키기라는 코드를 50번 수행하게 되는 것입니다.

**예제 파일** for-03.php

|  |
| --- |
| <?php  for ($i = 0; $i < 50; $i++) {  echo "윗몸 일으키기<br>";  }  ?> |

**결과**

윗몸 일으키기

윗몸 일으키기

윗몸 일으키기

...

윗몸 일으키기

윗몸 일으키기

윗몸 일으키기

윗몸 일으키기

**9.2.2 이차원 반복**

이차원 반복문이란 반복문이 또 다른 반복문을 하나 더 가지고 있는 경우를 말합니다. 아래 작성 문법처럼 for반복문 안에 또 다른 for 반복문이 포함되어 있습니다.

**|문법|**

|  |
| --- |
| <?php  // 1차 반복문  for (초기값 설정; 조건; 증가 연산자) {    // 2차 반복문  for (초기값 설정; 조건; 증가 연산자) {  }  }  ?> |

“헬스장에서 코치가 윗몸 일으키기 10번씩하고 1분 쉬고 다시 10번씩 하는 동작을 5번 하세요”라고 했을 때 총 수행하는 동작은 10번\*5세트=50번 하는 것과 동일하지만, 중간에 휴식이 있기 때문에 하나의 반복문으로는 처리할 수 없습니다.

이런 경우 이중 반복문을 이용하여 10번의 윗몸 일으키기 반복문을 또 다시 5번을 수행을 하면 됩니다.

**예제 파일** for-04.php

|  |
| --- |
| <?php  // 반복 루프1  for ($i = 0; $i < 5; $i++) {  echo "세트$i 시작 <br>";  // 반복 루프2  for ($j = 0; $j < 10; $j++) {  echo "윗몸 일으키기<br>";  }  echo "=== 휴식 10분 === <br>";  }  ?> |

**결과**

세트0 시작

윗몸 일으키기

...

윗몸 일으키기

=== 휴식 10분 ===

세트1 시작

윗몸 일으키기

...

윗몸 일으키기

=== 휴식 10분 ===

세트2 시작

윗몸 일으키기

...

윗몸 일으키기

=== 휴식 10분 ===

세트3 시작

윗몸 일으키기

...

윗몸 일으키기

=== 휴식 10분 ===

세트4 시작

윗몸 일으키기

...

윗몸 일으키기

=== 휴식 10분 ===

이처럼 반복문 안에 또 다른 반복문을 넣어 작성할 수 있습니다. 이때 주의해야 할 점은 1차 반복문에서 사용하는 제어변수와 2차 반복문에서 사용하는 제어변수는 달라야 합니다.

만일 값이 같으면 앞에서 1차 반복문에서 증가 연산 값과 2차 반복문의 증가 연산 값이 이중으로 작동하기 때문에 정상적인 결과 값이 출력되지 않습니다.

이중 반복문은 다차원 배열 등의 값을 처리하는 데 매우 유용합니다. 다차원 배열은 하나의 배열 각각의 값에 또 다른 배열을 포함하고 있기 때문에 $arr[][] 2개 이상의 인자값이 필요합니다.

**예제 파일)** for-05.php

|  |
| --- |
| <?php  $cars = array(  array("Volvo",10,300),  array("BMW",11,250),  array("Saab",12,350),  array("kia",13,200)  );  for ($i = 0; $i < 4; $i++) {  for ($j = 0; $j < 3; $j++) {  echo $cars[ $i ][ $j ];  echo " ";  }  echo "<br>";  }  ?> |

**결과**

Volvo 10 300

BMW 11 250

Saab 12 350

kia 13 200

다음은 이중 반복문을 이용하여 삼각형 별을 만드는 전형적인 예제입니다.

**예제 파일** for-06.php

|  |
| --- |
| <?php  for ($i=0;$i<5;$i++) {  for ($j=0;$j<$i;$j++) {  echo "\*";  }  echo "<br>";  }  ?> |

**결과**

\*

\*\*

\*\*\*

\*\*\*\*

다음 예제는 이중 반복문을 통해 정삼각형 별을 만드는 전형적인 예제입니다.

**예제 파일** for-07.php

|  |
| --- |
| <?php  for ($i=0;$i<5;$i++) {  for ($j=4; $j>$i-1;$j--) {  echo "\_";  }  for ($k=1; $k<$i+1;$k++) {  echo "\*";  }  for ($m=1; $m<$i;$m++) {  echo "\*";  }  echo "<br>";  }  ?> |

**결과**

\_\_\_\_\_

\_\_\_\_\*

\_\_\_\*\*\*

\_\_\*\*\*\*\*

\_\*\*\*\*\*\*\*

**9.2.3 다차원 반복**

반복문은 이중 반복문 외 더 많은 반복문을 포함할 수 있습니다. 즉 이차원, 삼차원, 사차원… 등 계속하여 반복문 삽입이 가능합니다. 이를 다차원 반복문이라고 합니다.

**9.3 foreach**

foreach는 간단하게 사용할 수 있는 반복 문법입니다. 기존 for 반복문과 달리 간단하게 배열값 등을 이용하여 반복문을 처리할 때 매우 편리합니다.

foreach는 기존 for 반복 문법처럼 반복 횟수를 정의하는 카운터가 없는 것이 특징입니다. 기존에 초기값 및 반복 증가, 조건 등의 횟수를 지정하는 것이 아니라 특이하게 입력된 객체 집합의 개수만큼 반복 실행하게 됩니다.

foreach의 장점은 간단한 배열 등을 처리하는 데 있어서 초기값, 증가, 조건 등을 잘못 적용하여 발생할 수 있는 오류를 방지하고, 단순하게 반복 처리 코드를 실행할 수 있습니다.

**|문법|**

|  |
| --- |
| foreach ($array as $value) {  반복 실행 코드;  } |

foreach 반복문은 key와 value 값의 한 쌍의 구조를 응용하여 처리하는 반복문입니다.

다음 예제는 어레이 배열의 포인터를 하나씩 증가하면서 마지막 엘리먼트까지 이동하면서 데이터를 출력하는 반복문입니다.

**예제 파일)** foreach-01.php

|  |
| --- |
| <?php  $colors = array("red", "green", "blue", "yellow");  foreach ($colors as $value) {  echo "$value <br>";  }  ?> |

**결과**

red

green

blue

yellow

위 예제에서 배열변수 $colors 에서 값을 하나씩 찾아서 as 키워드 다음에 정의한 $value 변수명에 값을 하나씩 저장합니다. $value의 값은 반복이 수행되면서 $color 의 값으로 한 번씩 변경됩니다.

위의 예제 foreach문법을 기본 for문으로 다시 구현 정의해보면 다음과 같습니다.

**예제 파일** foreach-02.php

|  |
| --- |
| <?php  $colors = array("red", "green", "blue", "yellow");  for($i=0;$i<count($colors);$i++){  echo $colors[$i]."<br>";  }  ?> |

**결과**

red

green

blue

yellow

위의 두 개의 문법은 같은 동작을 처리합니다. 하지만 배열 객체의 경우 foreach 문이 좀 더 간결하게 처리할 수 있는 것을 확인할 수 있습니다.

**9.3 while**

while 반복문은 for문과 같이 두 번째로 가장 많이 사용하는 반복 문법입니다. while은 for 문장과 달리 초기값과 증가 연산자 등의 반복 횟수를 제어하는 인자값이 없습니다. while의 인자값은 참의 값을 갖는 인자값 하나만 있습니다.

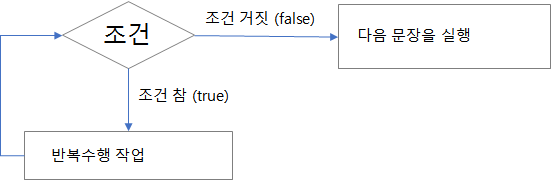
반복 조건이 거짓(false) 조건이 나올 때까지 계속 반복합니다. while문은 동작의 반복 횟수를 사전에 정하지 않기 때문에 조건이 만족될 때까지 반복을 처리해야 하는 경우 매우 유리합니다. 하지만 조건 처리하는 부분이 반복 소스 안에 포함되지 않으면 반복문은 종료하지 않고 계속 무한정 실행됩니다.

다음은 기본적인 while 반복문 문법입니다.

**|문법|**

|  |
| --- |
| while (참 조건) {  반복 실행 코드;  } |

문법 구조에서 보듯이 조건과 반복 실행되는 소스의 블록 { }만 있습니다. while문의 동작을 그림으로 알아보면 다음과 같습니다.



다음은 외부에 반복되는 초기 임시변수 값을 설정하고 while의 조건이 $i<=5가 될 때까지 반복수행하는 예제입니다.

**예제 파일** while-01.php

|  |
| --- |
| <?php  $i = 1;  while(**$i <= 5**) {  echo "The number is: $i <br>";  $i++;  }  ?> |

**결과**

The number is: 1

The number is: 2

The number is: 3

The number is: 4

The number is: 5

while 반복문 블록 {} 안에는 조건을 증가시키는 $i++ 증가 연산자가 포함되어 있습니다. 반복문이 한 번씩 수행될 때마다 내부에서는 조건값을 한 개씩 증가시키면서 코드를 수행하게 됩니다.

위의 예제는 외부에서 정의된 초기화 변수를 통하여 반복횟수를 정의하는 데 유용합니다. 어떻게 보면 for 구문에서 첫 번째 인자와 세 번째 인자를 입력하지 않고 작성한 다음 예제와 비슷할 수 있습니다.

**예제 파일** while-02.php

|  |
| --- |
| <?php  $i = 0;  for ( ; $i < 5; ) {  $i++;  echo "The number is: $i <br>";  }  ?> |

**결과**

The number is: 1

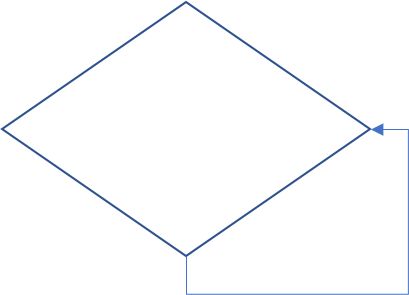
The number is: 2

The number is: 3

The number is: 4

The number is: 5

**9.3.1 무한 루프**

무한 루프란 프로그램상에서 조건이 **항상 참인 상태**의 반복문을 말합니다. 즉, 조건이 언제나 참(true)이기 때문에 거짓(false)이 나올 수 없습니다. 무한 루프는 프로그램을 강제로 종료하거나 반복문을 사용자가 종료시키기 전까지는 반복되는 코드를 무한정으로 실행합니다.

무한 반복 루프는 외부와의 통신 대기를 하거나, 사용자가 언제 입력할지 모르는 값을 대기하거나 모니터링할 때 매우 유용합니다.

무한 반복 사용은 매우 주의하여 사용해야 합니다. 무한 루프를 사용하더라도 항상 무한 루프를 탈출할 수 있는 처리 로직도 같이 만들어주는 것이 좋습니다. 잘못 사용하면 컴퓨터에 과부하를 주거나, 프로그램이 중단되는 것 같이 보이는 문제가 발생합니다.

**예제 파일)** while-03.php

|  |
| --- |
| <?php  While(**1**){  echo "무제한으로 반복합니다.";  }  ?> |

**결과**

무제한으로 반복합니다

무제한으로 반복합니다

무제한으로 반복합니다

무제한으로 반복합니다

….

….

…

..

위의 무한 루프 예는 조건 값이 1(true)입니다. 1이라는 값은 상수 값으로 조건이 거짓(false)으로 변경이 될 수가 없습니다. 이런 경우 while 반복문은 무한 루프를 돌게 됩니다.

그럼 무한 루프를 탈출하려고 한다면 어떻게 해야 할까요? break;문을 사용하면 됩니다.

break; 문장은 { }으로 싸인 블록 한 단계를 탈출합니다. 보다 자세한 예는 이후의 break 설명에서 다시 합니다.

**9.3.2 while 중첩**

while 반복문도 for반복문법처럼 중첩하여 다중 반복문을 구현할 수 있습니다.

|문법|

|  |
| --- |
| while (조건) {  실행1;  while (조건) {  실행2;  while (조건) {  실행3;  }  }  } |

while문을 통한 중첩 반복을 할 때는 프로그램이 무한 루프에 걸리지 않도록 탈출 조건을 잘 계산하여 처리해야 합니다.

**9.4 do..while**

while문은 또 다른 타입인 do..while 반복문법인 선순의 반복문을 지원합니다. while 반복문은 처음 반복을 수행기 전에 조건이 참(true)인 경우에만 반복문법 코드가 실행이 됩니다.

하지만 반복문을 수행하기 전에 최소한 한 번은 반드시 수행하고, 이후에 조건에 따라서 반복을 수행해야 할 경우가 코드 작성 시 종종 필요하게 됩니다.

다음은 do while문의 작성 문법입니다.

**|문법|**

|  |
| --- |
| do{  } while(참 조건); |

do..while의 반복 문법을 보면 while과 다르게 while(참 조건); 부분이 { } 하단에 적혀 있는 것을 확인할 수 있습니다. 즉, 조건을 검사하기 전에 한번은 { } 부분을 처리하라는 의미입니다.

다음 예제는 $x 값이 1부터 5까지 출력하는 반복문입니다. 반복문 안에서는 $x 값을 1씩 더해주면서 화면에 숫자를 표시합니다. 그리고 5 값이 넘으면 while 반복문을 종료합니다.

**예제 파일** dowhile-01.php

|  |
| --- |
| <?php  $x = 1;  do {  echo "The number is: $x <br>";  $x++;  } while ($x <= 5);  ?> |

**결과**

The number is: 1

The number is: 2

The number is: 3

The number is: 4

The number is: 5

하지만 아래 예처럼 $x의 초기값이 6일 때는 어떻게 될까요? do..while문에서는 먼저 6 값을 화면에 출력하고 while 조건문을 검사합니다. $x 값이 5보다 크기 때문에 다음 반복을 하지 않고 끝나게 됩니다.

**예제 파일** dowhile-02.php

|  |
| --- |
| <?php  // 초기값을 6으로 설정합니다.  $x = 6;  do {  echo "The number is: $x <br>";  $x++;  } while ($x <= 5);  ?> |

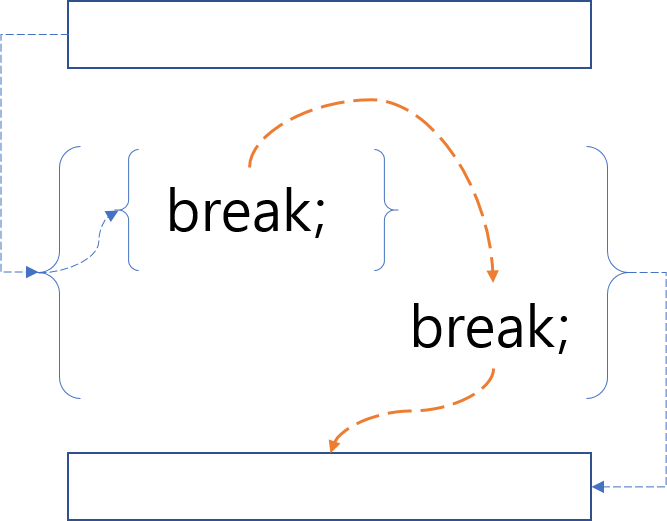
**결과**

The number is: 6

즉, 조건과 상관없이 { } 안의 명령을 한 번은 수행을 한다는 특징이 있습니다.

**9.5 break**

break; 명령문은 앞서 설명한 조건 switch문과 더불어 반복문의 제어 흐름을 처리하는 데도 종종 사용됩니다.



break;문의 특징은 break;를 감싸고 있는 조건이나 반복문의 중괄호 { }를 빠져나온다는 것입니다.

**9.5.1 무한 루프 탈출**

while 반복문을 통해 무한 루프 동작을 하고 있을 때 프로그램은 어떻게 무한 루프를 탈출할 수 있을 까요? while문의 반복되는 코드 블록 중괄호 {} 안에 break;를 처리하면 무한 반복문의 루프를 탈출할 수 있습니다.

**예제 파일** break-11.php

|  |
| --- |
| <?php  while(1){    if ($i<10) {  echo "10보다 작으면 무제한 반복합니다.<br>";  } else {  echo "조건이 10보다 크기 때문에 조건을 빠져나갑니다.<br>";  break;  }    $i++;  }  ?> |

**결과**

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

10보다 작으면 무제한 반복합니다.

조건이 10보다 크기 때문에 조건을 빠져나갑니다.

위의 예를 보면 while 반복문의 조건 값은 상수 1입니다. 항상 참(true) 조건입니다. 따라서 while문은 중괄호 { } 내용을 프로그램을 강제로 종료하기 전까지는 반복합니다.

하지만 반복문 안에 if 조건을 넣어서 또 다른 내부 조건이 만족하게 되면 break;문을 실행합니다. 따라서 무한 반복문이더라도 $i 값을 1씩 증가하면서 화면에 출력하다가 10이 넘으면 break문을 만나서 무한 루프를 탈출하게 됩니다.

무한 루프는 외부 통신 등 값을 기다리면서 반복을 처리하는 동작에 매우 유용합니다. 다음은 파일 읽기 예제입니다.

**예제 파일** break-12.php

|  |
| --- |
| <?php  $myfile = fopen("readme.txt", "r");    while(1) {  if (!feof($myfile)) {  echo fgets($myfile) . "<br>";  } else {  break;  }  }  fclose($myfile);  ?> |

보통 파일의 크기는 매우 다양해서 알 수가 없습니다. 크기가 다들 크기들이 제각각입니다. 이럴 때 while의 무한 반복문은 편리합니다. 파일을 글자 하나씩 읽어서 화면에 출력합니다. 이러한 동작을 무한 루프를 통해 반복합니다.

만일 읽은 파일의 끝을 나타내는 EOF 문자를 만나면 파일 화면 출력 무한 루프를 끝내고 break;를 처리하여 반복을 중단합니다.

**9.5.2 반복문 강제 종료**

break;문은 while 반복문 이외에 for 반복문에서도 사용이 가능합니다. for처럼 반복의 횟수를 정하여 반복 작업을 하더라도 특정한 조건에 따라 더 이상 반복을 처리하지 않아도 될 때 break;문을 통해 for 반복문을 종료, 탈출할 수 있습니다.

**예제 파일** break-13.php

|  |
| --- |
| <?php  for ($i=0;$i<10;$i++) {    if ($i>=6) {  break;  }  echo $i."<br>";  }  echo "종료";  ?> |

**결과**

0

1

2

3

4

5

종료

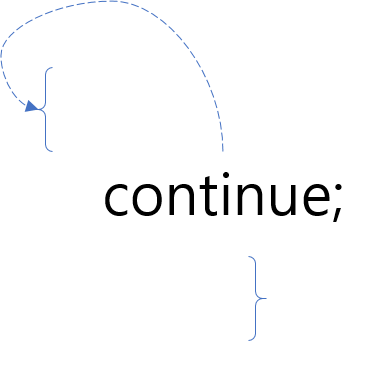
위의 예를 보면 for 반복문을 이용하여 프로그램은 0부터 9까지의 숫자를 한 줄씩 출력합니다. 하지만 실제 출력은 0~5까지만 출력이 됩니다. 왜 그럴까요?

정상적으로 10번 반복하지만 중간에 $i 값을 검사하여 6 값보다 크면 for문을 끝내고 다른 명령으로 처리되어 넘어갑니다.

이처럼 break;를 이용하면 반복문을 강제로 종료할 수도 있고 불필요한 작업을 방지하고 반복 루트를 중간에도 탈출을 할 수 있습니다.

**9.6 continue**

continue;는 break;와 비슷하게 제어의 흐름을 변경할 수 있는 명령 키워드입니다. break;는 중괄호 { }를 탈출하는 것이라면, continue;는 중괄호 { } 처음으로 돌아가라는 의미입니다.



소스를 해석하면서 continue; 명령을 만나게 되면 반복되는 소스 코드 중괄호 { }의 시작 부분으로 이동합니다. 즉, 반복문을 처리하는 데 있어서 특별한 조건에 따라서 이후 코드를 처리하지 말고 다시 처음부터 반복을 이어서 하라는 의미입니다.

다음은 for 반복문에서 continue를 사용하는 예제입니다.

**예제 파일** continue-01.php

|  |
| --- |
| <?php  for ($i=0;$i<10;$i++) {    if ($i%2 == 0) {  continue;  }  echo $i."<br>";  }  echo "종료";  ?> |

**결과**

1

3

5

7

9

종료

위의 예제에서는 0부터 9까지의 숫자를 출력하는 데 있어서 $i가 2로 나눈 나머지 값이 0인 경우에는 continue;를 처리합니다.

2로 나눈 나머지가 0이라는 의미는 $i가 짝수라는 의미입니다. 짝수일 때는 그 이후의 작업을 하지 않고 for문의 상단으로 이동하게 됩니다. 즉, 짝수는 화면에 출력하지 않겠다는 의미입니다.

continue;는 프로그램에서 불필요한 동작을 하지 않고 리소스를 절약할 수 있는 기능 중에 하나입니다.