jinyPHP 1권 - 10. 함수

**10 함수**

프로그램은 개발자가 한 줄씩 코드를 에디터에 작성하면서 실행과 테스트를 반복합니다. 이렇게 작성된 절차적 프로그램들은 첫 줄부터 순차적으로 해석하여 실행하게 되고 결과를 화면에 출력하게 됩니다.

간단하게 몇 줄의 기능으로만 처리가 가능하다면 기존 방식과 같은 절차적 순서로 작성이 가능할 수도 있을 것입니다. 하지만 본격적으로 실전 기능을 개발하게 되면 프로그램의 기능들은 구체적이고, 복잡하게 되고 또한 소스 코드도 길어지는 현상이 발생합니다. 심지어 중복된 코드와 기능이 발견되기도 합니다.

한 개의 파일 안에 모든 내용을 절차적으로 작성하는 것은 유지보수와 코드의 효율적인 측면에서는 적합한 개발 방법론이 아닐 수 있습니다. 한 개의 수정을 위해서 소스 코드를 전반적으로 수정해야만 하는 불편함이 있고, 소스의 기능들을 한눈에 구분하여 처리할 수도 없을 것입니다.

이는 마치 컴퓨터에서 폴더를 사용하지 않고 모든 파일을 관리하는 것과 같습니다. 따라서 대부분의 언어들은 컴퓨터의 폴더와 같이 함수라는 기능을 지원합니다. 함수는 코드들을 그룹화하고 중복된 내용들을 함수 코드로 재사용함으로써 코드의 유지보수와 개발 과정을 편리하게 사용할 수 있는 소프트웨어 공학적인 컨셉입니다.

**10.1 함수란?**

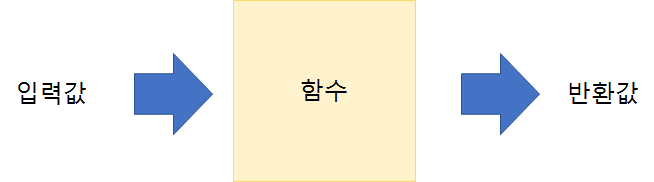
함수는 코드의 중복된 내용들을 하나의 블록으로 분리하는 작업 과정입니다. 또한 분리된 함수들은 다시 사용할 수 있도록 포장하는 과정이라고 이해하면 쉽습니다.

이렇게 코드들은 함수화 같은 분리 작업을 통해 중복된 소스의 크기를 줄일 수 있는 장점도 있습니다. 또한 이러한 함수형 코드 개발 방법론은 소프트웨어를 고도화함으로써 보다 유지보수가 편리해지고 성능도 향상될 수 있습니다.

대부분의 프로그램은 함수의 집합으로 동작을 합니다. 따라서 어떠한 언어를 통해 프로그램을 개발하기 위해서는 함수의 사용법은 반드시 알아야 하는 필수 프로그램 기법입니다.

함수는 크게 두 가지 종류로 구분할 수 있습니다. 내장 함수, 사용자 정의 함수입니다. 내장 함수는 프로그램 자체적으로 제공되는 기본 함수로 프로그램 언어가 소스 코드를 개발하는 데 필요한 기초적인 기능을 함수화하여 제공합니다.

내장 함수는 언어의 참고 레퍼런스를 통해 찾아볼 수 있습니다. 사용자 정의 함수는 기본적으로 언어에서 제공하는 함수 이외에 자신만의 처리 로직 및 여러 함수의 집합을 사용자가 임의적으로 만들어 사용하는 함수를 말합니다.



함수는 하나의 블랙박스와 같습니다. 값을 입력하면 데이터를 처리하거나 처리한 결과의 값을 출력합니다.

함수는 또 다른 함수를 실행을 할 수 있습니다. 함수의 계층적인 실행으로 먹이사슬처럼 작은 함수들을 먼저 만들고 이를 이용한 좀 더 큰 함수, 그보다 좀 더 큰 함수 형태로 만들어 갑니다. 이러한 과정을 최적화한다고 표현하기도 합니다.

**10.2 내장 함수**

프로그램 코딩을 하다 보면 수많은 함수들이 등장합니다. 수많은 함수 중에서는 PHP에서 편의상 미리 만들어 놓은 함수인 내장 함수와 개발자가 자체적으로 만들어 사용하는 사용자 정의 함수 두 개로 구분할 수 있습니다.

대부분의 초보 개발자들은 PHP의 내장 함수를 사용하는 경우가 많이 있습니다. 개발자로서 먼저 PHP에서 제공하는 내장 함수에는 어떤 것이 있는지를 확인하고 사용법을 습득하는 것은 매우 중요합니다. 이런 내장 함수들을 먼저 확인해야 하는 것은 추후 사용하는 사용자 정의 함수와 내장 함수를 구분하기 위해서입니다. PHP에서 제공하는 내장 함수들은 PHP 공식 사이트 www.php.net에서 확인할 수 있습니다.

일부 괜찮은 에디터의 경우 내장 함수와 사용자 함수를 확인하여 색상을 표시하는 기능을 제공하는 개발 툴도 있습니다. 하지만 이런 개발 툴에 의존하지 않고 내장 함수들을 많이 안다는 것은 초보 개발자를 넘어서 수많은 다른 사람들의 소스를 이해하는 데 매우 유용합니다.

함수명은 중복된 이름으로 작성할 수 없습니다. 내장 함수의 이름들은 기존 PHP에서 이미 한 번 사용한 함수로서 사용자 정의 함수로 사용할 수 없습니다. 만일 동일한 이름의 함수명을 사용하여 작성한 경우에는 실행 시 에러가 발생할 것입니다.

이러한 이유 또한 내장 함수들을 숙지하고 학습을 먼저 해야 하는 이유도 될 것입니다.

PHP 내장 함수의 보다 자세한 사용법은 공식 사이트 www.php.net에서 매뉴얼을 확인할 수 있습니다.

**함수 체크**

PHP 공식 사이트를 통해 정의된 함수들을 확인할 수 있습니다. 또는 사용자 함수를 많이 만들었을 때 함수명이 중복되는지 어떻게 확인할 수 있을까요? PHP는 내장 함수 및 사용자가 생성하는 함수들이 존재하는지 프로그램에서 검색할 수 있는 내장 함수를 제공합니다.

|관련함수|

|  |
| --- |
| bool **function\_exists** ( string $function\_name ) |

function\_exists() 내장 함수는 내장 함수와 사용자 함수 모두 함수명의 존재 여부를 확인하여 논리 값으로 반환합니다. 함수가 존재하면 true(1)과 없는 경우에는 false(0)을 반환합니다. 이렇게 반환된 값을 if 조건문을 통해 함수를 체크할 수 있습니다. 이처럼 함수 존재 여부를 체크하는 내장 함수를 통해 함수를 호출할 때 발생할 수 있는 호출 오류를 사전에 방지할 수 있습니다.

**예제 파일** func-01.php

|  |
| --- |
| <?php  $functionName = "intval";  if (function\_exists($functionName)){  echo "$functionName 함수가 존재합니다. <br />\n";  } else {  echo "$functionName 는 없는 함수입니다.<br />\n";  }  ?> |

**결과**

intval 함수가 존재합니다.

**10.3 함수의 선언**

PHP 언어는 내장 함수 이외에 개발자가 자체적인 사용자 함수를 선언하여 사용할 수 있습니다. 다른 언어에서는 함수의 정의와 사용 위치의 순서를 규정하기도 하나, PHP에서는 함수를 선언하는 위치는 프로그램의 상단 또는 하단 어느 곳에 정의해도 상관없습니다.

PHP는 인터프리터 언어는 일단 프로그램 소스의 처음부터 끝까지 코드를 로딩하여 살펴보고 소스상의 함수 사용을 매칭합니다.

**10.3.1 함수 정의 키워드**

프로그램에서 함수를 정의하는 키워드는 function입니다. 대부분의 언어에서도 이와 유사한 function 키워드를 자주 사용합니다. PHP 언어는 프로그램 코드를 분석하면서 function이라는 키워드를 만나게 되면 PHP는 함수를 선언하는 것으로 인지합니다.

사용자 정의 함수는 function이라는 함수를 정의하는 키워드와 사용자가 임의로 정의한 함수의 이름을 정의하여 생성할 수 있습니다.

**함수 선언 문법**

|  |
| --- |
| function **함수명** (인자1, 인자2, 인자3 ...) {  함수 정의 몸체 ...  소스 코드 내용  } |

사용자 정의 함수 선언은 function 키워드 다음에 바로 함수명을 적으면 됩니다.

함수명 다음에는 함수의 인자값 선언부 소괄호 ()와 함수의 몸체라고 일컫는 블록 중괄호 { }로 정의됩니다.

**10.3.2 함수명**

함수명을 작명할 때는 기존의 내장 함수 이름과 중복되지 않게 주의를 해야 합니다. 중복되지 않는 상태에서 함수의 이름은 규격에 맞게 사용자가 자유롭게 작성을 할 수 있습니다.

함수명은 영문자로 작명을 합니다. 함수명은 숫자로 시작할 수 없습니다. 하지만 특수기호 밑줄(\_)로 시작하거나 중간에 넣을 수 있습니다.

**예제 파일** func-02.php

|  |
| --- |
| <?php  function helloMsg() {  echo "Hello world! <br>";  }  ?> |

위의 예제는 function 키워드로 helloMsg 함수를 정의합니다. 사용자 정의된 helloMsg 함수는 "Hello world!"라는 메시지를 출력하는 단순한 기능을 가지고 있습니다.

**10.3.3 함수 실행**

앞에서 사용자 함수를 정의했다고 해서 함수가 바로 실행되지는 않습니다. 함수의 정의와 실행은 엄밀히 말하면 서로 다릅니다. function 키워드로 함수를 선언했다고 해서 코드상의 함수가 실행되지는 않습니다. 단지 PHP는 이러한 함수가 있다고 기억만 하고 있는 것입니다. PHP에는 수많은 내장 함수들이 포함되어 있지만 이 모든 함수들이 실행되지 않는 것과 유사합니다.

프로그램을 실행하기 위해서는 함수 호출이 필요합니다. 프로그램이 function이라는 키워드로 함수를 정의한 후 프로그램에서 함수명(); 형태로 실행할 위치에서 함수 호출을 합니다. PHP 언어는 함수를 실행하는 코드를 만나야 비로소 함수가 실행된다고 보면 됩니다.

PHP에서 함수를 실행 호출하는 방법은 매우 간단합니다.

|문법|

**함수명();**

형태로 소스상에 적으면 됩니다. 즉, 선언한 함수를 호출(사용)을 해야만 함수를 PHP 코드에서 실행을 합니다.

프로그램에서 함수를 실행한다는 것은 프로그램의 제어권이 변경되는 것입니다. 프로그램이 순차적으로 동작을 하다가 함수를 만나면 제어권이 함수로 넘어가 일정한 일을 수행한 다음 호출한 함수의 위치로 되돌아갑니다.

**예제 파일** func-03.php

|  |
| --- |
| <?php  function helloMsg() {  echo "Hello world! <br>";  }  helloMsg(); // 함수를 호출합니다.  helloMsg(); // 재사용 함수 재호출합니다.  ?> |

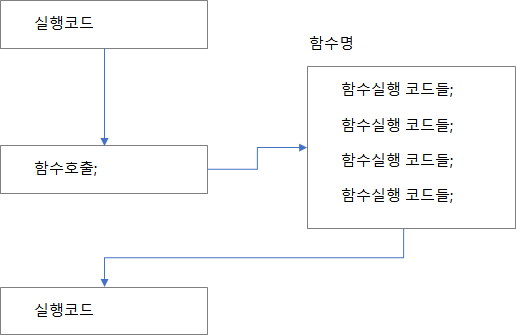
**결과**

Hello world!

Hello world!

**10.3.4 함수 실행 위치**

인터프리터 언어의 특성상 PHP는 함수를 선언하는 위치는 중요하지 않습니다. 하지만 함수의 실행 위치는 매우 중요합니다. 함수의 실행 위치는 프로그램의 제어권 변경과 또 다른 동작을 수행하기 때문입니다. 프로그램 코드가 함수를 실행 명령을 만나게 되면 프로그램의 실행 제어권이 함수로 전달됩니다. 함수 호출 시 PHP는 현재의 호출 시점을 임시로 저장한 후에 다른 함수들을 실행합니다. 즉, 함수를 호출하는 위치에서 함수의 코드들이 실행되기 때문입니다.



함수가 호출되면 그 자리에서 함수명과 동일한 함수를 실행하고 실행 전에 임시로 저장한 프로그램 제어권 자리로 다시 돌아옵니다. 이러한 제어권의 임시 저장과 함수 호출, 되돌아오는 과정을 통해 함수형 코딩으로 프로그램을 작성했다고 해서 프로그램의 처리 루틴이 꼬이거나 잘못된 경우가 발생되지 않는 것입니다.

**예제 파일** func-04.php

|  |
| --- |
| <?php  function helloMsg() {  echo "Hello world! <br>";  }  helloMsg(); // 함수를 호출합니다.  helloMsg(); // 재사용 함수 재호출합니다.  ?> |

위의 예제를 보면 화면에 "Hello world!"라는 글자가 한 줄 출력됩니다. 함수는 재사용이 가능합니다. 바로 밑에 동일한 함수 호출 helloMsg();을 만나게 됩니다.

그러면 또 다시 그 자리에서 동일한 함수명을 가진 함수를 실행하고 다시 그 위치에 돌아오게 됩니다.

위의 함수 예제는 하나의 함수 정의와 두 번의 함수 호출을 통해 두 줄의 "Hello world!" 메시지가 화면에 출력이 됩니다. 이것 또한 함수형 코딩의 장점으로 한 번 작성한 함수를 중복으로 재사용 가능하다는 것입니다.

**10.4 인자 매개변수**

함수는 마치 코드의 블랙박스와 같습니다. 함수의 작업을 의해서 기초 데이터 값을 전달해야 할 경우가 발생합니다. 이렇게 함수를 호출하는 데 필요한 기초 데이터 값을 전달해주는 것을 인자값이라고 합니다.

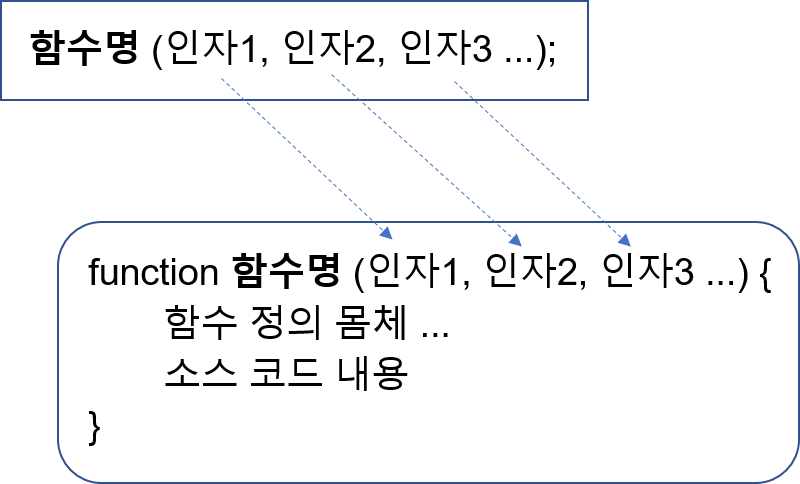
인자값은 함수에 임의의 값을 전달합니다. 인자 선언 방법은 함수 이름 다음에 오는 소괄호() 안에 필요한 기초 데이터 값을 나열하여 전달할 수 있습니다. 인자값은 직접 값을 넣을 수도 있지만 변수를 통하여 전달을 할 수도 있습니다.

**함수 선언 문법**

|  |
| --- |
| function **함수명** (인자1, 인자2, 인자3 ...) {  함수 정의 몸체 ...  소스 코드 내용  } |

앞의 함수 선언부를 다시 한번 확인해 봅시다. 함수명 뒤에 괄호 ( ) 부분이 있습니다. 함수명 뒤에는 꼭 괄호 () 부분이 같이 정의되는데 이 부분을 인자값을 지정하는 것입니다. 또한 함수 측면에서는 외부에서 들어오는 값을 매개변수라고 표현 합니다.

즉 함수가 처리를 하기 위한 데이터 값을 전달 하는 부분 입니다. 만일 전달하고자 하는 값이 여러 개인 경우에 콤마(,)로 구분하여 여러 개의 기초 데이터 값을 전달할 수 있습니다.



또한 함수는 처리 또는 호출 결과를 반환하여 이를 받아 다시 프로그램 실행에 필요한 처리를 할 수 있습니다.

**10.4.1 매개변수 선언**

함수는 외부로 들어오는 기초 데이터 값을 변수 형태로 입력받습니다. 외부에서 입력된 값을 전달받는 변수를 매개변수라고 합니다. 매개변수는 함수내에서 지역변수로 사용됩니다.

함수의 매개변수 선언는 소괄호 ( ) 안에 인자값으로 받는 변수명을 넣으면 됩니다.

|  |
| --- |
| function hellowWorld ($name, $age) {  함수 정의 몸체 ...  소스 코드 내용  echo $name . “<br>”;  echo $age . “<br>”;  } |

매개변수로 선언된 변수는 함수 몸체 안에서 별도의 선언 등을 하지 않고도 사용할 수 있습니다. 매개변수는 이미 함수 안에 정의 및 사용된 변수로 인식합니다.

인자로 전달되는 매개변수 데이터 값은 한 개 또는 여러 개를 사용할 수 있습니다. 만일 인자값이 다수일 때는 콤마(,)로 구분하여 여러 타입의 다수 데이터를 입력할 수 있습니다. PSR 코딩 스타일 규칙에 따르면 변수명 뒤에는 공백 없이 콤마(,)를 추가하며 다음 번 변수명 사이에 공백을 하나 추가합니다.

어떤 함수들은 매개변수 인자가 없는 경우도 있을 수 있습니다. 단순 기능만 수행하는 함수들이 그럴 수 있는데, 이때 함수가 매개변수가 없더라도 선언할 때 중괄호 ( )는 작성을 해야 합니다(생략이 불가능하다고 생각하면 됩니다).

또한 함수를 호출할 때도, 인자값이 없는 함수는 호출할 때 꼭 소괄호 ( )를 같이 붙여서 함수를 호출해야 합니다.

**10.4.2 내부변수와 매개변수**

함수의 매개변수로 선언된 변수의 이름은 약간의 독립된 변수입니다. 각각의 함수에 선언된 매개변수는 외부의 변수명과는 상관없습니다. 함수 밖에서 정의한 변수와 함수의 매개변수로 작성한 이름이 같다고 해서 동일한 변수는 아닙니다.

만일 동일한 함수를 두 번 이상 호출한다고 했을 때, 각각 호출할 때 생성된 변수들은 각각의 독립성을 가지고 있습니다. 1차 함수 호출 시 사용된 매개변수가 두 번째 호출될 때 영향을 주지 않습니다.

종종 우리는 코드에서 외부의 변수와 함수 안에 쓰이는 변수 이름이 같은 경우를 보는 경우가 있습니다. 이는 코드를 시각화로 편하게 하기 위해서 비슷한 이름을 사용하는 것이지 엄밀히 말하면 모두 각각 이름만 같은 다른 변수입니다.

내부변수와 매개변수가 다르다는 것은 함수의 호출에서 사용되는 변수와 함수 안에서 사용되는 변수가 서로 독립된 다른 변수라는 것입니다. 이 개념은 약간 혼동이 될 수도 있는데 이를 정확히 표현하는 명칭으로 함수를 호출할 때 이용하는 변수 이름을 ‘실행 인자’라고 부르고, 함수 안에서 받은 변수의 이름을 ‘매개변수’라고 표현을 구분하면 이해하기 더 쉬울 것입니다.

매개변수: 함수에서 인자값으로 받아 들이는 변수 값

실행 인자: 함수를 호출할 때 전달하는 값

**예제 파일** func-05.php

|  |
| --- |
| <?php  $name = “jiny”;  function hellowWorld ($name, $age.) {  echo $name .”는 $age 입니다. “<br>”;  }  function setUserName ($name) {  echo $name . “<br>”;  }  ?> |

위의 예를 보면 함수 외부에 선언되어 있는 $name 변수와 두 개의 함수의 매개변수로 선언된 $name 변수는 모두 각각의 다른 변수입니다.

함수 외부에 선언된 $name 변수는 hellowWorld 함수와 setUserName 함수에 있는 $name 변수는 다른 메모리 영역입니다. 또한 두 개의 함수에서도 외부에 선언된 $name 변수로 접근 할 수 없습니다.

또한 hellowWorld 함수의 $name 변수를 setUserName 함수에서 참조하거나 사용할 수도 없습니다. 즉 서로 다르게 분리된 변수라고 보면 됩니다.

이런 변수를 접근할 수 있는 권한을 각각이 분리하는 것은 절차적 프로그램과 함수 프로그램 간의 변수의 중복 및 처리에 대한 오류를 방지하기 위함입니다.

**10.4.2 상수 실행 인자**

함수를 호출할 때 전달되는 실행 인자값을 변수 형태 이외에 직접 값을 입력하는 상수 형태로도 전달할 수 있습니다.

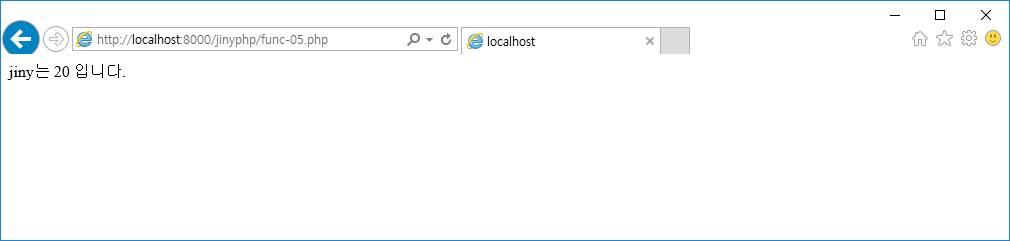
실행 인자를 변수 형태가 아니라 상수값 형태로 전달하는 방식은 함수의 값을 테스트하거나, 직관적으로 데이터 값을 전달할 때 편리합니다. 또한 고정된 기능을 처리하는 함수를 만들 때 편리합니다.

**예제 파일** func-06.php

|  |
| --- |
| <?php  function hellowWorld ($name, $age) {  echo $name ."는 $age입니다. <br>";  }  // 문자열 상수 실행 인자와, 정수형 상수 실행 인자를 전달합니다.  hellowWorld(**"jiny"**, **20**);  ?> |

**결과**

jiny는 20입니다.



위의 예제를 보면 hellowWorld 함수를 호출할 때 입력받는 함수의 매개변수 $name과 $age 값을 직접 문자열 및 정수로 데이터를 전달 합니다.

**10.4.3 변수로 실행 인자 전달**

상수값을 실행 인자로 전달하는 경우는 테스트 등의 작업 이외에는 별로 없습니다. 대부분 소스들은 데이터를 처리하고 값을 가공합니다. 또한 수많은 변수들을 통해 처리된 값들을 저장하고 실행 인자값으로 사용합니다. 변수로 실행 인자값을 전달한다는 표현은 함수 실행 시 호출하면서 상수값 형태로 전달하는 것과 달리 대부분은 변수에 데이터 값을 넣어서 값을 전달합니다.

**예제 파일** func-07.php

|  |
| --- |
| <?php  function addSum($x, $y) {  $sum = $s + $y;  echo $sum."<br>";  }  // 변수에 값을 설정합니다.  $x = 10;  $y = 5;  // 함수를 호출합니다.  addSum($x, $y);  $value1 = 50;  $value2 = 25;  // 함수를 호출합니다.  addSum($value1, $value2);  ?> |

**결과**

5

25

위의 예제를 보면 addSum 함수는 두 개의 정수 값을 입력받아 합계를 구합니다. 함수 외부에 선언된 $x와 $y에 대입 연산자를 통해 값을 설정합니다.

함수 외부의 $x, $y와 함수 매개변수 $x, $y는 다른 변수입니다. 함수를 호출할 때 인자값을 변수로 대입하여 데이터를 전송합니다.

변수의 값은 같을 수도 있지만 정확한 의미로는,

addsum 함수의 $x 변수 = 함수 외부의 $x

값을 대입하여 전송하는 것과 같은 의미입니다. 즉, 함수 호출 시 입력하는 매개변수 값은 **함수 내부의 변수에게 대입 연산**을 하는 것과 같습니다.

**10.4.4 매개변수 초기값**

함수의 매개변수를 선언했을 경우 매개변수는 반드시 값을 지정하여 함수를 호출하는 것을 기본으로 합니다. 또한 PHP는 특성상 매개변수에 실행 인자값을 생략하여 호출할 수도 있습니다. 생략하거나 미지정된 매개변수 값을 함수 내에서 호출할 때 데이터 처리 오류가 발생할 수 있습니다.

이러한 점들을 미리 방지하기 위해서 PHP는 함수 매개변수를 **선언할 때 초기값**을 같이 선언을 할 수 있습니다.

만일 이러한 매개변수 초기값을 설정하는 기능이 없다면 함수 초기에 매개변수의 유효성을 체크하는 기능을 매번 만들어야 될 것입니다. 초기값 설정 기능을 통해 함수의 오동작을 줄여주고 함수의 매개변수 초기화 기능을 생략함으로써 소스의 코드를 줄일 수 있는 기능이 됩니다.

**|문법|**

|  |
| --- |
| function 함수명 (인자1**=”초기값”**) {  함수 정의 몸체 ...  소스 코드 내용  } |

PHP의 매개변수의 초기값은 매우 직관적입니다. 매개변수를 선언할 때 변수명 = "기본값" 형태로 지정하면 됩니다.

**예제 파일** func-08.php

|  |
| --- |
| <?php  function setHeight($minheight = 50) {  echo "The height is : $minheight <br>";  }  // 함수 호출 시 매개변수 값 250을 입력합니다.  setHeight(250);  // 매개변수 값이 없습니다.  // 기본값 50이 적용됩니다.  setHeight();  ?> |

**결과**

The height is : 250

The height is : 50

위의 예제를 보면 높이를 설정하는 사용자 함수가 있습니다. 본 함수는 매개변수를 선언할 때 = 50으로 초기값이 설정되어 있습니다.

함수를 호출할 때 매개변수 값을 입력하면 입력된 매개변수 값이 전달되어 화면에 출력됩니다. 하지만 매개변수 값을 입력하지 않고 호출하면 기본값으로 설정된 값으로 화면에 출력됩니다.

보통 함수를 선언하고 실행 호출을 할 때 매개변수의 개수와 형식을 유지하여 작성해야 합니다. 하지만 PHP는 함수에서 선언한 매개변수를 생략하여 호출할 수 있는 특징이 있습니다. 이렇게 생략 처리된 경우에는 초기값으로 전달됩니다.

**10.4.4 타입 힌트**

PHP는 변수의 타입을 특별히 지정하지 않아도 쓸 수 있는 유연한 언어입니다. 하지만 함수의 매개변수의 데이터 값을 입력받을 때 오류를 방지하고 정확한 값을 받고 싶을 경우에는 변수의 타입 힌트 기능을 통해 값의 유형을 지정할 수 있습니다. 변수명 앞에 입력받은 매개변수의 타입 힌트를 같이 정의함으로써 잘못된 값의 매개변수 입력을 사전에 필터링할 수 있습니다.

변수의 타입 힌트로 사용할 수 있는 형식은 다음과 같습니다:

* Bool : 논리값
* Float : 실수형 자료
* Int : 정수형 자료
* String : 문자열 자료

**예제 파일** func-09.php

|  |
| --- |
| <?php  function test(**int** $x){  echo $x."<br>";  }  echo test(1);  function test2(**string** $x){  echo $x."<br>";  }  echo test2("문자열");  ?> |

**결과**

1

문자열

위의 예제에서 첫 번째 test(1); 은 실행 인자로 정수 값을 전달하고 함수도 타입 힌트를 통하여 정수 값을 받았기 때문에 오류 없이 입력한 정수 값을 출력합니다.

하지만 만일 타입이 틀린 형태 test("a");로 함수를 호출하는 경우에는 실행 인자가 문자열이고 입력 받는 매개변수의 타입이 정수이기 때문에 데이터 타입이 일치하지 않습니다. 이런 경우에는 PHP에서는 오류 메시지를 출력하게 됩니다..

C:\php-7.1.4-Win32-VC14-x86\jinyphp\func-22.php:2

Stack trace:

#0 C:\php-7.1.4-Win32-VC14-x86\jinyphp\func-22.php(8): test('a')

#1 {main}

thrown in C:\php-7.1.4-Win32-VC14-x86\jinyphp\func-22.php on line 2

**10.4.4 참조 매개변수**

C 언어와 같은 low-Level 언어에는 매개변수로 인수의 참조(call by reference) 형태로 매개변수를 전달할 수 있습니다. 이를 또 다른 말로 참조에 의한 전달(passing by reference)이라고도 합니다. PHP에서도 이와 같은 형태의 실행 인자 전달과 참조 매개변수를 선언하여 사용할 수 있습니다. C 언어에서 포인터와 같은 개념이라고 생각하면 됩니다.

참조 매개변수는 변수명 앞에 & 기호를 붙여줍니다.

**예제 파일** func-10.php

|  |
| --- |
| <?php  function drink(**&**$name){  $name = "사이다";  }  $drinkName = "콜라";  echo "음료수 이름 = $drinkName <br>";  drink($drinkName);  echo "음료수 이름 = $drinkName <br>";  ?> |

**결과**

음료수 이름 = 콜라

음료수 이름 = 사이다

위의 예를 보면 $drinkName 변수에는 “콜라” 값이 저장되어 있습니다. 이때 drink() 함수를 호출할 때 drink() 함수의 매개변수 인자는 입력받는 변수의 참조 주소 값으로 받습니다. 매개변수를 참조변수 형태로 받게 되면 drink() 함수의 $name 변수는 함수 외부에 있는 $drinkName 변수를 가리키게 됩니다.

즉, drink() 함수는 실행 인자로 전달된 변수의 주소를 참조하여 전달한 변수값 자체를 변경하게 됩니다. 그래서 두 번째 출력에서는 “사이다”로 변경되어 출력됩니다.

**10.4.5 가변 매개변수**

함수를 선언할 때 다수의 매개변수 인자값 선언을 했을 경우에 함수 호출도 동일한 수량의 인자값을 포함하여 함수를 호출해야 합니다. 모든 매개변수와 함수의 실행 인자의 개수는 한 쌍으로 맞추어 전달되게 됩니다.

함수를 선언할 때 매개변수의 개수는 선언 시 정의해야 합니다. 하지만 함수의 사용을 좀 더 유연적으로 처리하기 위해서 가변적으로 매개변수의 개수를 조절해야 하는 경우가 있습니다. PHP언어는 가변 매개변수로 함수 정의와 호출 시 한 쌍씩 호출하는 규칙을 유연하게 처리를 할 수 있습니다. 즉, 함수의 정의된 매개변수의 개수보다 호출하는 인자의 수가 적어도 오류를 발생하기 않습니다. 매칭되지 않는 변수들은 NULL 또는 기본값으로 처리됩니다.

이러한 가변 매개변수를 이용하는 것은 좀 더 유연한 함수를 생성하고 처리하는 데 매우 유용합니다.

심지어 매개변수를 하나도 지정하지 않아도 함수에 매개변수를 전달할 수 있습니다. 이런 경우 함수를 실행할 때 매개변수의 개수를 확인할 수 있는 func\_num\_args() 내부 함수와 매개변수의 데이터를 배열로 가지고 올 수 있는 func\_get\_args() 내부 함수 두 개를 사용하여 처리할 수 있습니다.

**예제 파일** func-11.php

|  |
| --- |
| <?php  function setting(){  $num = func\_num\_args();  echo "실행 인자 개수 = $num <br>";  $args = func\_get\_args();  foreach($args as $arg) echo $arg."<br>";    }  setting("aaa");  echo "====<br>";  setting("aaa","bbb");  echo "====<br>";    setting("aaa","bbb","ccc");  echo "====<br>";  ?> |

**결과**

실행 인자 개수 = 1

aaa

====

실행 인자 개수 = 2

aaa

bbb

====

실행 인자 개수 = 3

aaa

bbb

ccc

====

**예제 파일 :func-12**.php

|  |
| --- |
| <?php  function sum() {  $a = 0;  // func\_get\_args 함수를 통해 매개변수를 읽어옵니다.  foreach (func\_get\_args() as $n) {  $a += $n;  }  return $a;  }  echo sum(1, 2, 3, 4, 5);  ?> |

**결과**

15

**10.4.6 가변 인자**

가변 매개변수에서 func\_get\_args() 함수를 통해 입력되는 변수들을 배열로 입력 처리한 예를 보았을 것입니다. PHP 5.6 이상의 버전에서는 … 와 같은 인수의 변수 번호 기능을 지원합니다.

함수가 가변 인자를 받을 목적으로 … 키워드를 포함할 수 있습니다. … 키워드는 배열은 아니나 인수를 배열 형태로 전달됩니다.

… 키워드 가변인자는 함수의 인자값의 개수를 예측하기 어려울 때 매우 유용합니다.

**예제 파일** scalar-13.php

|  |
| --- |
| <?php  // 가변 인자로 받습니다.  function sum(**...**$numbers) {  $acc = 0;  foreach ($numbers as $n) {  $acc += $n;  }  return $acc;  }  echo sum(1, 2, 3, 4, 5);  ?> |

**결과**

15

위의 예에서 sum()함수는 가변 인자값으로 매개변수를 전달받습니다. 하지만 개수는 정의되지 않는다는 것입니다.

sum(1, 2, 3, 4, 5);처럼 여러 개의 값을 콤마(,)로 구분하여 다수의 인자값을 전달할 수 있습니다. 또한 전달된 인자값들은 가변변수 $numbers 로 받아 배열로 구분하여 각각의 값을 처리합니다.

**예제 파일** scalar-14.php

|  |
| --- |
| <?php  function add($a, $b) {  return $a + $b;  }  echo add(**...**[1, 2])."<br>";  $a = [4, 5];  echo add(**...**$a);  ?> |

**결과**

3

9

위의 add() 함수는 두 개의 값을 더하는 기능을 합니다. 함수 호출 시 배열 형태의 값으로 가변변수를 호출할 수도 있습니다. 또한 배열 변수를 가변변수의 인자값으로 전달할 수도 있습니다.

**10.5 함수의 반환값**

함수는 특정처리를 하는 블랙박스와 같다고 앞에서 설명을 했습니다. 입력값이 있으면 그에 상응하는 결과값이 있을 수도 있습니다. 함수는 입력받은 매개변수처럼 함수가 특정한 작업 처리한 후에 결과 값이 있을 경우에 반환할 수 있습니다.

실제적으로 함수를 작성하는 것은 어떠한 처리를 동작시키는 것뿐만 아니라 처리 결과를 얻기 위해서 많이 사용합니다. 또한 동작만 하는 경우에도 동작의 처리 결과의 성공 여부를 알려주기 위해서도 논리 반환값으로 많이 사용합니다.

**10.5.1 return 키워드**

함수의 데이터 처리 후에 값을 반환하기 위해서는 return이라는 키워드를 사용합니다. return의 사전적인 의미를 보면 ‘돌아오다’, ‘돌아가다’라는 의미를 가지고 있습니다. 함수가 실행 도중에 return 키워드를 만나게 되면 실행하는 함수의 처리를 중단하고 함수를 호출한 위치로 제어권을 돌려줍니다.

|  |
| --- |
| **문법**return **반환값**; |

return 키워드 다음에 반환할 값 또는 변수를 적으면 됩니다. 이때 return문 다음에 값을 같이 정의하여 함수의 처리 결과를 호출한 제어권의 위치에 값을 전달합니다.

반환으로 사용할 수 있는 변수 타입의 제한은 없습니다. 정수, 문자열, 어레이, 객체 등 다양한 형태의 변수를 반환할 수 있습니다. 심지어 상수나 논리값을 직접 쓰거나, 함수에서 처리한 변수를 적을 수도 있습니다.

호출한 함수의 반환값을 받기 위해서는 대입 연산자를 이용하면 됩니다. 즉, 함수를 호출하게 되면 호출된 위치에서 함수의 제어권이 변경되고 함수를 처리하게 됩니다. 또한 return 반환문을 받으면 함수를 호출한 코드의 라인과 그 위치에 값이 반환되기 때문에 반환값을 변수에 대입하게 됩니다. 따라서 반환값을 얻기 위해서는 함수의 호출과 함께 데이터를 받을 변수에 대입하는 형태로 작성해야 합니다.

**문법**

|  |
| --- |
| $변수 **=** 함수명(); |

즉, 함수를 실행 후 반환되는 값을 대입 연산자를 통하여 변수에 반환값을 저장하게 됩니다.

만일 반환값을 변수에 대입하지 않고 함수의 반환값을 처리할 수도 있습니다. 함수 처리의 성공 여부를 나타내는 논리값 같은 경우가 대표적입니다.

예제)

|  |
| --- |
| if(함수명()){  } else {  } |

위의 문법 예처럼 if 조건문 안에 바로 함수를 넣어서 작성할 수도 있습니다. 만일 반환값이 1보다 큰 의미 있는 값일 경우에는 참 조건을 실행하고 반환값이 0이거나 NULL 일 경우에는 거짓 부분을 실행합니다.

모든 함수는 반환 반환 키워드 return을 만나면 데이터 값을 반환함과 동시에 함수 종료의 의미를 가지고 있습니다. 반환값을 가지고 실행 제어권이 함수 호출 위치로 돌아가게 됩니다.

**예제 파일** func-15.php

|  |
| --- |
| <?php  function sum($x, $y) {  $z = $x + $y;  **return $z;**  }  echo "5 + 10 = " . sum(5, 10) . "<br>";  echo "7 + 13 = " . sum(7, 13) . "<br>";  echo "2 + 4 = " . sum(2, 4);  ?> |

**결과**

5 + 10 = 15

7 + 13 = 20

2 + 4 = 6

다음 예제는 함수의 인자값을 전달하여 비교 판단 후 결과를 반환합니다. sum() 함수는 두 개의 인자값을 매개변수로 받아서 값을 합산한 후에 결과값을 돌려줍니다. 예제의 echo 명령에서처럼 함수의 반환값을 변수나 조건문 등을 사용하지 않고도 바로 화면으로 출력할 수 있습니다.

**예제 파일** func-16.php

|  |
| --- |
| <?php  // 성인 여부를 판별하는 함수를 하나 생성합니다.  // 인수로는 나이 값 변수 하나를 전달합니다.  function is\_adult($age){  if($age>=18){  **return true;**  } else {  **return false;**  }  }  $memAge = 20;  if (is\_adult($memAge)) {  echo "성인입니다.";  } else {  echo "미성년입니다.";  }  ?> |

**결과**

성인입니다.

**10.5.2 복수 데이터 반환**

함수에서 return 키워드는 여러 번 사용할 수는 있으나, return 키워드를 이용하여 반환할 수 있는 값은 한 개의 데이터만 반환할 수 있습니다. 다음처럼은 사용할 수가 없습니다.

return $aaa, $bbb;

만일 여러 개의 데이터를 반환하기 위해서는 객체 또는 배열을 통해 반환하면 복수의 데이터를 반환 받아 처리할 수 있습니다. 이것은 반환하는 변수는 한 개이나 변수의 특성상 배열과 객체는 여러 개의 데이터를 포함하고 있기 때문입니다.

**10.5.3 반환 데이터 타입 선언**

PHP언어의 장점이라면 C 언어처럼 함수를 반환할 때 데이터 타입을 지정하지 않아도 됩니다. PHP에서는 함수에서 return으로 반환하는 변수의 타입으로 데이터가 반환됩니다. 즉, 변수의 값이 정수이면 정수로 반환되고, 논리값이면 논리변수 타입으로 반환하게 됩니다.

이러한 PHP의 유연한 변수 타입 설정으로 인해 함수의 반환값의 타입을 엄격하게 정하여 사용하지는 않았습니다. 하지만 PHP 7.x로 버전이 올라가면서 PHP도 함수의 반환 데이터의 타입을 지정할 수 있게 되었습니다. 만일 7.x 이상의 버전을 사용하고 좀 더 반환되는 타입을 엄격하게 지정하고 싶다면 다음과 같이 반환 타입을 함수 선언 시에 정의할 수 있습니다.

**문법**

|  |
| --- |
| function 함수명(인자1, 인자2, ... ) **: 반환 데이터 타입** {  함수내용...  } |

함수 선언 시에 함수 인자값 괄호 ( ) 뒤에 콜론과 반환 데이터 타입을 선언하면 됩니다. 함수로부터 반환되는 데이터 타입은 기존 PHP에서 사용 가능한 데이터 타입 리스트를 따릅니다.

**예제 파일** func-17.php

|  |
| --- |
| <?php  function arraysSum(array ...$arrays): array {  return array\_map(  function(array $array): int {  return array\_sum($array);  }  ,  $arrays  );  }  print\_r(  arraysSum(  [1,2,3],  [4,5,6],  [7,8,9]  )  );  ?> |

**결과**

Array ( [0] => 6 [1] => 15 [2] => 24 )

위의 예제는 2차원 배열의 인자값을 받아서 각각의 서버 1차 배열 간의 합계를 구한 후에 1차원 배열값을 반환합니다. 이때, 함수 선언부에 반환되는 데이터 값이 배열(array)로 설정되어 있습니다.

반환하는 데이터와 반환 데이터 타입이 일치하지 않은 경우에는 PHP에서 오류를 발생합니다. 반환받은 1차원 배열을 print\_r() 함수로 바로 출력하여 확인을 합니다.

**10.6 가변 함수**

가변 함수란 PHP에서 함수를 호출할 때 동적으로 함수를 호출할 수 있는 기능입니다.

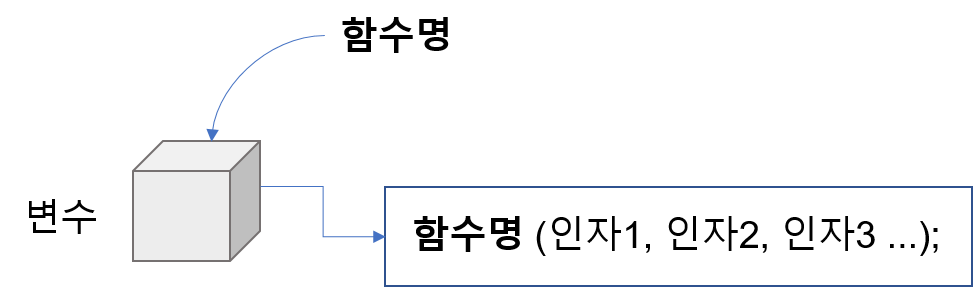
**10.6.1 문법**

지금까지 함수 선언 function 키워드를 통해 정의한 함수 이름을 사용하여 함수를 호출했습니다. 소스상에서 선언한 함수명을 기준으로 직접 함수명(); 형태로 적어서 함수를 호출하여 사용했습니다.

|문법|

|  |
| --- |
| $변수명 = “함수 이름”;  $변수명(); |

가변 함수는 변수명에 함수명을 대입하고 $변수명(); 형태로 함수를 호출할 수 있습니다.



함수명을 임시변수에 문자열로 저장한 후에 저장된 문자열과 일치하는 함수를 호출하는 것입니다.

**10.6.2 가변 함수 호출**

가변 함수 호출은 동적으로 함수를 처리하여 호출할 때 매우 유용합니다. 가변 함수를 이용하면 프로그램 로직 또는 입력값을 분석하여 동적으로 함수를 실행할 수 있는 장점 등이 있습니다.

|  |
| --- |
| if($func=="aaa"){  aaa();  } else if($func=="bbb"){  bbb();  } else if($func=="ccc"){  ccc();  } |

위와 같이 조건에 따라서 각각 다르게 함수 호출을 처리해야 하는 경우에는 가변 함수 호출을 통해 쉽게 작성할 수 있습니다.

**기존 함수 호출 방식**

|  |
| --- |
| <?php  function hello() {  echo "hello() 함수입니다.<br />\n";  }  hello();  ?> |

**가변 함수 호출 방식**

**예제 파일** func-18.php

|  |
| --- |
| <?php  function hello() {  echo "hello() 함수입니다.<br />\n";  }    $func = "hello";  $func();  ?> |

**결과**

hello() 함수입니다.

위의 예제를 보면 $func 변수에는 함수명을 저장합니다. 저장된 변수의 함수명을 이용하여 함수를 호출합니다. 이처럼 변수를 이용하여 가변적으로 함수를 호출할 수 있는 방법이 가변 함수입니다.

**10.6.3 클래스 가변 메서드**

가변 함수 문법은 향후 설명할 클래스에서도 메서드를 호출할 때 비슷한 방법으로 호출할 수 있습니다. 여기서는 간단히 학습하겠습니다.

**클래스 & 가변 함수**

**예제 파일** func-19.php

|  |
| --- |
| <?php  class jiny  {  function Variable()  {  $name = 'Bar';  $this->$name(); // Bar() 메서드를 호출합니다.  }    function Bar()  {  echo "This is Bar";  }  }  $jiny = new jiny();  $funcname = "Variable";  $jiny->**$funcname()**; // $jiny->Variable()를 호출합니다.  ?> |

**결과**

This is Bar

위의 예제처럼 클래스를 선언하고, 인스턴스를 생성한 후에 클래스에서 호출하는 메서드도 변수를 통해 가변 메서드 형태로 호출할 수도 있습니다.

**10.7 익명 함수**

지금까지 우리는 함수를 정의하고 호출을 하는 기준은 함수의 이름이었습니다. 하지만 톡특하게도 함수의 이름 없이 사용하는 경우가 있습니다. 익명 함수는 또 다른 말로 람다 함수라고도 표현합니다. 익명 함수는 말 그대로 함수의 이름이 없는 형태의 함수입니다.

익명 함수는 함수의 매개변수, 함수의 반환값 또는 변수와 바인딩(결합)하여 사용 합니다. 또한 익명 함수는 콜백 함수에 많이 사용을 합니다.

PHP 5.3 이전에는 콜백 함수를 정의하는 명확한 방법이 없었습니다. create\_function()과 같은 함수를 응용하여 처리를 해야 하는 프로그램은 가독성이 떨어지고 유지보수가 어려운 점이 있었습니다.

**10.7.1 익명 함수 문법**

함수는 항상 선언과 호출을 같이 쌍으로 이뤄진 형태의 문법입니다. 하지만 익명 함수는 기존 함수의 정의 후 호출하는 것과 달리 필요할 때마다 즉시 생성한 후에 호출하여 사용할 수 있습니다.

**문법**

|  |
| --- |
| function (인자1, 인자2, 인자3 ...) {  함수 정의 몸체 ...  소스 코드 내용  } |

위의 익명 함수 예제를 보면 함수 선언 키워드 function과 매개변수 인자 () 선언 사이에 있는 함수명이 생략되어 있습니다. 이러한 코딩 표현은 소스상에서 선언과 바로 실행 호출됩니다.

이러한 특성의 익명 함수는 한 번만 사용하는 일회성 함수를 모두 정의해서 프로그램에 상주하지 않고 필요할 때마다 간략하게 함수를 사용할 수 있는 특징이 있습니다.

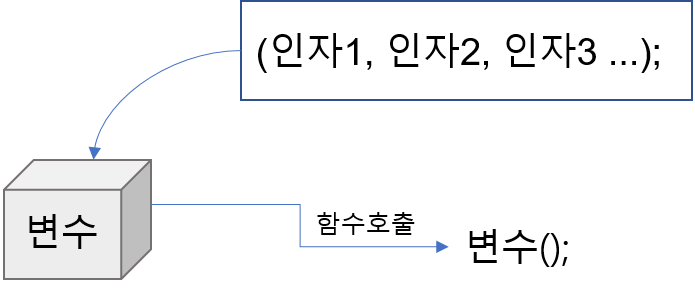
**10.7.2 변수 바인딩**

익명 함수는 변수에 바인딩(결합)하여 사용할 수 있습니다. 약간 가변 함수와 유사하다고 생각할 수도 있습니다. 가변 함수는 변수에 선언한 함수를 바인딩하는 것이었고, 익명함수 변수 바인딩은 함수 자체를 바인딩하는 것입니다.

**변수 바인딩**

|  |
| --- |
| $변수명 = function (인자){  } |

위와 같은 형태의 문법은 익명 함수를 변수에 결합하게 됩니다. 변수에 결합된 익명 함수를 가변 함수로 호출하는 것처럼 함수를 호출하여 사용할 수 있습니다.



**익명 함수 호출**

|  |
| --- |
| $변수명(인자); |

익명 함수를 변수에 바인딩 결합하여 호출하는 문법 표현은 가변 함수 호출하는 방식과 매우 유사합니다. 가변 함수는 선언한 함수명을 변수에 대입하여 호출하는 것이고, 익명 변수는 함수 선언으로 바인딩 결합합니다.

**예제 파일** func-20.php

|  |
| --- |
| <?php  $meeting **= function($name)**  {  printf("Hello %s\r\n", $name);  };  $meeting('jiny');  ?> |

**결과**

Hello jiny

위의 예제는 익명 함수를 통하여 문자열의 수를 출력합니다. $meeting 변수에 문자열을 출력하는 익명 함수를 바인딩 결합합니다. 또한 매개변수 인자로는 하나의 이름을 받습니다. 결합된 변수를 통하여 함수를 호출합니다.

**10.7.3 콜백 함수**

일반적으로 함수는 호출 시 매개변수 입력과 함수 내부의 처리 후 결과값을 return으로 반환하는 구조로 되어 있습니다. 또한 반환 키워드 return으로 함수가 반환되면 실행 제어권은 동작하는 현재의 함수를 탈출하여 호출한 실행 지점으로 돌아가게 됩니다.

하지만 return을 콜백 함수 형태로 정의하면 함수를 호출한 실행 지점으로 반환되는 것이 아니라 재귀 함수처럼 콜백 함수로 한 번 더 새로운 함수로 진입하는 것과 비슷합니다. 즉 한 번 더 함수를 실행하는 것과 같습니다.

익명 함수는 콜백 함수로 많이 이용하는 기술입니다.

**예제 파일** README.txt

|  |
| --- |
| 안녕하세요. jinyPHP입니다. |

아래 예제는 위의 예제 txt 파일을 읽어서 화면에 출력하는 예제입니다. 옛날 고전적인 create\_function() 함수를 사용하는 방법과 최신 익명 함수를 사용하는 두 가지 예제입니다. **콜백 호출**

**예제 파일)** func-21.php

옛날 create\_function() 함수를 통한 콜백 처리

|  |
| --- |
| <?php  /\* 유닉스 형식 명령줄 필터로  \* 문단의 첫 대문자를 소문자로 바꿉니다. \*/  $fp = fopen("README.txt", "r") or die("stdin을 읽을 수 없습니다");  while (!feof($fp)) {  $line = fgets($fp);  $line = preg\_replace\_callback(  '|<p>\s\*\w|',  **create\_function**(  // 여기에서 홑따옴표가 중요합니다.  // 혹은 모든 $를 \$로 회피해야 합니다.  '$matches',  'return strtolower($matches[0]);'  ),  $line  );    echo $line;  }  fclose($fp);  ?> |

**결과**

안녕하세요. jinyPHP입니다.

익명 함수를 통한 콜백 처리

**예제 파일** func-22.php

|  |
| --- |
| <?php  /\* 유닉스 형식 명령줄 필터로  \* 문단의 첫 대문자를 소문자로 바꿉니다. \*/  $fp = fopen("README.txt", "r") or die("can't read stdin");  while (!feof($fp)) {  $line = fgets($fp);  $line = preg\_replace\_callback(  '|<p>\s\*\w|',    **function ($matches)** {  return strtolower($matches[0]);  },  $line  );  echo $line;  }  fclose($fp);  ?> |

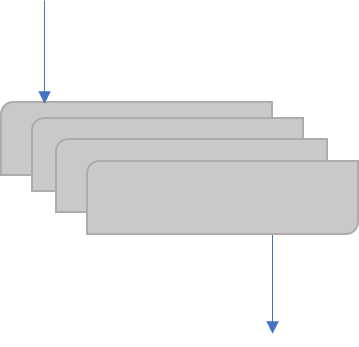
**결과**

안녕하세요. jinyPHP입니다.

위의 예제는 콜백을 이용한 정규표현식 검색과 치환을 수행하는 함수의 콜백 처리로 익명 함수를 사용한 예제입니다.

**10.8 재귀 호출**

재귀 호출이란 함수가 자기 자신을 호출하는 것을 말합니다. 또는 ‘함수 되부름’이라고도 합니다. 재귀 함수는 자기 자신을 반복 수행하기 때문에 세밀하게 주의하여 사용해야 합니다. 잘못 사용하면 프로그램의 논리적인 오동작이 발생할 수도 있습니다.



재귀 함수를 쓸 때는 매번 전달하는 호출 아규먼트 인자가 바뀝니다. 이는 자기 자신을 재호출하기 때문입니다. 값을 바꿔가면서 단계적으로 계속 자기 자신 호출을 실행합니다. 재귀 함수를 사용할 때는 자기 자신의 반복의 횟수를 정하는 것이 좋습니다.

또는 재귀 호출 시 탈출할 수 있는 조건을 넣어두는 것이 현명합니다. 자기 자신의 재귀 호출 반복 횟수를 정의해야 재귀 호출 처리를 한 후에 함수를 탈출할 수 있습니다.

**예제 파일** func-23.php

|  |
| --- |
| <?php  function recursion($a){  if ($a < 5) {  echo "입력값 = $a <br>";  // 자기 자신을 호출합니다.  **recursion($a + 1);**  echo "함수 탈출($a) <br>";  }  }  $a = 1;  recursion($a);    echo "<br>";  function recursion2($a) {  if ($a < 5) {  echo "이전: ".$a."<br>";  **recursion2($a + 1);**  echo "이후: ".$a."<br>";  }  }  recursion2(0);  ?> |

**결과**

입력값 = 1

입력값 = 2

입력값 = 3

입력값 = 4

함수 탈출(4)

함수 탈출(3)

함수 탈출(2)

함수 탈출(1)

이전: 0

이전: 1

이전: 2

이전: 3

이전: 4

이후: 4

이후: 3

이후: 2

이후: 1

이후: 0

함수는 한번 호출될 때마다 호출 값에 대한 메모리를 할당하게 됩니다. 무한적인 재귀 호출은 메모리 오버플로 등을 만들어내서 프로그램이 멈출 수 있습니다.

위의 예제에서 함수를 재귀 호출할 때 인자값으로 $a를 넘겨 줍니다. 함수는 $a < 5일 경우에만 재귀 호출을 합니다. 이런 형태로 조건 및 재귀 호출 탈출 조건을 생성합니다. 항상 재귀 호출을 기능을 구현할 때는 이러한 점들을 주의해야 합니다.

**10.9 제너레이터**

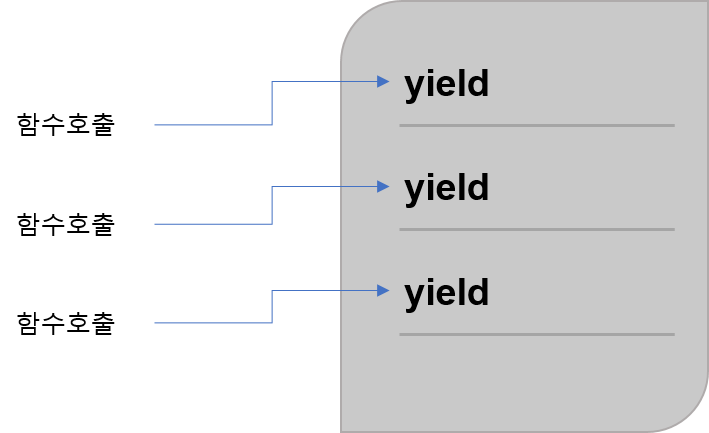
모든 함수의 실행은 한 번의 호출과 한 번의 함수 처리를 하게 됩니다. 하지만 특이하게도 제너레이터는 yield 키워드를 사용한 PHP 함수로 분할하여 실행할 수 있습니다. 특수한 형태의 함수입니다.

또한 제너레이터는 특성상 반환값을 갖지 안습니다. 제너레이터는 PHP 5.5 업그레이드되면서 적용된 기능입니다.

**10.9.1 yield**

제너레이터 함수의 핵심은 yield 키워드입니다.

yield 키워드는 return과 비슷한 것 같지만 다릅니다. 프로그램의 return문은 함수를 끝내는 역할을 하지만 yield는 함수를 잠시 멈추는 역할을 합니다. 즉, 함수는 잠시 멈춘 상태를 저장해 놓은 상태에서 함수를 재호출할 때 동작을 이어서 처리합니다.



제너레이터 함수에서 yield키워드를 사용하는 개수의 제한은 없습니다. 하나의 제너레이터 함수에 한 개 또는 여러 개의 yield 함수를 넣어서 잠시 멈출 수 있는 횟수를 지정할 수 있습니다. 제너레이터 함수를 호출하면 첫 번째 yield 명령이 나올 때까지 실행을 하고 잠시 멈춥니다.

또 다시 제너레이터 함수를 실행을 하면 이전에 멈춘 위치부터 다음 위치의 yield 명령문까지 실행합니다.

이런 형태로 멈춤 기능을 통해 함수를 분할하여 실행을 할 수 있습니다. 제너레이터는 함수의 마지막 끝까지 또는 return;문이 나올 때까지 반복합니다.

**예제 파일** generator-01.php

|  |
| --- |
| <?php  // yield 키워드는 값을 반환한다는 측면에서 return 키워드와 비슷하면서도 다르다.  function myGen(){  // 호출 시 첫 번째 yield 키워드까지 실행한다.  echo "첫 번째 호출=";  **yield "test1";**  // 호출 시 두 번째 yield 키워드까지 실행한다.  echo "두 번째호출=";  **yield "test2";**  // 호출 시 세 번째 yield 키워드까지 실행한다.  echo "세 번째 호출=";  **yield "test3";**    }  foreach (myGen() as $value) {  echo $value."<br>";  }  ?> |

**결과**

첫 번째 호출=test1

두 번째 호출=test2

세 번째 호출=test3

위의 예를 보면 제너레이터는 반환값이 없는 대신에 yield 뒤에 값을 입력하면 foreach문으로 특정 데이터를 가지고 올 수 있습니다.

**10.9.2 제너레이터 델리게이션**

제너레이터 함수 실행 시 **from** 키워드를 추가하여 또 다른 제너레이터 함수로 연결 실행할 수 있습니다. 즉, 제너레이터 함수 안에 또 다른 제너레이터 함수를 추가하여 실행할 수 있습니다.

**예제 파일** generator-02.php

|  |
| --- |
| <?php  function gen(){  yield "gen() 1 실행";  yield "gen() 2 실행";  // 3번째 yield는 gen2() 제너레이터로 연결  yield **from** gen2();  }  function gen2(){  yield "gen2() 3 실행";  yield "gen2() 4 실행";  }  foreach (gen() as $val){  echo $val."<br>";  }  ?> |

**결과**

gen() 1 실행

gen() 2 실행

gen2() 3 실행

gen2() 4 실행

위의 예제는 두 개의 제너레이터 함수를 연결하여 사용하는 예제입니다. gen() 함수에 yield 키워드 두 개와 gen2() 함수에 yield 키워드 두 개를 연결하여 총 네 개의 yield 키워드가 삽입되어 있습니다.

gen() 함수는 총 네 번으로 나누어 실행하게 됩니다.

**10.9.2 제너레이터 예**

**예제 파일** generator-03.php

|  |
| --- |
| <?php  function rowsCount($count){  for ($i=0;$i<$count;$i++){  // 반복문 안에도 yield를 넣어서 반복할 수 있습니다.  yield $i;  }  }  foreach (rowsCount(10) as $i){  echo " 실행 =". $i . "<br>";  }  ?> |

**결과**

실행 =0

실행 =1

실행 =2

실행 =3

실행 =4

실행 =5

실행 =6

실행 =7

실행 =8

실행 =9

위의 예는 제너레이터 함수 안에 반복문을 통해 yield 키워드를 동적으로 삽입합니다. 또한 제너레이터 함수를 호출할 때 매개변수로 반복 횟수를 전달합니다. 즉, 제너레이터 함수는 반복 횟수만큼 함수를 분할하여 실행하게 됩니다.