جزوه درس برنامه نویسی مبتنی بر وب

مقطع کاردانی

دانشگاه فنی و حرفه ای دختران اهواز

استاد: زارعی

Marziyeh.zareie@gmail.com

www.m-zareie.ir

برنامه نویسی سمت سرویس دهنده

معرفی PHP:

تاریخچه مختصر زبان PHP:

- این زبان در سال ۱۹۹۵ میلادی توسط راسموس لردورف (Rasmus Lerdorf) ساخته‌شد.
- در ۱۹۹۷ اندی گاتسمن و زیو سوراسکی نسخه ۳ منتشر کردند.
- در ۱۹۹۹ تاسیس zend technologies
- در ۲۰۰۰ PHP4
- در ۲۰۰۴ PHP5
- در ۲۰۱۵ PHP7

PHP یک زبان برنامه نویسی سمت سرویس دهنده است.

قبل از اینکه یادگیری PHP را شروع کنید باید با موارد زیر آشنایی داشته باشید.

سرویس دهنده وب (Web Server): نرم افزاری است که بصورت دائمی بر روی کامپیوتر سرویس دهنده در حال اجرا بوده به درخواست های HTTPگوش داده و با دریافت درخواست از سرویس گیرنده (مثلا" مرورگر وب) به آن پاسخ می دهد. وب سرورهای مهم عبارتند از

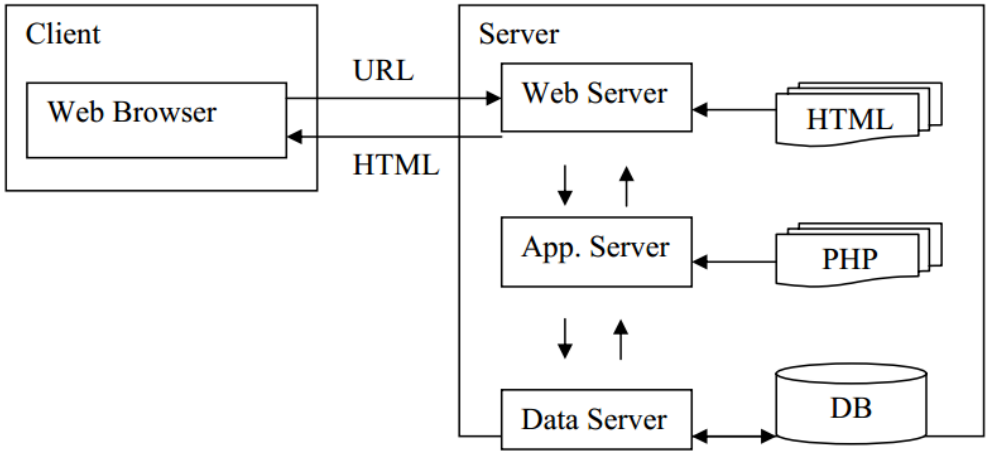
• IIS یا Internet Information Services :یکی از سرویس های ویندوز است و می توان در برنامه Control panel آنرا نصب و فعال کرد.

Apache: یک وب سرور قدرتمند ، اوپن سورس و رایگان و دارای نسخه های مختلفی برای استفاده در محیط ویندوز، لینوکس و غیره می باشد.

سرویس دهنده کاربردی (Application Server): در صورت استفاده از یک زبان اسکریپت نویسی مثل PHP نیاز به مفسر یا امکانات نرم افزاری دیگری برای تفسیر و اجرای دستورات می باشد.

سرویس دهنده داده(Data Server): در صورت نیاز به بانک اطلاعاتی در برنامه ها نیاز به یک نرم افزار DBMS یا بخشی از آن است که امکان مدیریت بانک اطلاعاتی و دسترسی برنامه ها به آن را فراهم می کند. این سرویس دهنده درخواست ها را به زبانی مثل SQL گرفته و پاسخ آنرا به برنامه بر می گرداند، این نرم افزار می تواند SQL، AccessMySQL، Oracle، Serverو غیره باشد.

رابطه بین اجزای مختلف:



همیشه سرویس گیرنده، شروع کننده ارتباط است، وقتی درخواست (Request) به وب سرور رسید، نوع فایل درخواستی بررسی می شود:

اگر مربوط به یک فایل اچتمل بود یا فایل های دیگری مثل txt و ... اگر فایل وجود داشته باشد، محتوای آن به درخواست کننده ارسال می شود.

اگر مربوط به اسکریپت برنامه نویسی بود از نرم افزار مخصوص آن برای اجرا کمک گرفته می شود. اگر دستورات برنامه برای دسترسی به بانک اطلاعاتی بود از نرم افزار فراهم کننده داده ها استفاده می شود . نتیجه هر چه باشدبعنوان پاسخ (Response) به سرویس گیرنده فرستاده می شود. برای شروع یک پروژه برنامه نویسی مبتنی بر وب باید تکنولوژی مورد نظر و مناسب را انتخاب کرد.

ابزارهای مورد نیاز:

ما از ابزار XAMPP استفاده میکنیم که مراحل نصب و پیکربندی آن ساده است. برای کدنویسی نرم‌افزار VScode پیشنهاد میشود.

گرامر زبان PHP

- متغیر ها ظرفی برای ذخیره اطلاعات هستند.
- هر متغیر با علامت $ در ابتدای آن مشخص میشود.
- متغیرها برای ذخیره مقادیر مانند رشته های متنی، اعداد یا آرایه ها استفاده میشوند.
- زمانیکه یک متغیر را تعریف میکنید میتوانید بارها از آن در کدتان استفاده نمایید.
- در PHP بر خلاف زبان C متغیرها نوع خاصی ندارند و از متغیر ها بدون تعریف قبلی میتوان استفاده کرد.

روش صحیح تعریف متغیر در زبان PHP:

```
$varname=value;
$n=10;
$txt="hello world!";
```

قواعد نام گذاری در PHP:

- حروف الفبا a-z A-Z
- اعداد ۰-۹
- underline _
- نام متغیر نباد دارای فضای خالی باشد.
- نکته: نام فقط میتونه با حروف الفبا و underline شروع بشه، با عدد نمیتونه شروع شود!
- بهتر است با _ شروع نکنیم!
- در تعریف متغیرها PHP به حروف بزرگ و کوچک حساس است.

انواع داده در PHP:

اعداد صحیح	integer
اعداد ممیز شناور	float
رشته های کاراکتری	String
دودویی	Boolean
آرایه	Array
تهی	Null
شی	Object
منبع	Resource

مثال: جمع دو عدد

```
<?php
$a=3;
$b=4;
$c=$a+$b;
echo “the value of $a is”.$a;
echo ‘<br/>’;
echo “the value of $b is”.$b;
echo ‘<br/>’;
```

```
echo “the sum of $a and $b is”.$c;
?>
```

ثابت ها در PHP:

- یک ثابت یک شناسه یا یک نام برای یک مقدار ساده است.
- مقدار ثابت نمیتواند تغییر کند.
- نام ثابت باید با یک حرف یا یک «_» آغاز شود.
- در تنظیم ثابت نیازی به استفاده از علامت $ نیست.
- با استفاده از تابع define() میتوان یک ثابت را تنظیم کرد.

این تابع سه پارامتر میگیرد: نام ثابت، مقدار ثابت و پارامتر احتیاری سوم تعیین میکند که آیا نام تابع به حروف بزرگ و کوچک حساس باشد یا خیر.

مقدار پیش فرض false است. به این معنی که به حروف بزرگ و کوچک حساس است.

مثال تنظیم یک ثابت در PHP:

```
<?php
define(“SABET", "Welcome to PHP!");
echo SABET;
 ?>
```

خروجی: Welcome to PHP!

مثال تنظیم یک ثابت در PHP با حساسیت true:

```
<?php
```

```
define(“TEST", "Welcome to PHP!", true);
echo test;
echo TETS;
?>
```

خروجی:

Welcome to PHP!

Welcome to PHP!

عملگرها در PHP:

عملگرها برای انجام عملیات و محاسبات روی مقادیر استفاده می شوند.

- عملگرهای محاسباتی
- عملگرهای انتسابی
- عملگرهای مقایسه ای
- عملگرهای منطقی
- عملگرهای رشته ای
- عملگرهای بیتی

عملگرهای محاسباتی: این عملگرها عملی را روی دو متغیر یا دو عدد انجام می دهند.

عملگر	توضیحات
+	جمع

-	تفریق
*	ضرب
/	تقسیم
**	توان
%	باقیمانده تقسیم
++	افزایش به میزان یک واحد
--	کاهش به میزان یک واحد

عملگرهای انتسابی:

عملگر	مثال	هم ارز با
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

عملگرهای مقایسه ای: برای مقایسه دو متغیر به کار میروند و نتیجه آن ها true یا false است.

عملگر	توضیح
= =	برابری
=!	نابرابری
>	بزرگتر از
<	کوجکتر از
>=	بزرگتر یا مساوی با
<=	کوچکتر یا مساوی با

عملگرهای منطقی: این عملگرها با مقدارهای true و false کار می کنند و آن ها را با هم ادغام می کند.

عملگر	توضیح
&&	and
||	or
!	not

عملگرهای رشته ای: این عملگر، دو رشته را به هم وصل می کند.

عملگر	توضیح
.	الحاق

عملگرهای بیتی: این عملگرها بر روی بیت های یک متغیر عملی را انجام میدهند.

عملگر	توضیحات
~	not
&	and
|	or
^	xor
<<	شیفت به چپ
>>	شیفت به راست

آرایه ها:

آرایه ها انواع خاصی از متغیرها به حساب می آیند که می توانند چندین داده را در قالب یک نام ذخیره کنند. اگر لیستی از آیتم ها برای مثال یک لیست از ماشین ها داشته باشیم و بخواهیم آن ها را در متغیر ذخیره کنیم چیزی شبیه به این خواهیم داشت:

```
$cars1="Samand";
 $cars2="Volvo";
 $cars3="BMW";
```

آرایه می تواند مقادیر متغیر را تحت یک نام برای شما نگه دارد و شما از طریق نام آرایه می توانید به مقادیر دسترسی داشته باشید.

هر آیتم در آرایه ایندکس منحصربفردی برای خود دارد که به راحتی از طریق ایندکس می توانید به مقادیر دسترسی پیدا کنید.

در PHP سه نوع آرایه وجود دارد.

- **آرایه عددی (Indexed array)**: منظور از عددی ایندکس آرایه است، یعنی یک آرایه با ایندکس عددی
- **آرایه انجمنی (Associative array)**: یک آرایه که به جای ایندکس عددی از یک نام یا مقدار برای ایندکس گذاری استفاده کرده است.
- **آرایه چند بعدی (Multidimensional array)**: یک آرایه که مقادیر هر سلول آن آرایه ای دیگر است.

آرایه عددی

در یک آرایه عددی مقادیر هر سلول آرایه با یک ایندکس عددی مشخص می شود.

دو روش برای ایجاد چنین آرایه ای وجود دارد:

در مثال زیر ایندکس به صورت اتوماتیک ساخته میشود و ایندکس از ۰ شروع می شود.

```
$cars=array("Samand","Volvo","BMW","Toyota");
```

در مثال زیر به صورت دستی میتوان ایندکس را ساخت.

```
$cars[0]="Samand";
 $cars[1]="Volvo";
 $cars[2]="BMW";
 $cars[3]="Toyota";
```

در مثال زیر بعد ار مقداردهی آرایه شما میتوانید به مقادیر هر سلول به وسیله نام و ایندکس آرایه دسترسی پیدا کنید.

```
<?php
 $cars[0]="Saab";
 $cars[1]="Volvo";
 $cars[2]="BMW";
 $cars[3]="Toyota";
 echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
 ?>
```

خروجی کد بالا:

```
Saab and Volvo are Swedish cars.
```

آرایه های انجمنی

در یک آرایه انجمنی ایندکس هر سلول از آرایه با یک نام یونیک مشخص میشود.

زمان ذخیره مقادیر سلول ها باید یک نام مشخص و یونیک به آن سلول اختصاص دهید.

مثال: آرایه ای که سن افراد مختلف را نشان می دهد:

```
$ages = array("Ali"=>32, "Reza"=>30, "Amir"=>34);
```

به صورت زیر هم میتوان یک آرایه را تعریف کرد:

```
$ages['Ali'] = "32";
 $ages['Reza'] = "30";
 $ages['Amir'] = "34";
```

در ادامه نشان داده شده است که چگونه می توان از طریق نام و ایندکس آرایه به محتویات آن دسترسی داشت.

```
<?php
 $ages['Ali'] = "32";
 $ages['Reza'] = "30";
 $ages['Amir'] = "34";
 echo "Ali is " . $ages['Ali'] . " years old.";
?>
```

خروجی کد بالا:

```
Ali is 32 years old.
```

آرایه چندبعدی

هر یک از عناصر آرایه در PHP می توانند از هر نوعی باشند پس میتوانیم آرایه را نیز به عنوان عضو عناصر در نظر بگیریم. بنابراین به زبان ساده تر میتوانیم یک آرایه را داخل آرایه دیگر تعریف کینم و به همین ترتیب.

مثال: در آرایه چند بعدی که ایندکس ان به صورت اتوماتیک تعریف می شود:

```
$families = array(array("Ali","Reza","Sara"),
  array("Amir"),
  array("Poya","Parniya")
   );
```

یا به صورت ایندکس مقداری:

```
$families    =    array("Ahmadi"=>array("a"=>"Ali","b"=>"Reza","c"=>"Sara"),
"Naderi"=>array("a"=>"Amir"),
"Mohamadi"=>array("a"=>"Poya","b"=>"Parniya")
 );
```

نحوه دسترسی به یک سلول از آرایه چندبعدی:

```
echo "Is " . $families['Ahmadi']['b'] . " a part of the Ahmadi family?";
```

خروجی کد بالا:

```
Is Reza a part of the Ahmadi family?
```

مرتب کردن آرایه ها در PHP:

عناصر یک آرایه را میتوان به صورت الفبایی یا عددی از کوچک به بزرگ یا بالعکس مرتب نمود.

توابع مرتب سازی آرایه ها:

- sort() - مرتب کردن مقادیر آرایه از کوچک به بزرگ
- rsort() - مرتب کردن مقادیر آرایه از بزرگ به کوچک
- asort() - مرتب کردن آرایه های انجمنی از نزولی به صعودی (بر اساس مقدار)
- ksort() - مرتب کردن آرایه های انجمنی از نزولی به صعودی (بر اساس کلید)
- arsort() - مرتب کردن آرایه های انجمنی از صعودی به نزولی (بر اساس مقدار)
- krsort() - مرتب کردن آرایه های انجمنی از صعودی به نزولی (بر اساس کلید)

مثال: آرایه انجمنی $age بر اساس مقادیر سلول ها از بزرگ به کوچک مرتب شده است:

```
<?php
 $age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
 ksort($age);
  foreach($age as $x=>$x_value)
 {
echo "Key=" . $x . ", Value=" . $x_value;
 echo "<br>";
}
?>
```

خروجی کد بالا:

```
Key=Ben, Value=37
 Key=Joe, Value=43
 Key=Peter, Value=35
```

برگرداندن تعداد عناصر یک آرایه

با استفاده از تابع count () میتوان تعداد عناصر یک آرایه را برگرداند.

مثال:

```
<?php
$cars=array("Volvo","BMW","Toyota");
```

```
echo count($cars);
?>
```

خروجی۳

نحوه استفاده از تابع count ():

```
Count(array,mode);
```

پارامتر	توضیحات
array	الزامی است، نام آرایه را تعیین می کند.
mode	اختیاری است، در آرایه های چند بعدی، نحوه شمارش عناصر را تعیین می کند. ۰: مقدار پیشفرض، در آرایه های چند بعدی، تعداد کل عناصر را برنمی گرداند. ۱: در آرایه های چند بعدی، تعداد کل عناصر را برمی گرداند.

توابع var_dump و print_r

تابع var_dump اطلاعات ساختاری (شامل نوع و مقدار) در رابطه با متغیرها را نمایش می دهد. اگر متغیر آرایه ای باشد، آنها به صورت بازگشتی به همراه مقادیری که دارند گسترش می یابند تا ساختار را نشان دهند. این تابع همچنین نشان می دهد که کدام مقادیر و ویژگی های شی مرجع هستند.

تابع print_r اطلاعاتی در رابطه با یک متغیر نشان می دهد که توسط انسان قابل خواندن باشند. در مورد متغیرهای آرایه این تابع در یک فرمت شامل کلید و عنصر خروجی را ارایه می دهد. همین حالت برای اشیا نیز صدق می کند.

(نکته: در مبحث **توابع** بیشتر به این موضوع پرداخته میشود.)

مثال:

```
<?php
    $arr=array("roz","yas");
    var_dump($arr);
    echo '</br>';
    print_r($arr);
     ?>
```

متغیرهای رشته ای و چند تابع دیگر

متغیرهای رشته ای برای ذخیره ی مجموعه ای از کاراکترها استفاده می شود.

بعد از اینکه یک متغیر رشته ای ایجاد کردید می توانید آن را دستکاری کنید. یک متغیر رشته ای مستقیما می تواند در یک تابع استفاده شود. همچنین می توانید آن را در یک متغیر ذخیره کنید. به عنوان مثال:

```
<?php
$txt=”hello”;
Echo $txt;
?>
```

معرفی چند اپراتور ساده :

قبلا در مباحث قبلی با عملگر الحاق (.) آشنا شده اید. از این عملگر برای چسباندن رشته ها به هم استفاده می شود:

```
<?php
echo “سلام” . ”روز خوبی داشته باشید”;
?>
```

تابع strlen():

از این تابع برای برگرداندن طول یک رشته استفاده می شود.

```
<?php
echo strlen(“سلام”);
echo “</br>”;
echo strlen(“hello”);
?>
```

توجه داشته باشید که به ازای هر کاراکتر فارسی ۲ بیت فضا در نظر گرفته می شود.

تابع strops():

برای پیدا کردن مکان یک کاراکتر یا یک کلمه در یک رشته به کار می رود.

```
<?php
echo strops(“welcome php” , “m”);
?>
```

جملات شرطی:

اغلب اوقات هنگام نوشتن یک اسکریپت لازم است که تصمیم های متفاوتی در مقابل نتایج مختلف گرفته شود. برای تحقق این موضوع از جملات شرطی استفاده می شود.

انواع جملات شرطی دز PHP:

۱- if...: هنگامیکه شرط درست باشد دستور مقابل if اجرا می شود.

۲- if ... else: اگر شرط درست باشه دستور مقابل if و گرنه دستور مقابل else اجرا می شود.

۳- if ... elseif ... else: برای اجرای یک دستور از بین چند دستور کاربرد دارد.

۴- switch: برای انتخاب و اجرای یک دستور از بین چند دستور استفاده می شود.

ساختار دستور if:

```
if (Condition)
 {
 Statement 1
 Statement 2
 ...
 }
```

مثال: قطعه کد زیر با استفاده از تابع Date() پیغام متناسب با روز هفته را نشان میدهد:

```
<?php
$d=date("D");
      if ($d=="Fri")
         echo"Have Nice Weekend";
<?
```

ساختار دستور if else: این دستور در واقع کامل کننده دستور if است، تعیین می کند که اگر شرط درست نبود چه دستوری اجرا شود.

```
if (condition)
 code to be executed if condition is true;
Else
 code to be executed if condition is false;
```

به مثال زیر توجه کنید:

```
<?php
	$d=date("D");
	if ($d=="Fri")
	   echo"Have Nice Weekend";
	else {
echo"have nice day";
	}
	?>
```

در دستور بالا چنانچه شرط درست نباشد، کد بعد از else اجرا خواهد شد.

ساختار دستور if elseif else: این نوع if برای اجرای یک دستور از بین چند دستور است.

```
if (condition)
 code to be executed if condition is true;
 elseif
 (condition) code to be executed if condition is true;
else
code to be executed if condition is false;
```

به مثال زیر توجه کنید:

```
<?php
$d=date("D");
	if ($d=="Fri")
	   echo"Have Nice Weekend";
```

```
elseif ($d=="Sun")
    echo 'have nice Sunday';
else {
echo"have nice day;"
}
?>
```

دستور switch:

برای انتخاب و اجرای یک دستور از بین چند دستور استفاده می شود.

این دستور در برخی موارد شباهت بسیار زیادی به if elseif else دارد.

دستور سوییچ یک حالت خاص دارد به نام default که اگر هیچکدام از مقادیر درست نبود، آن قسمت اجرا می شود.

ساختار دستور سوییچ به صوزت زیر است:

```
switch (n)
{
case label1:
code to be executed if n=label1;
break;
case label2:
code to be executed if n=label2;
break;
default:
code to be executed if n is different from both label1 and label2;
```

```
}
```

مثال:

```
<?php
$d=5;
      switch ($d){
      case 1:
$weekday="Sat";
         echo $weekday;
         break;
      case 2:
$weekday="Sun";
         echo $weekday;
         break;
      case 3:
$weekday="Mon";
         echo $weekday;
         break;
      case 4:
$weekday="Tue";
         echo $weekday;
         break;
```

```
        case 5:
$weekday="Wed";
            echo $weekday;
            break;
        case 6:
$weekday="Thu";
            echo $weekday;
            break;
        default:
$weekday="Fri";
            echo $weekday;
}
<?
```

در کد بالا با توجه به عدد هر کیس، روز متناظر آن را در خروجی نمایش می‌دهد.

حلقه های تکرار شونده:

اغلب مواقعی که کد مینویسیم میخواهیم که یک مجموعه دستورات بارها و بارها اجرا شوند، بجای اینکه آن چند خط که اغلب یکسان هستند را در دستورات تکرار کنیم از حلقه ها استفاده می کنیم.

انواع دستورات حلقه ای در PHP:

While: برای پیاده سازی حلقه های تکرار نامحدود و نا معین و هر نوع تکراری استفاده می شود. تا زمانیکه شرط برقرار باشد دستورات را اجرا می کند و شرط قبل ار اجرای دستورات را چک می کند. نحوه استفاده:

```
while (condition)
 {
code to be executed;
 }
```

در مثال زیر تا زمانیکه مقدار متغیر i کوچکتر مساوی ۵ باشد، کد درون حلقه اجرا خواهد شد.

```
<?php
    $i=5;
    while($i<=5)
      {
echo "The number is " . $i . "<br />";
      $i++;
      }
    ?>
```

Do/while: برای حداقل یک بار تکرار از do/while استفاده می شود. این دستور شبیه به دستور قبلیست با این تفاوت که شرط در پایان اجرای دستورات چک می شود. یعنی اگر از ابتدا هم شرط برقرار نباشد، دستورات یکبار اجرا میشوند و بعد شرط چک میشود. نحوه استفاده:

```
Do
 {
 code to be executed;
 }
 while (condition);
```

به مثال زیر توجه کنید:

```
<?php
    $i=1;
    do {
    $i++;
    echo"the number is".$i."</br>";
    }
    while ($i<=5);
 ?>
```

For: دستورات از یک عدد مشخص تا یک عدد مشخص دیگری اجرا می شوند. نحوه استفاده:

```
for (init; condition; increment)
{
code to be executed;
}
```

در مثال زیر عبارت hello world ۱۰ بار چاپ میشود.

```
<?php
For($i=1; $i<=10; $i++)
{
echo’hello world!’.’<br/>’;
}
?>
```

Foreach: مشابه for است و برای تکرار روی آرایه هاست.

این دستور تمام مقادیر آرایه را یکی یکی از ابتدایی تا انتهایی به درون value میریزد و دستورات را اجرا میکند.

نحوه استفاده:

```
foreach ($array as $value)
 {
 code to be executed;
}
```

مثال foreach

```
<?php
$x=array("one","two","three");
      foreach ($x as $value){
          echo $value . ‘<br/>;
}
<?
```

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت. برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد:

```php
<?php

    echo 'Demonstrating the use of break.'.'<br/>';

    for ($x = 1; $x < 10; $x++)
    {
        if ($x == 5)
            break;

            echo $x . '<br/>';
    }

    echo '<br/>'. 'Demonstrating the use of continue.'.'<br/>';

    for ($x = 1; $x < 10; $x++)
    {
        if ($x == 5)
            continue;

            echo $x . '<br/>';
    }

?>
```

اشاره گرها:

روش مستقیم دسترسی به حافظه برای ذخیره‌سازی مقداری در آن، و یا بازیابی محتوای ذخیره شده، استفاده از نام متغیر است. همین که متغیری معرفی شد، سیستم بر اساس نوع اعلام شده، تعداد بایت لازم را به آن اختصاص می‌دهد و آدرس متغیر، شماره اولین بایت از مجموعهٔ اختصاص یافته است. پس از آن به راحتی از طریق نام متغیر به محل مورد نظر دسترسی یافته، محتوای آن پردازش می‌گردد. اما گاهی لازم می‌شود به جای نام، آدرس متغیر در اختیار برنامه نویس قرار گیرد تا از طریق آن دستیابی به محل مربوطه صورت گیرد.

با استفاده از اشاره گرها (pointers) در PHP ، می توانیم به عناصر یک آرایه ارجاع بدهیم (اشاره کنیم). این نوع ارجاع (اشاره) به گونه ای است که آزادی عمل زیادی خواهیم داشت. مثلا اگر به یک عنصر از یک آرایه، ارجاع داده ایم (برای عنصر فعلی، کلمه current به کار می رود)، با کلمه next ، می توانیم اعلام کنیم که به عنصر بعد از آن در آرایه ارجاع داده شود. همچنین کلمه reset ، برای ارجاع به اولین عنصر آرایه می باشد (به نحوی، بازگشت به اولین عنصر خواهد بود).

مثال:

```
<?php
$numbers = array(4,2,5,9,7,6);
echo("first : " . current($numbers));
echo("<br />");

next($numbers);
echo("second : " . current($numbers));
echo("<br />");

next($numbers);
echo("third : " . current($numbers));
echo("<br />");

reset($numbers);
echo("first : " . current($numbers));
?>
```

نتیجه کد بالا:

```
first : 4
second : 2
third : 5
first : 4
```

ابتدا یک آرایه با نام numbers تعریف کرده‌ایم. سپس از دستور current($numbers)استفاده نموده ایم. این دستور اعلام می کند که عنصری مد نظر است که اشاره گر (pointer) بر روی آن است (به آن اشاره می کند). چون قبلا از اشاره گرها استفاده نکرده ایم، بنابراین فعلا اشاره گر بر روی اولین عنصر از آرایه numbers می باشد. پس از چاپ اولین عنصر آرایه، از دستور next($numbers)استفاده کرده ایم که باعث می شود که اشاره گر، بر روی عنصر بعدی آرایه numbers برود، یعنی عنصر دوم آرایه. بنابراین در این زمان، دستور current($numbers)، عنصر دوم آرایه را برمی گرداند. پس از چاپ عنصر دوم آرایه در خروجی، دوباره از next($numbers)استفاده کرده ایم که باعث می شود اشاره گر بر روی عنصر سوم آرایه برود (به عنصر سوم آرایه اشاره کند). بنابراین در این زمان، دستور current($numbers)، عنصر سوم آرایه را برمی گرداند. در آخر نیز، از دستور reset($numbers)استفاده کرده ایم که باعث می شود که اشاره گر، به اولین عنصر آرایه باز گردد، بنابراین دستور current($numbers)، اولین عنصر آرایه را برمی گرداند.

برنامه نویسی ماژولار چیست؟

طراحی و توسعه پروژه های نرم افزاری بزرگ یکی از نگرانی های همیشگی توسعه دهندگان بوده است چرا که درک عملکرد بخش های مختلف در پروژه هایی که از هزاران خط کدنویسی تشکیل می شوند بسیار پیچیده است. قابلیت ماژولار (Modular) برای رفع این پیچیدگی و تکرار کدهای یکسان طراحی شده است. در حقیقت هدف برنامه نویسی ماژولار این است که کدهای هر بخش از یک نرم افزار عملکرد خاصی داشته باشند.

در برنامه نویسی ماژولار در واقع عملکردهای مرتبط به هم در یک ماژول (Module) قرار داده می شوند و از آنجایی که هر ماژول به صورت کاملا مستقل فعالیت دارد، یک نرم افزار به مجموعه ای از واحدهای کاربردی (Functional Units) تبدیل خواهد شد

برنامه نویسی ماژولار با برنامه نویسی ساختارگرا و برنامه نویسی شی‌گرا ارتباط نزدیکی دارد و هدف مشترکی که در تمامی این تکنیک‌ها وجود دارد، تسهیل ساخت برنامه‌ها و سیستم‌های بزرگ نرم افزاری با تجزیه‌ی آن به قطعات کوچک‌تر است. در برنامه نویسی ماژولار کد برنامه از ماژول‌ها یا واحدهای مختلفی تشکیل شده است که هر کدام از آن‌ها به طور جداگانه تهیه می‌شوند. این امر به توسعه دهندگان مختلف این امکان را می‌دهد تا قطعات گسسته‌ای از سیستم را در دست بگیرند و بدون نیاز به درک بقیه، آن‌ها را طراحی و پیاده‌سازی کنند. ایده‌ی اصلی برنامه نویسی ماژولار این است که پس از ساخت ماژول‌های مستقل و قابل تعویض، می‌توان آن‌ها را به یکدیگر متصل کرد تا یک برنامه‌ی کامل را ایجاد کنند؛ بدون این‌که هر توسعه دهنده نیاز به درک همه‌ی کارهای انجام شده توسط سایر توسعه دهندگان تیم، داشته باشد.

مزیت های برنامه نویسی ماژولار

پروژه های بزرگ اغلب متشکل از چندین برنامه نویس هستند که بر روی هزاران خط کدنویسی فعالیت می کنند بنابراین پیگیری و دوباره نویسی کدها برای آنها بسیار سخت خواهد بود. اما مدل ماژولار از بروز سناریوی تکراری در برنامه نویسی جلوگیری می کند؛ به عبارت دیگر ویژگی ماژولار، کدها را با توجه به وظیفه ای که اجرا می کنند فروشکافی و سازماندهی خواهد کرد. در این نوع از برنامه نویسی استفاده مجدد، شناسایی باگ و مدیریت کدها بسیار آسان خواهد بود. در ادامه این مطلب ۷ مورد از مهم ترین مزایای برنامه نویسی ماژولار را به شما معرفی خواهیم کرد.

۱- *شناسایی آسان باگ:*

شناسایی باگ یا همان دیباگ کردن (Debugging) پروژه های نرم افزاری بزرگ پیچیده است به همین دلیل زمان زیادی از فرآیند برنامه صرف جستجو و شناسایی خطاها خواهد شد. در حالی که اگر یک برنامه مجهز به قابلیت ماژولار باشد، هر بخش آن کدهای مختص به خود را خواهد داشت در نتیجه اگر عملکرد خاصی از برنامه با مشکل روبرو شود، برنامه نویسان به راحتی با جستجو در بخش کوچکی از کدها آن را شناسایی خواهند کرد.

۲- استفاده مجدد از کد:

کدهای ماژولار امکان استفاده مجدد را در اختیار برنامه نویسان قرار می دهد. در صورتی که Task ها به توابع یا کلاس های مشخصی طبقه بندی شده باشند، برنامه نویسان در صورت لزوم اجرای دوباره Task می توانند از آن کدهای خاص استفاده کنند. اگر کدها به صورت مشخصی سازمان دهی نشده باشند آنگاه مراجعه، تقسیم بندی و تکمیل آنها در یک محیط برنامه نویسی دیگر بسیار مشکل یا غیر ممکن خواهد بود.

۳- خوانایی:

کد ماژولار سازماندهی سطح بالایی دارد به این معنی که برنامه نویسان می توانند با در نظرگیری عملکرد یک کد، هر بخش از آن را سازمان دهی کنند. به طور کل توسعه دهندگان قادر خواهند بود بر اساس الگوی سازمانی خود، کدهای مربوطه را شناسایی یا منشن (Mention) کنند. علاوه بر این برنامه نویسان دیگری که بر روی کد کار خواهند کرد نیز می توانند از الگوی سازمان برای خواندن آن استفاده کنند. به عبارت دیگر کدها برای استفاده چندین برنامه نویس بهینه سازی شده و درجه سختی کار را کاهش می دهند.

٤- قابل اطمینان:

یکی از بهترین مزایای برنامه نویسی ماژولار قابلیت اطمینان به آن است. در حقیقت کدهایی که خواندن، دیباگینگ، اشتراک گذاری و نگهداری از آن آسان تر باشد اغلب با خطاهای کمتری نیز مواجه می شود. این قابلیت در مواقعی که صدها نفر بر روی یک پروژه بزرگ فعالیت می کنند بسیار کاربردی خواهد بود. تمام توسعه دهندگان بر روی کدهایی کار می کنند که در آینده نیز مورد استفاده برنامه نویسان دیگر قرار خواهند گرفت. به طور کل می توان گفت که کدهای ماژولار یکی از ضرورت های اصلی فرآیند توسعه پروژه های نرم افزاری بزرگ محسوب می شود

٥- قابل مدیریت:

استراتژی ماژولار در حقیقت کل فرآیند توسعه نرم افزار را به بخش های قابل کنترل تقسیم می کند. متمرکز شدن بر روی یک کد خاص در فرآیند توسعه نرم افزارهای بزرگ کار بسیار دشواری محسوب می شود اما می توانید با تقسیم کردن آن به وظایف فردی، میزان سختی و پیچیدگی آن

را کاهش دهید. البته برنامه نویسان به لطف مدل ماژولار، بدون آن که نگرانی خاصی درباره حجم پروژه داشته باشند می توانند از سردرگمی خود جلوگیری کنند.

۶- *برنامه نویسی تیمی:*

قابلیت ماژولار امکان برنامه نویسی تیمی را در اختیار توسعه دهندگان قرار می دهد. در حقیقت سرپرست تیم، پروژه های بزرگ را بین تیم های برنامه نویسی تقسیم می کند به گونه ای که وظیفه (Task)خاصی را به هر برنامه نویس محول خواهد کرد. در آخر نتیجه کار برنامه نویسان مختلف با هم ترکیب شده و به عنوان یک نرم افزار منتشر می شود. این ویژگی به طور کل باعث افزایش سرعت روند توسعه پروژه و همچنین بروز رسانی های نرم افزار خواهد شد.

۷- *کیفیت:*

قابلیت ماژولار همچنین باعث ارتقای کیفیت کدها می شود. زمانی که یک نرم افزار به بخش های کوچک تقسیم شود و مسئولیت هر بخش با یک برنامه نویس باشد، آنگاه کیفیت هر بخش نیز افزایش خواهد یافت. به عبارت دیگر در این شرایط برنامه نویس دیگر به کل نرم افزار توجه نخواهد کرد بلکه تمرکزش را بر روی بخشی از کدهای مختص به خود خواهد گذاشت. علاوه بر این هنگامی که کل قسمت های مختلف پروژه با یکدیگر ترکیب شوند، احتمال بروز خطا نیز به طرز چشمگیری کاهش می یابد.

برخی دیگر از مزایای برنامه نویسی ماژولار

- کدها در چندین فایل ذخیره و نگهداری می شوند.
- کدها طولانی نیستند.
- پیش از کدنویسی، برنامه ریزی و طراحی برنامه توسط مدیر پروژه یا تحلیل گر سیستم انجام می شود.
- کدها به راحتی قابل درک و ساده هستند.
- کدهای هر بخش، به راحتی قابل استفاده در بخش های دیگر هستند.
- برنامه نویس می تواند متد خاصی را ایجاد کرده و از آن در قسمت های مختلف برنامه، بدون نیاز به کدنویسی مجدد استفاده کند.
- کنترل کردن متغیرها بسیار آسان است.

- نام متغیرها و ماژول ها، منطقی و مرتبط با یکدیگر انتخاب می شوند.

توابع:

توابع به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد.

توابع در PHP و اکثر زبانهای برنامه نویسی بر دو نوع‌اند:

- توابع از پیش تعریف شده
- توابعی که توسط کاربر تعریف می‌شوند.

ساده‌ترین ساختار یک تابع به صورت زیر است:

```
function MethodName()
{
    //code to execute;
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک تابع برای چاپ یک پیغام در صفحه نمایش استفاده شده است:

```
1   <?php
2
3       function PrintMessage()
4       {
5           echo 'Hello World!';
6       }
7
8       PrintMessage ();
9
10  ?>
```

در خطوط 3-6 یک تابع تعریف کرده‌ایم. همانطور که مشاهده می‌کنید در خط 3 و برای تعریف تابع از کلمه کلیدی function سپس نام تابع و بعد از آن پرانتز باز و بسته استفاده کرده‌ایم. نام تابع ما PrintMessage() است. به این نکته توجه کنید که در نامگذاری تابع از روش پاسکال (حرف اول هر کلمه بزرگ نوشته می‌شود) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص توابع استفاده کنید. بهتر است در نامگذاری توابع از کلماتی استفاده شود که کار آن تابع را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor. همچنین به عنوان مثال اگر مقدار برگشتی (در درس‌های آینده توضیح می‌دهیم) تابع یک مقدار بولی باشد، می‌توانید اسم تابع خود را به صورت یک کلمه سوالی انتخاب کنید مانند IsLeapyear یا IsTeenager. ولی از گذاشتن علامت سؤال در آخر اسم تابع خودداری کنید. دو پرانتزی که بعد از نام تابع می‌آید نشان دهنده آن است که نام متعلق به یک تابع است. بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه تابع را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم. در خط 8 تابع را صدا می‌زنیم. برای صدا زدن یک تابع کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم. برای اجرای تابع PrintMessage() برنامه از خط به محل تعریف تابع PrintMessage() می‌رود. مثلاً وقتی ما تابع PrintMessage() را در خط 8 صدا می‌زنیم برنامه از خط 8 به خط 3، یعنی جایی که تابع تعریف شده می‌رود و کدهای آن را اجرا می‌کند.

مقدار برگشتی از یک تابع

توابع می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک تابع است و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی تابع است. نکته مهم در مورد یک تابع، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک تابع آسان است. کافیست در تعریف تابع به روش زیر عمل کنید:

```
function MethodName()
{
    return value;
}
```

همانطور که در خط مشاهده می‌کنید مقدار بازگشتي از تابع را جلوي دستور return می‌نویسیم. مثال زیر یک تابع که دارای مقدار برگشتی است را نشان می‌دهد:

```
1   <?php
2
3       function CalculateSum()
4       {
5           $firstNumber  = 10;
6           $secondNumber = 5 ;
7           $sum          = $firstNumber + $secondNumber ;
8
9           return $sum;
10      }
11
12      $result = CalculateSum();
13      echo $result;
14
15  ?>
```

```
15
```

در خطوط 5 و 6 مثال فوق، دو متغیر تعریف و مقدار دهی شده‌اند. توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر توابع قابل دسترسی نیستند و فقط در تابعی که در آن تعریف شده‌اند قابل استفاده هستند. در خط 7 جمع دو متغیر در متغیر sum قرار می‌گیرد. در خط 9 مقدار برگشتی sum توسط دستور return فراخوانی می‌شود. در خط 12 یک متغیر به نام result تعریف می‌کنیم و تابع ()CalculateSum را فراخوانی می‌کنیم.

تابع ()CalculateSum مقدار 15 را بر می‌گرداند که در داخل متغیر result ذخیره می‌شود. در خط 13 مقدار ذخیره شده در متغیر result چاپ می‌شود. تابعی که در این مثال ذکر شد تابع کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در تابع بالا نوشته شده ولی همیشه مقدار برگشتی 15 است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار 15 را به آن اختصاص دهیم. این تابع در صورتی کارامد است که پارامترهایی به آن اضافه شود که در درس‌های آینده توضیح خواهیم داد. هنگامی که می‌خواهیم در داخل یک تابع از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید:

```php
<?php

    function GetNumber()
    {
        $number = 11 ;

        if ($number > 10)
        {
            return $number;
        }
        else
        {
            return 0;
        }
    }

    $result = GetNumber();
    echo $result;

?>
```

در خطوط 15-3 یک تابع با نام GetNumber() تعریف شده است. در خط 5 متغیری با مقدار 11 مقداردهی شده است که در خط 7 با مقدار 10 مقایسه می‌شود و چون مقدار این متغیر از 10 بیشتر است پس دستور return اول مقدار 11 را برمی گرداند. حال اگر مقدار این متغیر از 10 کمتر باشد دستور return مربوط به قسمت else اجرا و مقدار صفر چاپ می‌شود. که از کاربر یک عدد بزرگ‌تر از 10 را می‌خواهد. اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه نتیجه چاپ نمی‌شود. چون اگر شرط دستور if نادرست باشد برنامه به قسمت else می‌رود تا مقدار صفر را بر گرداند و چون قسمت else حذف شده است برنامه هیچ مقداری را چاپ نمی‌کند و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد برنامه هیچ مقداری را چاپ نمی‌کند. و آخرین مطلبی که در این درس می‌خواهیم به شما آموزش دهیم این است که شما می‌توانید از یک تابع که مقدار برگشتی ندارد خارج شوید. استفاده از return باعث خروج از بدنه تابع و اجرای کدهای بعد از آن می‌شود.

```php
<?php

    function TestReturnExit()
    {
```

```
        echo 'Line 1 inside the method TestReturnExit()';

        return;

        echo 'Line 2 inside the method TestReturnExit()';
    }

    TestReturnExit();

?>
```

```
Line 1 inside the method TestReturnExit()
```

در برنامه بالا نحوه خروج از تابع با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه تابع تعریف شده (TestReturnExit()) فراخوانی و اجرا می‌شود.

پارامترها و آرگومانها

پارامترها داده‌های خامی هستند که متد آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید که بر طبق آنها کارش را به پایان برساند. یک متد می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک متد با N پارامتر نشان داده شده است:

```
function MethodName(param1,param2, ...paramN)
{
    //code to execute;
}
```

پارامترها بعد از نام متد و بین پرانتزها قرار می‌گیرند. بر اساس کاری که متد انجام می‌دهد می‌توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومانهای آن را نیز تأمین کنید. آرگومان‌ها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. اجازه بدهید که یک مثال بزنیم:

```php
<?php

    function CalculateSum($number1,$number2)
    {
        return $number1 + $number2;
    }

    $result = CalculateSum(10,5);
    echo $result;

?>
```

در برنامه بالا یک متد به نام CalculateSum() (خطوط 3-6) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. متد دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. در بدنه متد دستور return نتیجه جمع دو عدد را بر می‌گرداند. در خط 8 دو عدد 5 و 10 را به عنوان آرگومان به متد ارسال می‌کنیم. بعد از ارسال مقادیر 5 و 10 به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه متد (خط 5) دو مقدار با هم جمع می‌شوند و در خط 9 نتیجه چاپ می‌شود. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما متدهای کارآمدتری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

```php
<?php

    function MyMethod()
    {
        return 5;
    }

    function AnotherMethod($number)
    {
        echo $number;
    }

    AnotherMethod(MyMethod());

?>
```

```
5
```

چون مقدار برگشتی متد MyMethod() عدد 5 است و به عنوان آرگومان به متد AnotherMethod() ارسال می‌شود خروجی کد بالا هم عدد 5 است.

پارامترهای اختیاری

پارامترهای اختیاری همانگونه که از اسمشان پیداست اختیاری هستند و می‌توان به آنها آرگومان ارسال کرد یا نه. این پارامترها دارای مقادیر پیشفرضی هستند. اگر به اینگونه پارامترها آرگومانی ارسال نشود از مقادیر پیشفرض استفاده می‌کنند. به مثال زیر توجه کنید:

```php
<?php

    function PrintMessage($String = "Welcome to PHP Tutorials!")
    {
        echo $String . '<br/>';
    }

    PrintMessage();
    PrintMessage("Learn PHP Today!");

?>
```

```
Welcome to PHP Tutorials!
Learn PHP Today!
```

متد ()PrintMessage (خطوط 3-6) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت = یک مقدار را به یک پارامتر اختصاص داد (مثال بالا خط 3). دو بار متد را فراخوانی می‌کنیم. در اولین فراخوانی (خط 8) ما آرگومانی به متد ارسال نمی‌کنیم بنابراین متد از مقدار پیشفرض (Welcome to PHP Tutorials!) استفاده می‌کند. در دومین فراخوانی (خط 9) یک پیغام (آرگومان) به متد ارسال می‌کنیم که جایگزین مقدار پیشفرض پارامتر می‌شود.

ارسال آرگومان به روش ارجاع و مقدار

آرگومان‌ها را می‌توان به کمک ارجاع ارسال کرد. این بدان معناست که شما آدرس متغیری را ارسال می‌کنید نه مقدار آن را. ارسال با ارجاع زمانی مفید است که شما بخواهید یک آرگومان که دارای مقدار بزرگی است (مانند یک آبجکت) را ارسال کنید. در این حالت وقتی که آرگومان ارسال شده را در داخل متد اصلاح می‌کنیم مقدار اصلی آرگومان در خارج از متد هم تغییر می‌کند. اما در روش مقدار مقدار اصلی آرگومان در خارج از متد تغییر نمی‌کند. در زیر دستورالعمل پایه‌ای تعریف پارامترها که در آنها به جای مقدار از آدرس استفاده شده است نشان داده شده:

```php
function FunctionName(& param1)
{
    //code to execute;
}
```

همانطور که در کد بالا مشاهده می‌کنید باید قبل از پارامتری که قرار است به روش ارجاع مقداری به آن ارسال شود علامت (&) قرار داده شود. اجازه دهید که تفاوت بین ارسال با ارجاع و ارسال با مقدار آرگومان را با یک مثال توضیح دهیم.

```php
<?php
    function ModifyNumberVal($number)
    {
        $number += 10;
        echo 'Value of number inside method is '.$number.'<br/>';
    }

    function ModifyNumberRef(&$number)
    {
        $number += 10;
        echo 'Value of number inside method is '.$number.'<br/>';
    }

    $num = 5;

    echo 'num = '.$num;

    echo '<br/><br/>';

    echo 'Passing num by value to method ModifyNumberVal() ...<br/>';
    ModifyNumberVal($num);
    echo 'Value of num after exiting the method is '.$num.'<br/>';

    echo '<br/><br/>';

    echo 'Passing num by ref to method ModifyNumberRef() ...<br/>';
    ModifyNumberRef($num);
    echo 'Value of num after exiting the method is  '. $num.'<br/>';
?>
```

```
num = 5

Passing num by value to method ModifyNumberVal() ...
Value of number inside method is 15.
Value of num after exiting the method is 5.

Passing num by ref to method ModifyNumberRef() ...
Value of number inside method is 15.
```

در برنامه بالا دو متد که دارای یک هدف یکسان هستند تعریف شده‌اند و آن اضافه کردن عدد 10 به مقداری است که به آنها ارسال می‌شود. اولین متد (خطوط 6-2) دارای یک پارامتر است که نیاز به یک مقدار آرگومان دارد. وقتی که متد را صدا می‌زنیم و آرگومانی به آن اختصاص می‌دهیم (خط 21)، کپی آرگومان به پارامتر متد ارسال می‌شود. بنابراین مقدار اصلی متغیر خارج از متد ($num) هیچ ارتباطی به پارامتر متد ندارد. سپس مقدار 10 را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می‌کنیم (خطوط 5 و 4). برای اثبات اینکه متغیر $num هیچ تغییری نکرده است مقدار آن را یکبار دیگر چاپ کرده و مشاهده می‌کنیم که تغییری نکرده است (خط 22). دومین متد (خطوط 12-8) نیاز به یک مقدار با ارجاع دارد. در این حالت به جای اینکه یک کپی از مقدار به عنوان آرگومان به ان ارسال شود آدرس متغیر به آن ارسال می‌شود. حال پارامتر به مقدار اصلی متغیر که زمان فراخوانی متد به آن ارسال می‌شود دسترسی دارد. وقتی که ما مقدار متغیر پارامتری که شامل آدرس متغیر اصلی است را تغییر می‌دهیم (خط 10) در واقع مقدار متغیر اصلی در خارج از متد را تغییر داده‌ایم. در نهایت مقدار اصلی متغیر را وقتی که از متد خارج شدیم را نمایش می‌دهیم و مشاهده می‌شود که مقدار آن واقعاً تغییر کرده است.

توضیح تکمیلی: (ارسال پارامتر به صورت ارجاع و مقدار) به صورت پیش فرض پارامترهایی که به توابع ارسال می شوند طوری هستند که در صورتیکه در تابع تغییر کنند مقدار اصلی آنها تغییری نخواهد کرد و به همان صورت باقی خواهند ماند. اما اگر در تعریف تابع قبل از اسم متغیر از علامت **&** استفاده شود این ویژگی تغییر می کند و یعنی با تغییر یک متغیر در درون تابع، اصل متغیر هم تغییر خواهد کرد. به مثال زیر توجه بفرمایید:

```
<?php
 function changeit(&$string)
 {
echo "String is: " . $string . '<br />';
$string="Learning PHP";
echo "String Changed to: " . $string . '<br />';
 }
$str="PLUS";
 changeit($str);
echo $str;
 ?>
```

خروجی کد بالا:

```
String is: PLUS
 String Changed to: Learning PHP
 Learning PHP
```

محدوده متغیر

متغیرها در PHP دارای محدوده هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متد تعریف می‌شود فقط در داخل بدنه متد قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند:

```php
<?php

    function firstLocalVariable()
    {
        $number = 10;
        echo $number;
    }

    function secondLocalVariable()
    {
        $number = 5;
        echo $number;
    }

    firstLocalVariable ();
    echo '<br/>';
    secondLocalVariable ();

?>
```

```
10
5
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم (خطوط 5 و 11) که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل متد firstLocalVariable() هیچ ارتباطی به متغیر داخل متد secondLocalVariable() ندارد. وقتی به مبحث کلاس‌ها رسیدیم در این باره بیشتر توضیح خواهیم داد. php دارای چهار محدوده است:

- متغیرهای محلی (local)
- متغیرهای سراسری (global)
- متغیرهای ایستا (static)

متغیرهای محلی

متغیرهایی که داخل توابع تعریف می‌شوند محلی هستند و فقط داخل همان تابع قابل استفاده‌اند. به مثال زیر توجه کنید:

```php
<?php

    function LocalVariable()
    {
        $number = 10;
        echo $number;
    }

    LocalVariable ();
    echo $number;

?>
```

```
10
Notice: Undefined variable: number in C:\wamp\www\test.php on line 10
```

همانطور که مشاهده می‌کنید با فراخوانی متد در خط 9 مقدار متغیر number چاپ می‌شود ولی در خط 10 که سعی در چاپ مقدار این متغیر داریم با پیغام خطا مواجه می‌شویم چون طول عمر این متغیر تا زمانی است که تابع به پایان نرسیده است. با پایان تابع متغیر و مقدار آن هم از بین می‌رود در نتیجه در خارج از تابع نمی‌توان مقدار آن را چاپ کرد.

متغیرهای سراسری

متغیرهایی که در بیرون تابع تعریف می‌شوند از نوع سراسری هستند. به مثال زیر توجه کنید:

```php
<?php

    $firstNumber  = 10;
    $secondNumber = 5;
    $Sum;

    function GlobalVariable()
    {
        global $firstNumber, $secondNumber, $Sum;
        $Sum = $firstNumber + $secondNumber;
    }

    GlobalVariable ();
    echo $Sum;

?>
```

```
15
```

متغیرهای firstNumber و secondNember و Sum در بیرون تابع تعریف شده‌اند و از نوع سراسری هستند، در داخل تابع اگر بخواهیم به مقدار آنها دسترسی پیدا کنیم باید ابتدا با کلمه کلیدی global در تابع تعریف کنیم سپس از آن استفاده نماییم. روش دیگر برای دسترسی به متغیرهای سراسری استفاده از آرایه فوق سراسری $GLOBALS است. یعنی کد بالا را به صورت زیر هم می‌توان نوشت:

```php
<?php

    $firstNumber  = 10;
    $secondNumber = 5;
    $Sum;

    function GlobalVariable()
    {
        $GLOBALS["Sum"] = $GLOBALS["firstNumber"] + $GLOBALS["secondNumber"];
    }

    GlobalVariable ();
    echo $Sum;

?>
```

متغیرهای ایستا

با اتمام اجرای تابع تمام متغیرها و آن تابع از بین می‌شوند، به غیر از متغیرهایی که بصورت static تعریف شده باشند، به مثال زیر توجه کنید:

```php
<?php

    function StaticVariable()
    {
        static $firstNumber  = 10;
        echo $firstNumber . '<br/>';
        $firstNumber ++;
    }

    StaticVariable();
    StaticVariable();
    StaticVariable();

?>
```

```
10
11
12
```

همانطور که در کد بالا مشاهده می‌کنید هر بار که تابع فراخوانی می‌شود، متغیر firstNumber که به صورت static تعریف شده، مقدار قبلی خود را حفظ می‌کند. بنابراین با هر بار فراخوانی، مقداری آن به اضافه 1 شده و ذخیره می‌گردد. در خطوط 3-8 یک متد و در داخل آن یک متغیر از نوع ایستا (static) تعریف شده است (خط 5). در خطوط 10-12 سه بار متد را فراخوانی کرده‌ایم. در فراخوانی اول مقدار 10 چاپ می‌شود. در خط 7 یک واحد به این متغیر اضافه می‌شود و این مقدار در فراخوانی دوم چاپ می‌شود (مقدار 11). در فراخوانی سوم هم یک واحد به مقدار قبلی اضافه شده (11+1) و این مقدار یعنی 12 چاپ می‌شود.

بازگشت (Recursion)

بازگشت فرایندی است که در آن متد مدام خود را فراخوانی می‌کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه نویسی است و تسلط به آن کار راحتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای بی نهایت بار، متد، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت (n!) شامل حاصل ضرب همه اعداد مثبت صحیح کوچک‌تر یا مساوی آن می‌باشد. به فاکتوریل عدد 5 توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

بنابراین برای ساخت یک متد بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بازگشت، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچک‌ترین عدد صحیح مثبت 1 است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

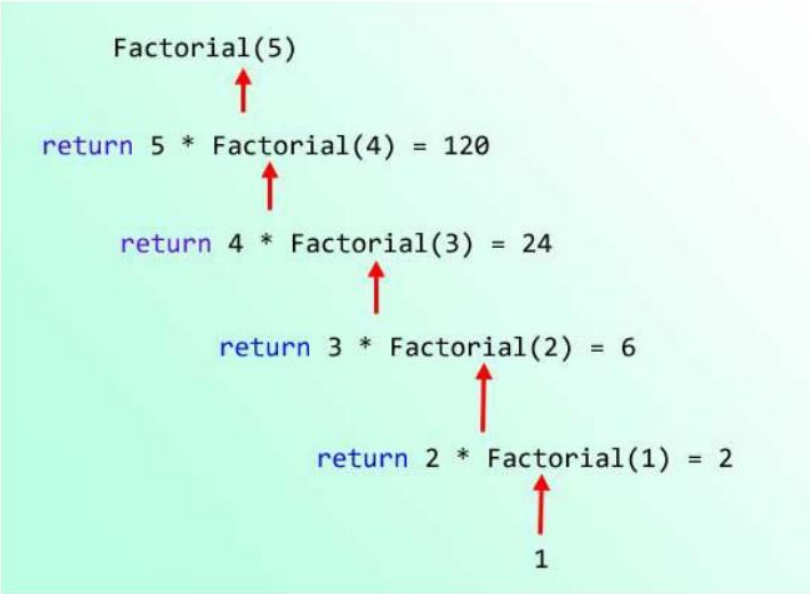
```php
<?php
    function Factorial($number)
    {
        if ($number == 1)
            return 1;

            return $number * Factorial($number - 1);
    }

    echo Factorial(5);
?>
```

```
120
```

متد مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. متد یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل متد یک دستور if می‌نویسیم و در خط 4 می گوییم که اگر آرگومان ارسال شده برابر 1 باشد سپس مقدار 1 را برگردان، در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود. در خط 7 مقدار جاری متغیر number در عددی یک واحد کمتر از خودش (number - 1) ضرب می‌شود. در این خط متد Factorial خود را فراخوانی می‌کند و آرگومان آن در این خط همان number - 1 است. مثلاً اگر مقدار جاری number عدد 10 باشد یعنی اگر ما بخواهیم فاکتوریل عدد 10 را به دست بیاوریم آرگومان متد Factorial در اولین ضرب 9 خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد 1 برابر نشود. شکل زیر فاکتوریل عدد 5 را نشان می‌دهد.



کد بالا را به وسیله یک حلقه for نیز می‌توان نوشت:

```php
<?php
    $factorial = 1;

    for ( $counter = 5; $counter >= 1; $counter-- )
        $factorial *= $counter;

    echo $factorial;
?>
```

```
120
```

این کد از کد معادل بازگشتی آن آسان‌تر است. از بازگشت در زمینه‌های خاصی در علوم کامپیوتر استفاده می‌شود. استفاده از بازگشت حافظه زیادی اشغال می‌کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

سربارگذاری متدها

در درس‌های قبل با چگونگی ایجاد متدها آشنا شدید. در این درس می‌خواهیم شما را با یک مفهوم دیگر درباره متدها به نام سربارگذاری متدها (Method Overloading) آشنا کنیم. در PHP تعریف دو متد با نام یکسان که دارای تعداد پارامترهای متفاوتی باشند امکان پذیر نیست. به مثال زیر توجه کنید:

```php
<?php

    function ShowMessage()
    {
        echo 'Hello World !';
    }

    function ShowMessage($string)
    {
        echo 'Hello '.$string;
    }

    ShowMessage();

?>
```

```
(!) Fatal error: Cannot redeclare Person::showMessage() in C:\wamp\www\Tuts\index.php on line 11
```

با اجرای کد بالا با خطا مواجه می‌شوید، چون PHP نمی‌داند که شما کدام متد را فراخوانی کرده‌اید. در PHP توابعی وجود دارند که توسط توسعه دهندگان این زبان برای مقاصد خاصی تعریف شده‌اند و همراه با نصب PHP به صورت توکار وجود دارند. با این توابع، توابع از پیش تعریف شده (Predefined functions) می گویند. مثلاً از برخی از این توابع برای به دست آوردن طول یک رشته، به دست آوردن تعداد عناصر موجود در یک آرایه، کار با تاریخ و ساعت، کار با پوشه‌ها و فایل‌ها ... استفاده می‌شود. قدرت PHP در همین توابع از پیش تعریف شده است و تعداد آنها در هر نسخه جدید از PHP تغییر کرده و بیشتر می‌شود. یکی از این متدها، متد func_get_args() است، که می‌توان با استفاده از آن یک متد با تعداد پارامترهای متفاوت ایجاد کرد:

```php
<?php
    function showMessage()
```

```
    {
        $string = func_get_args();
        foreach($string as $str)
        {
            echo $str;
        }
    }

    showMessage('1');
    echo '<br/>';
    showMessage('1','2');
    echo '<br/>';
    showMessage('1','2', '3');
?>
```

```
1
12
123
```

همانطور که در کد بالا مشاهده می‌کنید با استفاده از این متد توانستیم در هربار فراخوانی متد ShowMessage() تعداد پارامترهای متفاوتی به آن ارسال کنیم.

نکته: (ارسال پارامتر به صورت ارجاع و مقدار) به صورت پیش فرض پارامترهایی که به توابع ارسال می شوند طوری هستند که در صورتیکه در تابع تغییر کنند مقدار اصلی آنها تغییری نخواهد کرد و به همان صورت باقی خواهند ماند. اما اگر در تعریف تابع قبل از اسم متغیر از علامت **&** استفاده شود این ویژگی تغییر می کند و یعنی با تغییر یک متغیر در درون تابع، اصل متغیر هم تغییر خواهد کرد. به مثال زیر توجه بفرمایید:

```
<?php
 function changeit(&$string)
 {
echo "String is: " . $string . '<br />';
$string="Learning PHP";
echo "String Changed to: " . $string . '<br />';
 }
$str="PLUS";
 changeit($str);
echo $str;
```

```
?>
```

خروجی کد بالا:

```
String is: PLUS
 String Changed to: Learning PHP
 Learning PHP
```

مقدمه‌ای کوتاه بر برنامه نویسی شی گرا (OOP)

برنامه نویسی شیء گرا (OOP) شامل تعریف کلاس‌ها و ساخت اشیاء مانند ساخت اشیاء در دنیای واقعی است. برای مثال یک ماشین را در نظر بگیرید. این ماشین دارای خواصی مانند رنگ، سرعت، مدل، سازنده و برخی خواص دیگر است. همچنین دارای رفتارها و حرکاتی مانند شتاب و پیچش به چپ و راست و ترمز است. اشیاء در PHP تقلیدی از یک شیء مانند ماشین در دنیای واقعی هستند. برنامه نویسی شیء گرا با استفاده از کدهای دسته بندی شده کلاس‌ها و اشیاء را بیشتر قابل کنترل می‌کند. در ابتدا ما نیاز به تعریف یک کلاس برای ایجاد اشیاءمان داریم. شیء در برنامه نویسی شیء گراء از روی کلاسی که شما تعریف کرده‌اید ایجاد می‌شود. برای مثال نقشه ساختمان شما یک کلاس است که ساختمان از روی آن ساخته شده است. کلاس شامل خواص یک ساختمان مانند مساحت، بلندی و مواد مورد استفاده در ساخت خانه می‌باشد. در دنیای واقعی ساختم آنها نیز بر اساس یک نقشه (کلاس) پایه گذاری (تعریف) شده‌اند. برنامه نویسی شیء گرا یک روش جدید در برنامه نویسی است که بوسیله برنامه نویسان مورد استفاده قرار می‌گیرد و به آنها کمک می‌کند که برنامه‌هایی با قابلیت استفاده مجدد، خوانا و راحت طراحی کنند. PHP نیز یک برنامه شیء گراست. در درس بعد به شما نحوه تعریف کلاس و استفاده از اشیاء آموزش داده خواهد شد. همچنین شما با مفهوم وراثت که از مباحث مهم در برنامه نویسی شیء گرا است در آینده آشنا می‌شوید.

کلاس

کلاس به شما اجازه می‌دهد یک نوع داده‌ای که توسط کاربر تعریف می‌شود و شامل متغیرها (فیلدها) و خواص (properties) و متدها است را ایجاد کنید. کلاس در حکم یک نقشه برای یک شیء می‌باشد. شیء یک چیز واقعی است که از ساختار، خواص و یا رفتارهای کلاس پیروی می‌کند. وقتی یک شیء می‌سازید یعنی اینکه یک نمونه از کلاس ساخته‌اید (در درس ممکن است از کلمات شیء و نمونه به جای هم استفاده شود). برای تعریف یک کلاس از کلمه کلیدی class استفاده شود:

```
class ClassName
{
```

```
    Variable1;
    Variable2;
    ...
    VariableN;

    method1;
    method2;
    ...
    methodN;
}
```

این کلمه کلیدی را قبل از نامی که برای کلاسمان انتخاب می‌کنیم می‌نویسیم. در نامگذاری کلاس‌ها هم از روش نامگذاری Pascal استفاده می‌کنیم. در بدنه کلاس متغیرها و متدهای آن قرار داده می‌شوند. متغیرها اعضای داده‌ای خصوصی هستند که کلاس از آنها برای رفتارها و ذخیره مقادیر خاصیت‌هایش (property) استفاده می‌کند. متدها رفتارها یا کارهایی هستند که یک کلاس می‌تواند انجام دهد. در زیر نحوه تعریف و استفاده از یک کلاس ساده به نام person نشان داده شده است:

```php
<?php

    class Person
    {
        public $name;
        public $age;
        public $height;

        public function TellInformation()
        {
            echo 'Name:   ' . $this -> name    .'<br/>';
            echo 'Age:    ' . $this -> age     .'<br/>';
            echo 'Height: ' . $this -> height;
        }
    }


    $person1 = new Person();
    $person2 = new Person();

    $person1 -> name   = 'Jack';
    $person1 -> age    = 21;
    $person1 -> height = 180;
    $person1 -> TellInformation ();

    echo "<br/><br/>";//Separator

    $person2 -> name   = 'Mike';
    $person2 -> age    = 23;
    $person2 -> height = 158;
    $person2 -> TellInformation ();

?>
```

```
Name: Jack
Age: 21
Height: 160

Name: Mike
Age: 23
Height: 158
```

سازنده

سازنده‌ها متدهای خاصی هستند که وجود آنها برای ساخت اشیا لازم است. آن‌ها به شما اجازه می‌دهند که متغیرهای کلاس را مقداردهی اولیه کنید و کدهایی که را که می‌خواهید هنگام ایجاد یک شیء اجرا شوند را به برنامه اضافه کنید. اگر از هیچ سازنده‌ای در کلاستان استفاده نکنید، PHP از سازنده پیشفرض که یک متد بدون پارامتر است استفاده می‌کند. در مثال زیر یک کلاس که شامل سازنده پیشفرض (خطوط 9-12) است را مشاهده می‌کنید:

```php
<?php

    class Person
    {
        public $name;
        public $age;
        public $height;

        public function Person()
        {

        }
    }

    $person1 = new Person();
    var_dump ($person1);

?>
```

```
object(Person)[1]
  public 'name' => null
  public 'age' => null
  public 'height' => null
```

می‌توانیم این سازنده را هم تعریف نکنیم چون PHP به طور خودکار آن را ایجاد می‌کند. همانطور که در خط 15 مشاهده می‌کنید ما یک شیء یا یک نمونه از کلاس ایجاد (در درس بعد بیشتر توضیح می‌دهیم) و با استفاده از تابع var_dump مقادیر موجود در این شیء را چاپ کرده‌ایم (خط 16). در خروجی مشاهده می‌کنید که سازنده پیشفرض به هر سه متغیر مقدار null را اختصاص داده است. بهتر است که با استفاده از سازنده مقدار پیشفرض به متغیرها اختصاص دهیم. مثلاً فردی که به دنیا می‌آید نام (name) ندارد ولی سن (age) و قد (height) دارد. پس می‌توانیم به صورت زیر، این مقادیر را به متغیرها با استفاده از سازنده پیشفرض اختصاص دهیم:

```php
<?php

    class Person
    {
        public $name;
        public $age;
        public $height;

        public function Person()
        {
            $this -> name   ='';
            $this -> age    = 9;
            $this -> height = 30;
        }

        public function TellInformation()
        {
            echo 'Name:   ' . $this -> name   .'<br/>';
            echo 'Age:    ' . $this -> age    .' Month'.'<br/>';
            echo 'Height: ' . $this -> height .' cm';
        }
    }

    $person1 = new Person();
    $person1 -> TellInformation ();

?>
```

```
Name:
Age: 9 Month
Height: 30 cm
```

در خطوط 13-11 کد بالا با استفاده از سازنده پیشفرض مقادیری را به فیلدهای خطوط 7-5 اختصاص داده‌ایم. به این نکته توجه کنید که هنگام اختصاص مقدار به فیلدها فقط نام آنها را بدون علامت $ بنویسید. برای نمایش مقادیر این فیلدها هم یک متد در خطوط 21-16 تعریف کرده‌ایم. در نهایت در خط 24 یک نمونه از کلاس Person ایجاد و با فراخوانی متد ()TellInformation در خط 25، مقادیر فیلدها را چاپ می‌کنیم. به این نکته توجه کنید که سازنده درست شبیه به یک متد است با این تفاوت که

- مقدار برگشتی ندارد.
- نام سازنده باید دقیقاً شبیه نام کلاس باشد.

حال فرض کنید می‌خواهیم سازنده‌ای ایجاد کنیم که بعد از ایجاد یک شیء از کلاس، متغیرهای شیء ایجاد شده را خودمان و با استفاده از سازنده‌ای که تعریف کرده‌ایم مقداردهی کنیم. به کد زیر توجه کنید:

```php
<?php

    class Person
    {
        public $name;
        public $age;
        public $height;

        public function Person($n, $a, $h)
        {
            $this -> name   = $n;
            $this -> age    = $a;
            $this -> height = $h;
        }

        public function TellInformation()
        {
            echo 'Name:   ' . $this -> name   . '<br/>';
            echo 'Age:    ' . $this -> age    . '<br/>';
            echo 'Height: ' . $this -> height;
        }
    }

    $person1 = new Person("Jack", 21, 160);
    $person1 -> TellInformation();

    echo '<br/><br/>';

    $person2 = new Person("Mike", 32, 158);
    $person2 -> TellInformation();

?>
```

```
Name: Jack
Age: 21

Height: 160

Name: Mike
Age: 32
Height: 158
```

همانطور که مشاهده می‌کنید در مثال بالا سازنده‌ای را سه آرگومان قبول می‌کند به کلاس Person اضافه کرده‌ایم (خطوط 9-14). در خطوط 24 و 29 بعد از ایجاد شیء و در داخل پرانتزها سه مقدار را به سازنده (خط 9) ارسال می‌کنیم و سازنده این مقادیر را به متغیرها (خطوط 5-7) اختصاص می‌دهد.

مخرب

مخرب‌ها نقطه مقابل سازنده‌ها هستند. مخرب‌ها متدهای خاصی هستند که هنگام تخریب یک شیء فراخوانی می‌شوند. اشیاء از حافظه کامپیوتر استفاده می‌کنند و اگر پاک نشوند ممکن است با کمبود حافظه مواجه شوید. می‌توان از مخرب‌ها برای پاک کردن منابعی که در برنامه مورد استفاده قرار نمی‌گیرند استفاده کرد. معمولاً PHP به صورت اتوماتیک از زباله روب (garbage collection) برای پاک کردن حافظه استفاده می‌کند و لازم نیست شما به صورت دستی اشیا را از حافظه پاک کنید. به عنوان مثال وقتی کاربر یک فایل متنی را برای خواندن باز می‌کند و آن را نمی‌بندد، می‌توان عمل بستن فایل را با استفاده از مخرب انجام داد. دستور نوشتن مخرب به صورت زیر است:

```
public function __destruct()
{
    //code to execute;
}
```

برنامه زیر نحوه فراخوانی سازنده و مخرب را نشان می‌دهد:

```
1   <?php
2
3       class Test
4       {
5           public function Test()
6           {
7               echo "Constructor was called." . '<br/>';
8           }
9
10          public function __destruct()
11          {
12              echo "Destructor was called.";
13          }
14      }
15
16      $test = new Test();
17
18  ?>
```

```
Constructor was called.
Destructor was called.
```

سطح دسترسی

سطح دسترسی مشخص می‌کند که متدها یک کلاس یا متغیرها در چه جای برنامه قابل دسترسی هستند. در PHP سه سطح دسترسی وجود دارد:

- public (عمومی)
- private (خصوصی)
- protect (محافظت شده)

در این درس می‌خواهیم به سطح دسترسی private و public نگاهی بیندازیم. سطح دسترسی public زمانی مورد استفاده قرار می‌گیرد که شما بخواهید به یک متد یا متغیر در خارج از کلاس و حتی پروژه دسترسی یابید. به عنوان مثال به کد زیر توجه کنید:

```php
<?php

    class Test
    {
        public  $number1 = 10;
        private $number2 = 20;
    }

    $x = new Test();

    echo $x -> number1;
    echo $x -> number2;

?>
```

```
10
Fatal error: Cannot access private property Test::$number2 in C:\wamp\www\test.php on line 13
```

کپسوله سازی

کپسوله کردن (تلفیق داده‌ها با یکدیگر) یا مخفی کردن اطلاعات فرایندی است که طی آن اطلاعات حساس یک موضوع از دید کاربر مخفی می‌شود و فقط اطلاعاتی که لازم باشد برای او نشان داده می‌شود.

وقتی که یک کلاس تعریف می‌کنیم معمولاً تعدادی فیلد برای ذخیره مقادیر مربوط به شیء نیز تعریف می‌کنیم. برخی از این فیلدها توسط خود کلاس برای عملکرد متدها و برخی دیگر از آنها به عنوان یک متغیر موقت به کار می‌روند. لازم نیست که کاربر به تمام فیلدها یا متدهای کلاس دسترسی داشته باشد. اینکه فیلدها را طوری تعریف کنیم که در خارج از کلاس قابل دسترسی باشند بسیار خطرناک است چون ممکن است کاربر رفتار و نتیجه یک متد را تغییر دهد. به برنامه ساده زیر توجه کنید:

```php
<?php

    class Test
    {
        public $five = 5;

        public function AddFive($number)
        {
            $this -> five += $number;
            return $this -> five;
        }
    }

    $test = new Test;

    $test -> five = 10;
    echo $test -> AddFive(100)

?>
```

```
110
```

متد داخل کلاس Test به نام ()AddFive (خطوط 11-7) دارای هدف ساده‌ای است و آن اضافه کردن مقدار 5 به هر عدد می‌باشد (همانطور که از اسم متد پیداست). در خط 14 یک نمونه از کلاس Test ایجاد کرده‌ایم و مقدار فیلد آن را در خط 16 از 5 به 10 تغییر می‌دهیم (در اصل نباید تغییر کند چون ما از برنامه خواسته‌ایم هر عدد را با 5 جمع کند ولی کاربر به راحتی آن را به 10 تغییر می‌دهد). همچنین متد ()AddFive را در خط 17 فراخوانی و مقدار 100 را به آن ارسال می‌کنیم. مشاهده می‌کنید که قابلیت متد ()AddFive به خوبی تغییر می‌کند و شما نتیجه متفاوتی مشاهده می‌کنید. اینجاست که اهمیت کپسوله سازی مشخص می‌شود. اینکه ما در درس‌های قبلی فیلدها را به صورت public تعریف کردیم و به کاربر اجازه دادیم که در خارج از کلاس به آنها دسترسی داشته باشد کار اشتباهی بود. فیلدها باید همیشه به صورت private تعریف شوند.

خواص

property (خصوصیت) استانداردی برای دسترسی به متغیرهایی با سطح دسترسی private در داخل یک کلاس می‌باشد. هر property دارای دو بخش می‌باشد، یک بخش جهت مقدار دهی (بلوک set) و یک بخش برای دسترسی به مقدار (بلوک get) یک داده private می‌باشد. property در مثال زیر نحوه تعریف و استفاده از property آمده است:

```php
<?php

    class Person
    {
        private $name;
```

```php
        public function setName($myName)
        {
            $this->name = $myName;
        }

        public function getName()
        {
            return $this->name;
        }
    }

    $person1 = new Person();

         $person1 -> setName('Jack');
    echo $person1 -> getName();

?>
```

```
Jack
```

وراثت

وراثت به یک کلاس اجازه می‌دهد که خصوصیات یا متدهایی را از کلاس دیگر به ارث برد. وراثت مانند رابطه پدر و پسری می‌ماند به طوریکه فرزند خصوصیاتی از قبیل قیافه و رفتار را از پدر خود به ارث برده باشد.

- کلاس پایه یا کلاس والد کلاسی است که بقیه کلاس‌ها از آن ارث می‌برند.
- کلاس مشتق یا کلاس فرزند کلاسی است که از کلاس پایه ارث بری می‌کند.

همه متد و خصوصیات کلاس پایه می‌توانند در کلاس مشتق مورد استفاده قرار بگیرند به استثنای اعضا و متدهای با سطح دسترسی private.

مفهوم اصلی وراثت در مثال زیر نشان داده شده است:

```php
<?php

    class classParent
    {
```

```php
5         private function privateMessage()
6         {
7             echo 'This is private Message From Parent Class!';
8         }
9
10        public function publicMessage()
11        {
12            echo 'This is public Message From Parent Class!';
13        }
14    }
15
16    class classChild extends classParent
17    {
18
19    }
20
21    $child = new classChild();
22
23    $child -> publicMessage();
24    $child -> privateMessage ();
25
26 ?>
```

```
This is public Message From Parent Class!
 Fatal error: Call to private method classParent::privateMessage()
```

همانطور که مشاهده می‌کنید در کد بالا دو کلاس تعریف کرده‌ایم. یکی کلاس classParent (خطوط 3-14) که دارای دو متد یکی با سطح دسترسی private و دیگری با سطح دسترسی public است و کلاس دیگر (خطوط 16-19) که در بدنه خود هیچ متد یا متغیری ندارد. نحوه ارث بری یک کلاس به صورت زیر است:

```
classChild extends Parent
```

که در خط 16 مشخص کرده‌ایم که کلاس classChild قرار است از کلاس classParent ارث بری کند:

```
class classChild extends classParent
```

در خط 21 یک نمونه از کلاس فرزند ایجاد می‌کنیم و در خطوط 23 و 24 دو متد کلاس پدر ر فراخوانی می‌کنیم. همانطور که در خروجی مشاهده می‌کنید متدی که دارای سطح دسترسی public است فراخوانی و اجرا شده ولی در فراخوانی متدی با سطح دسترسی private با خطا مواجه می‌شویم. در پایان یادآور می‌شویم که کلاس فرزند (خطوط 16-19) هیچ متد یا متغیری در بدنه خود ندارد ولی چون از کلاس classParent ارث بری کرده است متد با سطح دسترسی public آن را می‌تواند برای خود داشته باشد.

سطح دسترسی Protect

سطح دسترسی protect اجازه می‌دهد که اعضای کلاس، فقط در کلاسهای مشتق شده از کلاس پایه قابل دسترسی باشند. بدیهی است که خود کلاس پایه هم می‌تواند به این اعضا دسترسی داشته باشد. کلاس‌هایی که از کلاس پایه ارث بری نکرده‌اند نمی‌توانند به اعضای با سطح دسترسی protect یابند. در مورد سطوح دسترسی public و private قبلاً توضیح دادیم. در جدول زیر نحوه دسترسی به سه سطح ذکر شده نشان داده شده است:

protected	private	public	قابل دسترسی در
true	true	true	داخل کلاس
false	false	true	خارج از کلاس
true	false	true	کلاس مشتق

مشاهده می‌کنید که public بیشترین سطح دسترسی را داراست. صرف نظر از مکان، اعضای public در هر جا فراخوانی می‌شوند و قابل دسترسی هستند. اعضای private فقط در داخل کلاسی که به آن تعلق دارند قابل دسترسی هستند. کد زیر رفتار اعضای دارای این سه سطح دسترسی را نشان می‌دهد:

```php
<?php

    class classParent
    {
        protected $protectedMember = 10;
        private   $privateMember   = 20;
        public    $publicMember    = 30;
    }

    class classChild extends classParent
    {
        public function __construct ()
        {
            echo $this -> publicMember    .'<br/>';
            echo $this -> protectedMember .'<br/>';
            echo $this -> privateMember;
        }
    }

    $child = new classChild();

?>
```

```
10
2
(!) Notice: Undefined property: classChild::$privateMember
```

همانطور که در خط 16 مشاهده می‌کنید کلاس classChild سعی می‌کند که به عضو private کلاس classParent دسترسی یابد. از آنجاییکه اعضای private در خارج از کلاس قابل دسترسی نیستند، حتی کلاس مشتق در خط 16 نیز ایجاد خطا می‌کند. اگر شما به خط ۱۴ توجه کنید کلاس classChild می‌تواند به عضو protect کلاس classParent دسترسی یابد چون کلاس classChild از کلاس classParent مشتق شده است. حال کد زیر را در نظر بگیرید:

```php
<?php

    class classParent
    {
        public    $publicMember    = 10;
        protected $protectedMember = 20;
        private   $privateMember   = 30;
    }

    $parent = new classParent();

    echo $parent -> publicMember;
    echo $parent -> protectedMember;
    echo $parent -> privateMember;

?>
```

```
10
(!) Fatal error: Cannot access protected property classParent::$protectedMember
```

همانطور که در کد بالا مشاهده می‌کنید از آنجاییکه اعضای protected و private در خارج از کلاس قابل دسترسی نیستند خطوط 13 و 14 ایجاد خطا می‌کنند.

Overriding

متدهای مجازی متدهایی از کلاس پایه هستند که می‌توان به وسیله یک متد از کلاس مشتق آنها را override کرده و به صورت دلخواه پیاده سازی نمود. به عنوان مثال شما متد A را در کلاس A دارید و کلاس B از کلاس A ارث بری می‌کند، در این صورت متد A در کلاس B در دسترس خواهد بود. اما متد A دقیق همان متدی است که از کلاس A به ارث برده شده است. حال اگر بخواهید که این متد رفتار متفاوتی از خود نشان دهد چکار می‌کنید؟ Overriding یا بازنویسی این مشکل را برطرف می‌کند. به تکه کد زیر توجه کنید:

```php
<?php
    class Person
    {
        public function ShowMessage()
        {
            echo 'Message from Parent.';
        }
    }

    class Child extends Person
    {
        public function ShowMessage()
        {
            parent::ShowMessage();
            echo '<br/>ShowMessage method was overrided !' ;
        }
    }

    $myPerson = new Person();
    $myChild = new Child();

    $myPerson -> ShowMessage();
    echo '<br/><br/>';
    $myChild  -> ShowMessage();
?>
```

```
Message from Parent.

Message from Parent.
ShowMessage method was overrided !
```

همانطور که در کد بالا مشاهده می‌کنید یک متد به نام ShowMessage() (خطوط 4-7) در کلاس Person تعریف شده است که یک پیغام چاپ می‌کند. حال می‌خواهیم این متد در کلاس Child علاوه بر این پیغام، پیغام !ShowMessage method was overrided را نیز چاپ کند. برای این کار همانطور که مشاهده می‌کنید همین متد را در خطوط (12-16) و در داخل کلاس Child می‌نویسیم و سپس با استفاده از کلمه کلیدی parent و سپس دو نقطه (::) در خط 14 به PHP اعلام می‌کنیم که قصد استفاده از تمام کدهای بدنه همین متد در کلاس مادر را داریم بعلاوه اینکه در خط بعد از این دستور یعنی خط 15 کدهای اضافی را که قرار است این متد در کلاس فرزند یعنی Child داشته باشد می‌نویسیم. شاید این کار برای متدی به این سادگی زیاد کارا نباشد، اما اگر متد کلاس مادر دارای کدهای زیادی در بدنه خود باشد و شما بخواهید کد دیگری در کلاس فرزند به آن اضافه کنید استفاده از این روش کدنویسی را بهینه و ساده‌تر می‌کند.

اعضای Static

اگر بخواهیم عضو داده‌ای (فیلد) یا خاصیتی ایجاد کنیم که در همه نمونه‌های کلاس قابل دسترسی باشد از کلمه کلیدیstatic استفاده می‌کنیم. کلمه کلیدی static برای اعضای داده‌ای و خاصیت‌هایی به کار می‌رود که می‌خواهند در همه نمونه‌های کلاس تقسیم شوند. وقتی که یک متد یا خاصیت به صورت static تعریف شود، می‌توانید آنها را بدون ساختن نمونه‌ای از شی، فراخوانی کنید. برای فراخوانی یک عضو استاتیک ابتدا نام کلاس سپس علامت دو نقطه (::) و در آخر نام عضو استاتیک را می‌نویسید:

```
Class Name :: Static Member
```

به مثالی در مورد متدها و خاصیت‌های static توجه کنید:

```php
<?php

    class SampleClass
    {
        static $number = 10;

        static function PrintNumber()
        {
            echo self::$number;
        }

        static function ShowStaticMessage()
        {
            echo 'This is a Static Function!';
        }
    }

    SampleClass::PrintNumber ();
    echo '<br/>';
    SampleClass::ShowStaticMessage ();

?>
```

```
This is a Static Function!
10
```

همانگونه که در کد بالا مشاهده می‌کنید یک کلاس (خطوط 3-16) تعریف کرده‌ایم که دارای یک متغیر (خط 5) و دو متد (خطوط 7-10 و خطوط 12-15) از نوع استاتیک می‌باشد. برای دسترسی به متد یا متغیرهای استاتیک به دو صورت عمل می‌کنیم:

اگر بخواهیم به یک متغیر استاتیک در داخل یک متد استاتیک دسترسی داشته باشیم به جای استفاده از کلمه کلیدی this از کلمه کلیدی self و به جای علامت -> از علامت :: استفاده می‌کنیم. مثلاً در خط 9 مثال بالا برای چاپ متغیر number که استاتیک است در داخل متد استاتیک PrintNumber() به روشی که ذکر شد عمل کرده‌ایم.

اما اگر بخواهیم به یکی از اعضای استایک کلاس در خارج از کلاس دسترسی داشته باشیم لازم نیست که از کلاس نمونه ایجاد کنیم و فقط کافیست که نام کلاس را نوشته و بعد از آن علامت دو نقطه و در آخر نام عضو استاتیک را بنویسید مانند خطوط 18 و 20 مثل بالا.

چند ریختی

چند ریختی (Polymorphism) به کلاس‌هایی که در یک سلسله مراتب وراثتی مشابه هستند اجازه تغییر شکل و سازگاری مناسب می‌دهد و همچنین به برنامه نویس این امکان را می‌دهد که به جای ایجاد برنامه‌های خاص، برنامه‌های کلی و عمومی‌تری ایجاد کند. به عنوان مثال در دنیای واقعی همه حیوانات غذا می‌خورند، اما روش‌های غذا خوردن آنها متفاوت است. در یک برنامه برای مثال، یک کلاس به نام Animal ایجاد می‌کنید. بعد از ایجاد این کلاس می‌توانید آن را چند ریخت (تبدیل) به کلاس Bird کنید و متد Fly() را فراخوانی کنید. به مثالی در باره چند ریختی توجه کنید:

```php
<?php

    class Animal
    {
        public function Eat()
        {
            echo 'The animal ate!' .'<br/>';
        }
    }

    class Dog extends Animal
    {
        function Eat()
        {
            echo 'The dog ate!' .'<br/>';
        }
    }


    class Bird extends Animal
    {
        function Eat()
        {
            echo 'The bird ate!' .'<br/>';
        }
```

```php
    }

    class Fish extends Animal
    {
        function Eat()
        {
            echo 'The fish ate!' .'<br/>';
        }
    }


    $myDog    = new Dog();
    $myBird   = new Bird();
    $myFish   = new Fish();
    $myAnimal = new Animal();

    $myAnimal ->Eat();

    $myAnimal = $myDog;
    $myAnimal ->Eat();

    $myAnimal = $myBird;
    $myAnimal ->Eat();

    $myAnimal = $myFish;
    $myAnimal ->Eat();

?>
```

```
The animal ate!
The dog ate!
The bird ate!
The fish ate!
```

همانطور که مشاهده می‌کنید ۴ کلاس مختلف تعریف کرده‌ایم. Animal کلاس پایه است و سه کلاس دیگر از آن مشتق می‌شوند. هر کلاس متد ()Eat مربوط به خود را دارد. نمونه‌ای از هر کلاس ایجاد کرده‌ایم (40-37). حال متد ()Eat را به وسیله نمونه ایجاد شده از کلاس Animal به صورت زیر فراخوانی می‌کنیم:

```php
$myAnimal = new Animal();
$myAnimal ->Eat();
```

در مرحله بعد چندریختی روی می‌دهد. همانطور که در مثال بالا مشاهده می‌کنید شیء Dog را برابر نمونه ایجاد شده از کلاس Animal قرار می‌دهیم (خط 44) و متد ()Eat را بار دیگر فراخوانی می‌کنیم. حال با وجود اینکه ما از نمونه کلاسAnimal استفاده کرده‌ایم ولی متد ()Eat کلاس Dog فراخوانی می‌شود. این به دلیل تأثیر چند ریختی است.

سپس دو شیء دیگر (Bird و Fish) را برابر نمونه ایجاد شده از کلاس Animal قرار می‌دهیم و متد ()Eat مربوط به هر یک را فراخوانی می‌کنیم. (خطوط 51-47) به این نکته توجه کنید که وقتی در مثال بالا اشیاء را برابر نمونه کلاسAnimal قرار می‌دهیم از عمل Cast استفاده نکرده‌ایم چون این کار (cast) وقتی که بخواهیم یک شیء از کلاس مشتق (مثلاً Dog) را در شیی از کلاس پایه (Animal) ذخیره کنیم لازم نیست.

توابع از پیش تعریف شده:

در php بیش از ۵۳۰۰ تابع از پیش تعریف شده وجود دارد. توابع از پیش تعریف شده همانند توابعی که کاربر تعریف می کند اجرا می شود. (در مباحث قبل به تعدادی از این توابع اشاره شده است.)

چند نمونه دیگر از توابع از پیش تعریف شده در PHP:

تست نوع متغیر

برای تشخیص نوع داده‌ای که در یک متغیر ذخیره شده است از تابع gettype() استفاده می‌شود. برای این کار کافیست که نام متغیر را در داخل پرانتز تابع قرار دهید تا نوع آن را به شما نمایش دهد. به مثال زیر توجه کنید:

```php
<?php
    $number = 10.2;
    echo gettype($number);
?>
```

```
double
```

همانطور که در مثال بالا مشاهده می‌کنید خروجی کلمه double هست که نشان دهنده نوع داده‌ای است که در متغیر $number ذخیره شده است. در php توابع دیگری برای تشخیص نوع داده ذخیره شده در متغیر وجود دارد که در جدول زیر لیست آنها آمده است:

تابع	عملکرد
is_int(value)	اگر value از نوع صحیح باشد مقدار 1 یا true را بر می‌گرداند
is_ string (value)	اگر value از نوع رشته باشد مقدار 1 یا true را بر می‌گرداند
is_float(value)	اگر value از نوع اعشار باشد مقدار 1 یا true را بر می‌گرداند
is_ bool (value)	اگر value از نوع boolean باشد مقدار 1 یا true را بر می‌گرداند
is_ array (value)	اگر value از نوع آرایه باشد مقدار 1 یا true را بر می‌گرداند
is_ object (value)	اگر value از نوع object باشد مقدار 1 یا true را بر می‌گرداند
is_ resource (value)	اگر value یک منبع داده باشد مقدار 1 یا true را بر می‌گرداند
is_ null (value)	اگر value از نوع null باشد مقدار 1 یا true را بر می‌گرداند

تغییر نوع متغیر

فرض کنید که می‌خواهید مقدار یک متغیر را به نوع اعشار تغییر دهید. این کار در php توسط تابع settype() انجام می‌شود. به مثال زیر توجه کنید:

```
<?php
    $number = 10;
    settype($number,"double");
?>
```

همانطور که در مثال بالا مشاهده می‌کنید، در داخل پرانتز این تابع ابتدا نام متغیر و سپس در داخل یک جفت کوتیشن نام نوعی که قرار است متغیر به آن تبدیل شود را می‌نویسیم. در مثال بالا $number به نوع double تبدیل شده است. روش دیگری که به آن عمل cast می گویند هم برای انجام این کار وجود دارد. در عمل cast نام نوعی که قرار است متغیر به آن تبدیل شود را در داخل پرانتز و قبل از نام متغیر می‌نویسیم. مثلاً در مثال بالا:

```
<?php
    $number = 10;
    echo gettype($number);

    echo '<br/>';

    $newType = (double)$number;
    echo gettype($newType);
?>
```

```
integer
double
```

جا دادن یک رشته در داخل رشته دیگر

PHP به شما اجازه می‌دهد که با استفاده از متد substr_replace() یک رشته را در داخل رشته دیگر قرار دهید. به عنوان مثال اگر رشته‌ای مانند Hello World! داشته باشید می‌توانید با استفاده از متد ذکر شده کلمه Happy را در وسط آن قرار دهید و رشته جدیدی مانند Hello Happy World! ایجاد کنید. کد زیر نحوه استفاده از متد substr_replace() را نشان می‌دهد:

```
<?php
    $Str1 = "Hello World!";
    $Str2 =" Happy";
    echo substr_replace($Str1, $Str2, 5, 0);
?>
```

```
Hello Happy World!
```

متد substr_replace() چهار آرگومان قبول می‌کند، اولین آرگومان رشته اصلی و دومین آرگومان رشته جدید که قرار است در داخل رشته اصلی قرار بگیرد را نشان می‌دهد. سومین آرگومان اندیس یا مکان انجام جایگذاری رشته جدید است. اندیس اولین کاراکتر یک رشته با صفر و اندیس آخرین کاراکتر یک رشته با یک واحد کمتر از طول رشته نمایش داده می‌شود. در رشته Hello World! می‌خواهیم رشته جدید را درست بعد از اولین فضای خالی یعنی جایی که حرف W قرار دارد جای دهیم. به کادر زیر توجه کنید:

```
H  e  l  l  o     W  o  r  l  d  !
0  1  2  3  4  5  6  7  8  9  10 11
```

اگر توجه کرده باشید رشته Happy در این مکان (بعد از اولین فضای خالی) قرار گرفته است. و هر چیز بعد از آن به جلو رانده و رشته Hello Happy World! ایجاد می‌شود. آرگومان چهارم هم بدین معناست که در کاراکترهای بعد از رشته جایگذاری شده (در اینجا World) هیچ کاراکتری کم و زیاد نشود.

اضافه کردن کاراکتر به سمت چپ یا راست یک رشته با استفاده از متد ()str_pad

با استفاده از متد str_pad() می‌توان کاراکتر یا فضاهای خالی را به سمت چپ یا راست رشته‌ها اضافه کرد. این متد چهار آرگومان می‌گیرد. اولین آرگومان رشته، دومین آرگومان تعداد کاراکترهایی که قرار است به سمت چپ، راست و یا هر دو سمت رشته اضافه شوند، سومین آرگومان نوع کاراکتری که قرار است اضافه شود و چهارمین آرگومان هم سمت اضافه شدن را نشان می‌دهد که می‌تواند یکی از مقادیر STR_PAD_LEFT، STR_PAD_RIGHT و یا STR_PAD_BOTH باشد.

```php
<?php
    $Str1 = "Exapmle";
    echo str_pad($Str1, 10, "*", STR_PAD_LEFT);
?>
```

```
***Example
```

همانطور که در مثال بالا مشاهده کردید عدد 10 ارسال شده باعث اضافه شدن سه علامت * در سمت چپ رشته می‌شود. برای به دست آوردن مقدار این فضاها یا کاراکترهایی که در سمت چپ یا راست یک رشته قرار می‌گیرند می‌توان از فرمول (تعداد کاراکتر یا فضای خالی – طول رشته) استفاده کرد.

مثلاً در مثال بالا طول رشته Example برابر 7 می‌باشد و زمانی که ما عدد 10 را به عنوان آرگومان ارسال می‌کنیم با استفاده از فرمول 3= 7 – 10 می‌توان فهمید که چند فضای خالی در سمت چپ رشته قرار می‌گیرد (3 فضای خالی). اگر مقدار آرگومانی که به متد str_pad() ارسال می‌شود از طول رشته کمتر باشد هیچ تاثیری بر رشته نهایی ندارد. عمل STR_PAD_LEFT هم شبیه STR_PAD_RIGHT است.

مقایسه رشته‌ها

می‌توان رشته‌ها را به روش‌های مختلف با هم مقایسه کرد. به عنوان مثال با استفاده از متد strcmp() می‌توان تست کرد که آیا دو رشته با هم برابرند یا نه:

```php
<?php
    echo strcmp("Hello","Hello");
    echo "<br>";
    echo strcmp("Hello","hELLo");
?>
```

```
0
-1
```

این متد نسبت به بزرگی و کوچکی حروف حساس است و دو رشته را قبول می‌کند و اگر با هم برابر باشند مقدار 0، اگر مقدار اولین رشته از دومین رشته بیشتر باشد مقدار 1 و اگر مقدار اولین رشته از دومین رشته کوچک‌تر باشد مقدار 1- را بر می‌گرداند. در این متد هر کاراکتر در رشته به یونیکد معادل خود تبدیل می‌شود. اولین کاراکتر اولین رشته با اولین کاراکتر دومین رشته مقایسه می‌شود. اگر برابر بودند سپس دومین کاراکترها و به همین ترتیب بقیه کاراکترها با هم مقایسه می‌شوند. با استفاده از متد strcasecmp() می‌توان دو رشته را بدون در نظر گرفتن بزرگی و کوچکی حروف با هم مقایسه کرد:

این متد نسبت به بزرگی و کوچکی حروف حساس است و دو رشته را قبول می‌کند و اگر با هم برابر باشند مقدار 0، اگر مقدار اولین رشته از دومین رشته بیشتر باشد مقدار 1 و اگر مقدار اولین رشته از دومین رشته کوچک‌تر باشد مقدار 1- را بر می‌گرداند. در این متد هر کاراکتر در رشته به یونیکد معادل خود تبدیل می‌شود. اولین کاراکتر اولین رشته با اولین کاراکتر دومین رشته مقایسه می‌شود. اگر برابر بودند سپس دومین کاراکترها و به همین ترتیب بقیه کاراکترها با هم مقایسه می‌شوند. با استفاده از متد strcasecmp() می‌توان دو رشته را بدون در نظر گرفتن بزرگی و کوچکی حروف با هم مقایسه کرد:

```php
<?php
    echo strcasecmp("Hello","Hello");
    echo "<br>";
    echo strcasecmp("Hello","hELLo");
?>
```

```
0
0
```

جدا کردن رشته‌ها

اگر بخواهید یک رشته را به چند رشته تکه تکه کنید می‌توانید از متد explode() استفاده نمایید. اجازه دهید نگاهی به سربارگذاری های مختلف این متد بیندازیم. متد explode() آرایه‌ای از رشته‌ها را بر می‌گرداند که هر عنصر از این آرایه شامل یک زیر رشته است. اولین سربارگذاری این متد یک رشته را قبول می‌کند و بر اساس آنها تشخیص می‌دهد که رشته باید در چه جایی به قسمت‌های مختلف تقسیم شود:

```php
<?php
    $string = "The quick brown fox jumps over the lazy dog.";
    $substrings = explode(' ',$string);

    foreach ($substrings as $word)
    {
        echo $word .'<br/>';
    }
?>
```

```
The
quick
brown
fox
jumps
over
the
lazy
dog.
```

جستجو کردن در رشته‌ها

جستجوي رشته‌ها به وسيله متدهاي PHP بسيار راحت است. اجازه بدهيد كه نگاهي به متدهاي مختلفي كه محل وقوع يک رشته خاص را پيدا می‌کنند بيندازيم. متد strpos() اولين محل وقوع يک رشته خاص را در رشته ديگر نشان می‌دهد. اگر رشته مورد نظر پيدا نشود متدهاي فوق مقدار null را بر می‌گردانند. به مثالي در مورد متد strpos() توجه کنيد:

```php
<?php
    $string = "The quick brown fox jumps over the lazy dog.";

    $index = strpos($string,"quick");

    echo $string .'<br/>';
    echo "quick was found at position ".$index;
?>
```

```
The quick brown fox jumps over the lazy dog.
quick was found at position 4
```

انواع خطاها در PHP

هنگام کدنویسی در PHP ممکن است با خطاهایی مواجه شویم. خطاها ممکن است بر اثر اشتباه تایپی و یا اشتباه در منطق برنامه به وجود بیایند. در جدول زیر لیست خطاهایی که ممکن است در هنگام برنامه نویسی PHP به وجود بیایند آمده است:

خطا	توضیح
Fatal error	این نوع از خطاها که به خطاهای بحرانی هم معروف هستند باعث می‌شوند که ادامه کار برنامه با مشکل مواجه شده و برنامه اجرا نشود.
Parse error	این نوع خطاها فقط در زمان اجرای برنامه تولید می‌شوند و اسم دیگر این نوع خطاها Syntax Error می‌باشد. فراموش کردن یک سمیکالن و یا خطای تایپی باعث به وجود آمدن این خطاها می‌شود. این خطاها هم از اجرای برنامه بقیه برنامه جلوگیری می‌کنند.
Warning	این نوع خطاها توسط PHP به کاربر نمایش داده می‌شوند، اما مانع از اجرای بقیه برنامه نمی‌شوند. مثلاً وقتی یک عدد رو بر صفر تقسیم می‌کنیم یک Warning دریافت می‌کنیم.
Notices	این نوع هم مثل انواع خطاهای قبلی می‌تواند خودکار توسط خود PHP و یا با استفاده از تابع trigger_error که توسط کاربر ایجاد شده است درست شوند. این نوع خطا بیشتر هشداری است.

مثالی از Fatal Error

```php
<?php
    Method();
    echo "Save Successfully!"
?>
```

مثالی از Parse error:

```php
<?php
    echo "Save Successfully!"
    echo "PHP Learning";
?>
```

مثالی ازwarning:

```php
<?php
    $x = 200;
    $y = 0;
    $z = $x/$y;
    echo "RESULT: ". $z;
?>
```

مثالی از notices:

```
<?php
    $x += 1;
    echo "RESULT: ". $x;
?>
```

مدیریت استثناءها و خطایابی

بهترین برنامه نویسان در هنگام برنامه نویسی با خطاها و باگ‌ها در برنامه‌شان مواجه می‌شوند. درصد زیادی از برنامه‌ها هنگام تست برنامه با خطا مواجه می‌شوند. بهتر است برای از بین بردن یا به حداقل رساندن این خطاها، به کاربر در مورد دلایل به وجود آمدن آنها اخطار داده شود. خوشبختانه PHP دارای یک مکانیسم داخلی و راه‌هایی برای نشان دادن دلیل وقوع خطا در هنگام اجرای برنامه است. استثناءها توسط برنامه به وجود می‌آیند و شما لازم است که آنها را اداره کنید. به عنوان مثال در دنیای کامپیوتر یک عدد صحیح هرگز نمی‌تواند بر صفر تقسیم شود. اگر بخواهید این کار را انجام دهید (یک عدد صحیح را بر صفر تقسیم کنید)، با خطا مواجه می‌شوید. اگر یک برنامه در PHP با چنین خطایی مواجه شود پیغام خطای "DivideByZeroException" نشان داده می‌شود که بدین معنا است که عدد را نمی‌توان بر صفر تقسیم کرد.

باگ (Bug) اصطلاحاً خطا یا کدی است که رفتارهای ناخواسته‌ای در برنامه ایجاد می‌کند. خطایابی فرایند برطرف کردن باگ‌ها است، بدین معنی که خطاها را از برنامه پاک کنیم. PHP دارای ابزارهایی برای خطایابی هستند، که خطاها را یافته و به شما اجازه می‌دهند آنها را برطرف کنید. در درس‌های آینده خواهید آموخت که چگونه از این ابزارهای کارآمد جهت برطرف کردن باگ‌ها استفاده کنید. قبل از اینکه برنامه را به پایان برسانید لازم است که برنامه‌تان را اشکال زدایی کنید.

استثناءهای اداره نشده

استثناءهای اداره نشده، استثناءهایی هستند که به درستی توسط برنامه اداره نشده‌اند و باعث می‌شوند که برنامه به پایان برسد. در اینجا می‌خواهیم به شما نشان دهیم که وقتی یک برنامه در زمان اجرا با یک استثناء مواجه می‌شود و آن را اداره نمی‌کند چه اتفاقی می‌افتد. در آینده خواهید دید که یک استثناء چگونه به صورت بالقوه باعث نابودی جریان و اجرای برنامه شما می‌شود. به کدهای زیر توجه کنید:

```
<?php
```

```
    $five = 5;
    $zero = 0;

    $result = $five / $zero ;

    echo $result;

?>
```

```
( ! ) Warning: Division by zero in ...
```

همانطور که در کد بالا مشاهده می‌کنید، خط 6 دلیل به وجود آمدن خطاست، چون تقسیم عدد بر صفر غیر ممکن است. این خطا باعث می‌شود که کدهای بعد از ان اجرا نشوند.

دستورات try و catch

می‌توان خطاها را با استفاده از دستور try…catch اداره کرد. بدین صورت که کدی را که احتمال می‌دهید ایجاد خطا کند در داخل بلوک try قرار می‌دهید. بلوک catch هم شامل کدهایی است که وقتی اجرا می‌شوند که برنامه با خطا مواجه شود. تعریف ساده‌ی این دو بلوک به این صورت است که بلوک try سعی می‌کند که دستورات را اجرا کند و اگر در بین دستورات خطایی وجود داشته باشد برنامه دستورات مربوط به بخش catch را انجام می‌دهد. برنامه زیر نحوه استفاده از دستور try…catch را نمایش می‌دهد:

```
1   <?php
2
3       $five = 5;
4       $zero = 0;
5
6       try
7       {
8           if($zero > 0  ||  $zero < 0)
9           {
10              $result = $five / $zero ;  //Error
11              echo $result;
12          }
13          else
14          {
15              throw new Exception('An attempt to divide by 0 was detected.');
16          }
17      }
18      catch(Exception $ex)
19      {
20          echo 'Error : '.$ex -> getMessage();
21      }
22
23  ?>
```

در کد بالا ما قصد داریم خطاهای احتمالی را که در عملیات تقسیم ممکن است به وجود آید را اداره کنیم. در ریاضیات عمل تقسیم عدد بر صفر ممکن نیست. در خطوط 3 و 4 دو متغیر تعریف کرده‌ایم که مقدار یکی از آنها 5 و دیگری 0 است. چون ممکن اس که عمل تقسیم بر صفر اتفاق افتد پس در قسمت try در خط 8 با استفاده از دستور if چک می‌کنیم که اگر مقدار متغیر $zero بزرگ‌تر و یا کوچک‌تر از 0 باشد عملیات تقسیم انجام و نتیجه نمایش داده شود (خطوط 10 و 11) در غیر اینصورت یک استثناء ایجاد کند. از آنجاییکه مقدار متغیر $zero عدد 0 است، یک استثناء رخ می‌دهد. این استثناء را در خط 15 و در قسمت else می‌نویسیم و یک پیغام دلخواه و قابل فهم برای نمایش به کاربر می‌نویسیم.

برای چاپ پیغام خطا در دستور catch یک نمونه از کلاس Exception ایجاد کرده (خط 18) و با فراخوانی متد ()getMessage (خط 20) این کلاس آن را چاپ و به کاربر نمایش می‌دهیم.

ایجاد یک استثناء توسط کاربر

در PHP می‌توان یک استثناء سفارشی ایجاد کرد. استثناء سفارشی استثنایی است که توسط کاربر تعریف می‌شود و باید از کلاس پایه Exception ارث بری کند. به کد زیر توجه کنید:

```php
<?php
    class DivideByZeroException extends Exception
    {
        public function __construct()
        {
            $message = "Attempted to divide by zero...";
            parent::__construct($message);
        }
    }


    $firstNumber = 10;
    $secondNumber = 0;

    try
    {
        if($secondNumber == 0 )
        {
            throw new DivideByZeroException();
        }
        else
        {
            $result = $firstNumber / $secondNumber;
        }
    }
    catch (DivideByZeroException $error)
    {
        echo $error->getMessage();
    }
?>
```

```
Attempted to divide by zero...
```

همانطور که در کد بالا و در خط 2 مشاهده می‌کنید، یک کلاس جداگانه که از کلاس پایه Exception ارث می‌برد ایجاد کرده و نام آن را DivideByZeroException گذاشته‌ایم. هدف از ایجاد این کلاس هم نمایش پیغام خطایی به کاربر مبنی بر عدم به عنوان یک قرارداد باید به آخر نام کلاس‌های استثنایی که توسط کاربر تعریف می‌شوند کلمه Exception اضافه می‌شود. برای اینکه یک پیغام خطای سفارشی هم ایجاد کنیم در خط 7 سازنده کلاس Exception را فراخوانی می‌کنیم. این سازنده دو پارامتر می‌گیرد که اولین پارامتر آن یک رشته است که می‌تواند همان پیغام خطایی باشد که ما دوست داریم به کاربر نمایش داده شود. حال به نحوه استفاده از این کلاس می‌پردازیم. در خطوط 12 و 13 دو متغیر تعریف کرده‌ایم که مقدار یکی از آنها 10 و دیگری 0 است. می‌خواهیم متغیر firstNumber را بر secondNumber تقسیم کنیم. یعنی تقسیم یک عدد بر صفر. در داخل بلوک try، تست می‌کنیم که اگر مقدار متغیر secondNumber برابر 0 است یک استثناء تولید شود (خط 19)

و چون این شرط برقرار است در نتیجه یک استثناء رخ می‌دهد، چون تقسیم عدد بر صفر بی معنی است. پیغام مربوط به استثناء را با استفاده از متد ()getMessage که کلاس DivideByZeroException از کلاس Exception به ارث برده است در خط 28 نمایش می‌دهیم.