**PHP Sillasbusu**

**Mövzu 1: PHP Əsasları**

- PHP-in tərəfdaşlığı və hansı proqram dilləri ilə birlikdə işləyə bilərliyi

- PHP quraşdırılması və konfiqurasiyası

- PHP-də sintaksis və müstəqil dəyişənlər

**Mövzu 2: Dizayn prinsipləri və məntiqi operatorlar**

- Müəyyən edicilər, şərt operatorları və dövri operatorlar

- Dizayn prinsipləri və kodun təkmilləşdirilməsi

- Funksiya və siniflərin düzgün tətbiqi

**Mövzu 3: Form işləmə**

- İstifadəçi dəyərlərini almaq və doğrulamaq üçün formun tərtib edilməsi

- Form dəyərlərini işləmək üçün PHP-in istifadəsi

- Xətaların idarə edilməsi və mesajlarının göstərilməsi

Mövzu 4: Verilənlər bazası əməliyyatları

- MySQL və ya digər verilənlər bazası sistemləri ilə əməliyyatlar

- SQL istifadə edərək verilənlər bazasına sorğu göndərilməsi

- Verilənlər bazası əməliyyatlarının təhlili və təhlil edilməsi

\*\*Mövzu 5: Şablon motorları və məzmun idarəetmə sistemi\*\*

- PHP-ə aid şablon motorları və istifadəsi

- Məzmun idarəetmə sistemlərinin (CMS) əsasları

- CMS-in əməliyyatları və konfiqurasiyası

\*\*Mövzu 6: Sessiyalar və Cookie-lər\*\*

- Sessiyaların və cookie-lərin əsasları

- PHP-də sessiyaların idarə edilməsi və yaradılması

- Sessiyaların təhlili və istifadəsi

\*\*Mövzu 7: Təhlil və serializasiya\*\*

- XML və JSON təhlili

- PHP-də seriyalaşdırma və dəserializasiya

\*\*Mövzu 8: PHP Framework-ləri\*\*

- Ən populyar PHP framework-lərinin müqayisəsi

- MVC (Model-View-Controller) patterni və onun tətbiqi

\*\*Mövzu 9: Təhlil və Təhlil Edilmə\*\*

- PHP-də lazım olan təhlil və təhlil edilmənin anlayışı

- Təhlil və təhlil edilmə proseslərinin tətbiqi

**Mövzu 1: PHP-in Əsasları**

**"PHP-in tərəfdaşlığı və hansı proqram dilləri ilə birlikdə işləyə bilərliyi"ni ətraflı şəkildə müzakirə edək.**

PHP, öncədən gələn "Hypertext Preprocessor" qısaldılmış adıdır. PHP, dinamik veb səhifələri yaratmaq və veb tətbiqlərini inkişaf etdirmək üçün nəzərdə tutulmuş güclü bir proqramlaşdırma dilidir. Əsas olaraq, PHP server tərəfində işlənir və HTML kodu ilə birləşdirilərək dinamik veb səhifələr hazırlamağa imkan verir.

**PHP-in tərəfdaşlığı:**

PHP, tərəfdaşlığı (server-tərəfli dildir) ilə diqqəti cəlb edir. Bu, PHP kodunun veb serverində işləndiyi və işlənən veb səhifələrə çıxışın göndərildiyi deməkdir. İstifadəçilər veb səhifələrinin içində PHP kodu görmür, çünki o zaman tərəfdaş dildir və səhifəni server işlədir. Server tərəfindən işlədilən PHP kodu, HTML, CSS və digər proqram dilləri ilə birlikdə birləşdirilir, dinamik məzmun yaradır və müəyyən funksiyaları yerinə yetirir.

**Proqram dilləri ilə birlikdə işləyə bilərliyi:**

PHP, digər bir çox proqramlaşdırma dilləri ilə birlikdə işləyə bilər və bu onun böyük gücünə və yayılmasına səbəbdir. PHP ilə birlikdə əlaqəli olan başlıca proqramlaşdırma dilləri aşağıdakılardır:

1. HTML/CSS: PHP ilə birlikdə, statik HTML veb səhifələri yaratmağın yanı sıra, dinamik məzmunu HTML və CSS ilə birləşdirərək dinamik səhifələr hazırlamaq mümkündür.

2. JavaScript: JavaScript, ən populyar veb səhifələrinin interaktivliyini və dinamikliyini təmin edən bir müstəqil istifadə olunan proqramlaşdırma dilidir. PHP və JavaScript birlikdə işləyərək daha dinamik veb tətbiqlərini təmin edir.

3.MySQL və digər verilənlər bazası sistemləri: PHP, verilənlər bazası ilə əlaqə quraraq məlumatları serverdə saxlamaq və istifadəçilərin tələblərinə uyğun məzmun təklif etmək üçün verilənlər bazası ilə əlaqə qura bilir.

4. XML və JSON: PHP, XML və JSON məlumatları ilə işləmək üçün inkişaf etdirilmiş daxili funksiyonlara sahibdir. Bu, verilənlər dəyişmələri və ya məlumat dəyişikliklərini təhlil və təhlil edərək proqramlaşdırma tətbiqatları ilə əməkdaşlığı daha asanlaşdırır.

5. C/C++ və başqa dillər: PHP, C və C++ kimi proqramlaşdırma dilləri ilə uyğunlaşdırılaraq, daxili və ya əlavə genişləndirilmiş funksiyonlara və ya müstəqil proqramların çağırılmasına imkan verir.

Bu sayədə, PHP-in tərəfdaşlığı və digər dillərlə uyğunlaşdırılması, geniş funksionalı və güclü veb tətbiqlərinin yaradılmasına imkan verir. Bu da PHP-in yayılmasında və çox məşhur olmasında əsas səbəblərdən biridir.

**"PHP quraşdırılması və konfiqurasiyası" haqqında danışaq:**

PHP-i quraşdırmaq, onu yerli kompüterinizdə və ya serverdə işlədə biləcəyiniz üçün lazımlıdır. PHP, həm Windows, həm də bəzi digər əməliyyat sistemlərində çalışa bilər. İstifadə edəcəyiniz mühit, müstəqil tələblərinizə və veb hosting tərəfindən təklif olunan imkanlara bağlı olacaq.

1. PHP Mühitini Quraşdırmaq:

Əvvəlcə, PHP-in ən son versiyasını zəngin veb səhifələri hazırlamaq və müasir funksionalı tətbiqlər yaratmaq üçün rəsmi veb saytından yükləyə bilərsiniz (https://www.php.net/downloads.php).

2. Web Server Quraşdırmaq:

PHP-in əsasən dinamik veb səhifələr yaratmaq üçün istifadə edildiyi üçün, bir veb server quraşdırmaq da vacibdir. Ən populyar veb serverlərdən biri Apache HTTP Server-dır. Apache, Windows və əksər UNIX bazalı sistemlər üçün mövcuddur.

Youtube-da çoxlu videolar var:

[How to install PHP 8.2.0 on Windows 10/11 [2023 Update] Run your first PHP Program | Complete guide - YouTube](https://www.youtube.com/watch?v=MPRLUd8Pmyo&ab_channel=GeekyScript)

[How To Run PHP Projects in XAMPP Server - YouTube](https://www.youtube.com/watch?v=Op5TTJPCtso&ab_channel=ProgrammingKnowledge2)

**PHP-də sintaksis və müstəqil dəyişənlər**

PHP-də sintaksis və müstəqil dəyişənlər (variables) proqramlaşdırmanın əsas əlifbasını təşkil edir. Sintaksis, PHP-də yazılan kodun düzgün və doğru formada olmasını təmin edir. Müstəqil dəyişənlər isə məlumatları saxlamaq və onları proqramda işləmək üçün istifadə olunur. PHP-də dəyişənlər məlumatları saxlamaq üçün çeşitli tipdə olur (məsələn, rəqəmlər, mətn, array və s.) və dəyişənlər adlarla təyin edilir.

**Sintaksis qaydaları:**

1. Kodun başlanğıcı və sonu: PHP kodu, `<?php` və `?>` qaydaları arasında yerləşdirilir. Kodun başlanğıcında `<?php` yazılır və sonunda `?>` qoyulmalıdır.

2. Komentarlar: Komentarlar kodun oxunması üçün yararlıdır və kodun təhlili üçün icazə vermir. Komentarlar iki türdə ola bilər:

 - Satır içi komentarlar: `//` işarəsi ilə başlayır və həmin sətirdən sona kimi təsir edir.

 Nümunə: //Bu sətirdə hələ ki istifadəçinin yaşını qeyd etməmişəm

 - Çox sətrləri əhatə edən komentarlar: `/\*` ilə başlayır, `\*/` ilə başa çatır və bir və ya daha çox sətiri əhatə edir. Bu daha çox frameworkde görmək olur ki, php faylının nə etdiyini bilmək üçün yazırlar.

 Nümunə: Laravelin view.php faylının içindən nümunə kimi

/\*\*

     \* Get the name of the view.

     \*

     \* @return string

     \*/

    public function name();

3. Birbaşa kod yazmaq: PHP kodu, HTML faylına doğrudan birbaşa yazılaraq tətbiq oluna bilər. Nümunə:

  <?php

     echo "Hello World";

    ?>

**Müstəqil Dəyişənlər:**

PHP-də müstəqil dəyişənlər (variables) məlumatları saxlamaq üçün istifadə olunur. Dəyişənlər, dollar işarəsi (`$`) ilə başlayır və ardından bir ad gəlir. Dəyişənlər böyük kiçik hərflərə həssasdır və rəqəmlər və alətlər (nöqtə, xarakterlərlə və s.) ilə başlaya bilməz.

Dəyişənlər adlandırılarkən aşağıdakı qaydalar nəzərə alınmalıdır:

- Adlar anlaşıqlı və oxunması asan olmalıdır

$userNameOfProfile yox $username olmalıdır.

- Adlar rəqəmlərlə başlaya bilməz, lakin \_ işarəsi ilə başlaya bilər.

$\_username, $\_password

- Dəyişənlər adlarında boşluq olmamalıdır.

Nümunə:

   <?php

    $ad = "Ali";

    $yas = 25;

    $is\_student = true;

    $yas = $yas + 5;

    echo "Mənim adım: " . $ad . "<br>";

    echo "Yaşım: " . $yas . "<br>";

    echo "Tələbəyəm: " . ($is\_student ? "Bəli" : "Xeyr") . "<br>";

    ?>

Bu nümunədə, `ad`, `yas` və `is\_student` adlı üç müstəqil dəyişən yaradılır və onlara müxtəlif tiplərdə məlumatlar təyin olunur. Sonra `echo` funksiyası ilə dəyişənlərin qiymətlərini ekrana çap edirik. Dəyişənlərin qiymətlərini dəyişdirmək üçün onları sadəcə yeni qiymətlərlə əvəzləyə bilərik.

**Mövzu 2: Dizayn prinsipləri və məntiqi operatorlar**

Mövzu 2, "Dizayn prinsipləri və məntiqi operatorlar," proqramlaşdırmada kodun təmiz, effektiv və əhatəli olmasına nail olmağa kömək edən əsas prinsiplər və məntiqi operatorlar haqqında danışılır. Bu mövzuda üç əsas məqamı nəzərdən keçirək:

1. Dizayn Prinsipləri(design patterns):

 - DRY (Don't Repeat Yourself): Bu prinsip əsasən təkrarlanan kodu azaltmağa tövsiyə edir. Təkrarlanan kod hissələri funksiyalara, klasslara və ya modullara çevrilməlidir, beləliklə də kodunuz daha təmiz və idarə olunması asan olar.

 - KISS (Keep It Simple, Stupid): Kodun sadə və yaxşı anlaşılacaq formada olması tövsiyə edilir. Kompleks yanaşmaları istifadə etməkdən imtina etmək, kodun daha dəqiqliklə işləməsinə və müasirliyinə nail olmağa kömək edir.

<?php

// Pis: if else coxlugu

if ($condition1) {

    if ($condition2) {

        if ($condition3) {

            // Do something

        } else {

            // Do something else

        }

    } else {

        // Handle condition2 failure

    }

} else {

    // Handle condition1 failure

}

// Yaxşı: if else azlığı

if ($condition1 && $condition2 && $condition3) {

    // Do something

} elseif ($condition1 && $condition2) {

    // Do something else

} else {

    // Handle failures

}

if ($condition) {

    $result = 'Yes';

} else {

    $result = 'No';

}

// Good: Use ternary operator

$result = $condition ? 'Yes' : 'No';

 - YAGNI (You Aren't Gonna Need It) Bu prinsip, tələb olunmadığı halda funksionalı əlavə etməkdən imtina etməyə dəstək verir. Yəni, yalnız tələb olunan funksionalı kodunuzda əlavə edin, çünki əks təqdirdə bu, daha çox yeniləməyə səbəb ola bilər.

 -SoC (Separation of Concerns): Bu prinsip, proqramın fərqli məqsədləri arasında funksionalı və məzmunu müstəqil şəkildə tutmağa tövsiyə edir. Bu, kodun oxunması və genişləndirilməsi üçün daha asan edir.

<?php

// UserRegistration.php

class UserRegistration

{

    public function register($username, $password)

    {

        // Validation and registration logic...

        // Store user data in the database...

    }

}

// UserLogin.php

class UserLogin

{

    public function login($username, $password)

    {

        // Validation and login logic...

        // Set user authentication session...

    }

    public function logout()

    {

        // Clear user authentication session...

    }

}

// UserProfile.php

class UserProfile {

    public function displayProfile($userId) {

        // Fetch user data from the database...

        // Display user profile information...

    }

    public function updateProfile($userId, $newData) {

        // Validation and update logic...

        // Update user data in the database...

    }

}

2. Məntiqi Operatorlar:

 - Və (&&) operatoru: Bu operator, hər iki şərt doğru olduğu zaman doğru cavab verir.

 if ($name & $surname) {

    echo "Ad ve Soyad düzdür";

}

 - Və ya (||) operatoru: Bu operator, hər iki şərtin birinin doğru olması halında da doğru cavab verir.

 if ($name & $surname) {

    echo "Ad ve Soyad düzdür";

}

 - Nida :`!` operatoru, daxil etdiyi şərtin əksini göstərir. Məsələn, `!true` cavabı `false` olacaq.

 If(!$password) {

    echo "Kod sehvdir";

}

**Müəyyən edicilər, şərt operatorları və dövri operatorlar**

Müəyyən edicilər, şərt operatorları və dövri operatorlar, proqramlamanın əsas hissələrindən biridir və proqramların fərqli hal və şərtlər altında davranışlarını tənzimləməyə imkan verir. İndi hər birini daha ətraflı şəkildə açıqlayaq:

**1. Müəyyən edicilər (Comparison Operators):**

Müəyyən edicilər, dəyişənlərin və ya ifadələrin bir-biri ilə müqayisəsini təmin edir. PHP-də ən müstəqil müəyyən edicilər aşağıdakılardır:

* == : Bərabərdir - İki dəyişən və ya ifadə eyni dəyərlərə malikdirsə, doğru cavab verir.
* != və ya <> : Bərabər deyil - İki dəyişən və ya ifadə eyni dəyərlərə sahib deyilsə, doğru cavab verir.
* === : Identikdir - İki dəyişən eyni dəyər və tiplərə sahibdirsə, doğru cavab verir.
* !== : Identik deyil - İki dəyişən eyni dəyər və ya tiplərə sahib deyilsə, doğru cavab verir.
* >: böyükdür - Bir dəyişən digərindən böyükdirsə, doğru cavab verir.
* < : kiçikdir - Bir dəyişən digərindən kiçikdirsə, doğru cavab verir.
* >= : böyükdür və ya bərabərdir - Bir dəyişən digərindən böyük və ya bərabərdirsə, doğru cavab verir.
* <= : kiçikdir və ya bərabərdir - Bir dəyişən digərindən kiçik və ya bərabərdirsə, doğru cavab verir.

$a = 10;

$b = 5;

if ($a > $b) {

    echo "$a böyükdür $b";

} else {

    echo "$a kiçikdir $b";

}

1. **Şərt Operatorları (Conditional Operators):**

Şərt operatorları, şərti təyin etmək üçün istifadə olunur və çox zaman ifadələri dəyişənlərin qiymətləri ilə təyin edir. Ən ümumi şərt operatorları aşağıdakılardır:

* **&&** və **AND** : Və - Hər iki şərt doğru olarsa, doğru cavab verir.
* **||** və **OR** : Və ya - Hər iki şərtin biri doğru olarsa, doğru cavab verir.
* **!** və **NOT** : Şərt doğru deyilsə, doğru cavab verir.

<?php

$a = 10;

$b = 5;

if ($a > 5 && $b < 8) {

    echo "Hər iki şərt doğrudur.";

}

if ($a > 8 || $b < 2) {

    echo "Ən azı bir şərt doğrudur.";

}

if (!($a == $b)) {

    echo "Dəyişənlərin qiymətləri bərabər deyil.";

}

1. **Dövri Operatorlar (Loop Operators):**

Dövri operatorlar, kod bloklarının müxtəlif şərt vəziyyətlərində təkrarlanmasına kömək edir. PHP-də üç əsas dövri operator var:

* **for**: Qoşmaq üçün istifadə olunur və bir şərt doğru olduğu müddətdə bir hissəni təkrarlar.
* **while**: Şərt doğru olduğu müddətdə təkrarlanmağa davam edir.
* **do...while**: Əvvəlcə bir dəfə kodu icra edir və daha sonra şərt doğru olduğu müddətdə təkrarlanmağa davam edir.

// for dövrü

for ($i = 1; $i <= 5; $i++) {

    echo $i . " ";

}

// Çıxış: "1 2 3 4 5 "

// while dövrü

$j = 1;

while ($j <= 5) {

    echo $j . " ";

    $j++;

}

// Çıxış: "1 2 3 4 5 "

// do...while dövrü

$k = 1;

do {

    echo $k . " ";

    $k++;

} while ($k <= 5);

// Çıxış: "1 2 3 4 5 "

**Dizayn prinsipləri və kodun təkmilləşdirilməsi**

Dizayn prinsipləri və kodun təkmilləşdirilməsi, proqramlaşdırma təcrübəsinin daha effektiv və təmiz kod yaratmağa nail olmağa kömək edən mühüm hissəsidir. Proqramın funksionallığının yanında kodun təmizliyi, müasirliliyi və idarə edilə bilənliyi də çox önəmlidir. Aşağıda, kodun təkmilləşdirilməsini və bəzi dizayn prinsiplərini artırmağa kömək edən əsas məqamlara nəzər yetirək:

1. **Təmiz Kod Yazmaq:**
	* DRY prinsipi: Təkrarlanan kodu azaltmağa çalışın. Təkrarlanan kod hissələrini funksiyalara çevirin və funksiyalardan uyğun şəkildə istifadə edin.
	* KISS prinsipi: Kodunuzu sadə və anlaşıqlı saxlayın. Çox kompleks yanaşmalardan imtina edin.
	* YAGNI prinsipi: Tələb olunmayan funksionalı əlavə etməyin. Sadəcə lazım olan funksionalı əlavə edin.
2. **Kodun Effektivliyini Artırmaq:**
	* İdarə olunan təmiz kod yazmağın yanı sıra, effektivlik üçün alqoritmlər və məntiqi operatorlar üçün effektiv yanaşmalar seçin.
	* Yüklənməni minimal edin. Proqramda lüzumsuz qoşulmaları və faydalardan imtina edin.
3. **Dizayn Prinsipləri:**
	* SoC prinsipi: Funksionalı və məzmunu ayrı tutun. Funksiyalardan mümkün qədər işə salıma əsaslanın.
	* DİP prinsipi (Dependency Inversion Principle): Aşağıdan yuxarıya doğru tətbiqatlar üçün asılılığı minimuma endirin və üst səviyyədəki modullara daha çox asılı olun.
	* SOLİD prinsipləri: Bu beş prinsip, obyekt-orientasiyalı proqramlaşdırma (OOP) ilə bağlıdir və effektiv, təhlili olunmuş və genişlənən kod yazmağa nail olmağa kömək edir.
4. **Performans Optimizasiyası:**
	* İcazəsiz əməliyyatlardan imtina edin və lazım olduğu zaman optimallaşdırmağa fikir verin.
	* Databazalar və fayl əməliyyatları kimi kritikal hissələri effektiv edin və istifadə olunan alqoritmlər və tərzi əməliyyatlara əməl edin.
5. **Düzgün Şəkildə Qeydləndirin:**
	* Kodunuzu nəzərə alaraq təyin edin, əsas məqamları şərhləyin və kod bloklarını açıqlamağa nail olun.
	* Digər proqramçılar üçün kodunuzu anlaşıqlı və oxunaqlı edin.
6. **Mənfiyyətli Kodları İdarə Et:**
	* Ləyaqətli və mənfiyyətli kodları təmizləmək üçün vaxtınızı və sərf edən müddətləri əmr etməyin.
	* İdarə etmək üçün işarələmə (version control) vasitələri (məsələn, Git) istifadə edin və kodunuzu mümkün ən yaxşı vəziyyətdə saxlayın.

Həmçinin, əlaqəli tətbiqatların, kod şablonlarının və köməkçi alətlərin müstəsna formada olması, kodun təkmilləşdirilməsində böyük bir rola malikdir.

Ümumiyyətlə, proqramların təmizliyi və effektivliyi, onların tətbiq edilməsinin və təmirinin asanlığını təmin edir. Proqramlaşdırma və dizayn prinsiplərinə hörmətlə yanaşmaqla, yaxşı nəticələr əldə etmək daha asan olacaq.

**Funksiya və siniflərin düzgün tətbiqi**

Funksiyalar və siniflər, proqramlaşdırmada kodun təmiz və idarə edilə bilən olmasına imkan verir. Düzgün funksiya və sinif tətbiqi, kodun oxunması, düzəldilməsi və yenilənməsi üçün vacibdir. İndi hər birini daha ətraflı şəkildə açıqlayaq:

**Funksiyaların düzgün tətbiqi:**

* Düzgün adlandırma: Funksiyaların adları açıq və mənalı olmalıdır. Funksiyanın nə etdiyini göstərən adlar seçməyə çalışın.
* Bir iş görməsi: Funksiya bir iş görməlidir. İdeal olaraq, funksiya yalnız bir əməliyyatı icra etməlidir.
* Parametr qəbulu: Funksiyaların mümkün qədər az parametr qəbul etməsinə çalışın. Əgər çox parametr lazım olursa, funksiyanız çox mürəkkəb olmağa başlaya bilər və kodun oxunması və idarə edilməsi çətinləşə bilər.
* Geri qayıdış dəyəri: Funksiyanızın geri qayıdış dəyərini düzgün şəkildə nəzərdən keçirin. Geri qayıdış dəyəri mümkün qədər açıq və düzgün olmalıdır.
* Şərhlər: Funksiyaların təsviri şərhlər ilə açıqlayın. Digər proqramçıların funksiyanızı daha yaxşı başa düşməsinə kömək edəcəkdir.

**Siniflərin düzgün tətbiqi:**

* Sinif adlandırması: Sinifin adı mümkün qədər açıq və aydın olmalıdır. Sinifin nəyi təmsil etdiyini göstərən adlar seçin.
* Düzgün sinif tərtibatı: Sinifinizin üzvlərinin və metodlarının tərtibatını düzgün və aydın şəkildə təşkil edin.
* Encapsulation (Kapsullama): Üzvlərə və metodlara gizlilik dərəcəsi təyin edin. Məşhur Access Modifiers (public, private, protected) istifadə edin.
* Inheritance (Miras alınma): Proqramınızda ortaq funksionalı olan siniflər yaratmağa çalışın və miras alma vasitəsilə kodun təkrarlanmasını azaltın.
* Polymorphism (Çoxtipiklik): Eyni adlı metodların fərqli siniflərdə fərqli işləri etməsinə imkan verin. Bu, kodunuzun daha genişlənməsini və təmiz olmasını təmin edir.
* Şərhlər: Siniflərə uyğun şərhlər yazın. Sinifin nə etdiyini və necə istifadə edilməsi gerektiyini izah edin.

Funksiya üçün bir nümunə:

// Düzgün adlandırma və bir iş görməsi

function cemi($a, $b) {

    return $a + $b;

}

Sinif üçün bir nümunə:

// Sinif adlandırması və düzgün tərtibat

class Telefon

{

    // Üzvlər və kapsullama

    private $marka;

    private $model;

    // Metodlar və miras alma

    public function \_\_construct($marka, $model)

    {

        $this->marka = $marka;

        $this->model = $model;

    }

    public function telefonuGoster()

    {

        return "Telefon: " . $this->marka . " " . $this->model;

    }

}

Həm funksiyaları, həm də sinifləri düzgün tətbiq edərək, kodunuzun daha təmiz, idarə olunan və effektiv olmasını təmin edə bilərsiniz. Əmin olun ki, digər proqramçılar və ya özünüz daha sonra kodu oxuyub başa düşmək üçün çox mürəkkəb hissələrdən qaçınmağa nail olacaqsınız.

**Obyekt-yönümlü proqramlaşdırma (OOP)**

Obyekt- yönümlü proqramlaşdırma (OOP), proqramın funksionallığını obyektlər və siniflər kimi təsvir edən bir proqramlaşdırma paradigmasıdır. OOP, kodun təmiz, düzgün təşkil edilmiş, genişlənə bilən və idarə olunan olmasına kömək edir. İndi, OOP-nin əsas mənşəyini və mühüm konseptlərini nümunələr ilə izah edək:

1. **Sinif və Obyekt yaratmaq:**

class Meyve

{

    public $ad;

    public $reng;

    public function neseVer()

    {

        return "Meyvə nəsə verir...";

    }

}

// Obyekt yaratmaq

$alma = new Meyve();

$alma->ad = "Alma";

$alma->reng = "Qırmızı";

echo $alma->ad . " " . $alma->reng; // Çıxış: "Alma Qırmızı"

echo $alma->neseVer(); // Çıxış: "Meyvə nəsə verir..."

**2.Encapsulation**

**Encapsulation (Mənimsətmə):** OOP-də əsas prinsiplərdən biridir və obyektin daxilindəki məlumatları və funksiyonları gizlədərək daxilindəki məzmunu tənzimləməyi təmin edir. Bu prinsip məzmunun təyinatına və funksionalın açığa çıxarılmasına imkan verir. Başqa bir deyilən, daxilindən necə işlədiyini bilmədən və ya nə etdiyini anlamadan hansısa obyektin funksionalını istifadə edə bilərik.

<?php

//ENCAPSULATİON

class BankHesab

{

    private $balans;

    public function \_\_construct($balans)

    {

        $this->balans = $balans;

    }

    public function balansGoster()

    {

        return $this->balans;

    }

    public function pulCek($miktar)

    {

        if ($miktar > $this->balans) {

            return "Kifayət qədər pul yoxdur.";

        } else {

            $this->balans -= $miktar;

            return "Uğurla çəkildi.";

        }

    }

    public function pulYatir($miktar)

    {

        $this->balans += $miktar;

        return "Uğurla yatırıldı.";

    }

}

$hesab = new BankHesab(1000);

echo $hesab->balansGoster(); // Çıxış: 1000

echo $hesab->pulCek(500); // Çıxış: "Uğurla çəkildi."

echo $hesab->pulYatir(200); // Çıxış: "Uğurla yatırıldı."

**Üzvlərə və metodlara gizlilik dərəcəsi təyin etmək və məşhur olan Access Modifiers (public, private, protected) istifadə eləmək**

Access Modifiers (Daxili Giriş Növü) olan **public**, **private** və **protected**ın necə istifadə edildiyini nümunələrlə açıqlayaq:

1. Public daxili giriş növü, sinifdən və ya hər hansı başqa sinifdən istifadəçi tərəfindən istifadə edilə bilən üzvləri və metodları bildirir.

class Telefon

{

    public $marka; // Hər kəs tərəfindən istifadə edilə bilər

    public function zəngEt()

    {

        return "Zəng edilir...";

    }

}

$telefon = new Telefon();

$telefon->marka = "Samsung"; // Public üzv dəyərini dəyişmək olar

echo $telefon->zəngEt(); // Public metod istifadə etmək olar

2. private (Özəl): private daxili giriş növü, yalnız həmin sinifin daxili funksiyalarından istifadə etməyə imkan verir, başqa siniflər və ya obyektlər bu üzvlərə və metodlara çatışa bilməz.

class Kassa

{

    private $balans = 100; // Yalnız Kassa sinfi üçün gözlənilir

    private function balansDeyis($miktar)

    {

        $this->balans += $miktar;

    }

    public function pulYatir($miktar)

    {

        $this->balansDeyis($miktar);

    }

}

$kassa = new Kassa();

// $kassa->balans = 1000; // Səhv: Private üzv dəyərini dəyişmək olmaz

// $kassa->balansDeyis(50); // Səhv: Private metod istifadə edilməz

$kassa->pulYatir(50); // Doğru: Public metod vasitəsilə private metod istifadə edilir

**3. protected (Müdafiə olunan):** **protected** daxili giriş növü, yalnız həmin sinifin daxili funksiyaları və onun miras alan sinifləri tərəfindən istifadə oluna bilən üzvləri və metodları təyin edir.

class Heyvan

{

    protected $adi;

    protected function səsÇıxar()

    {

        return "Səs çıxır...";

    }

}

class It extends Heyvan

{

    public function qulaqVur()

    {

        return $this->adi . " - " . $this->səsÇıxar();

    }

}

$it = new It();

$it->adi = "Buldog"; // Səhv: Protected üzv dəyərini dəyişmək olmaz

echo $it->qulaqVur(); // Doğru: Protected metod və üzv miras alan sinifdən istifadə edilir

**Miras alınma (inheritance), obyekt-orientasiyalı proqramlaşdırmada (OOP) ən mühüm konseptlərdən biridir.**

Sinifin digər bir sinifin xüsusiyyətlərini və metodlarını özündə birləşdirməsinə imkan verir. Miras alınma, kodun təkrarlanmasını azaltmağa, kodun düzgün təşkil olunmasına və proqramın effektivliyinin artırılmasına yardım edir.

Aşağıda, miras alınma mövzusunda iki sinifin nümunələrini göstərəcəyim:

Üst sinif (Base Class) və Miras alan sinif (Derived Class):

<?php

// Üst sinif (Base Class)

class Heyvan

{

    protected $adi;

    public function \_\_construct($adi)

    {

        $this->adi = $adi;

    }

    public function səsÇıxar()

    {

        return "Səs çıxır...";

    }

}

// Miras alan sinif (Derived Class)

class It extends Heyvan

{

    public function \_\_construct($adi)

    {

        parent::\_\_construct($adi);

    }

    public function qulaqVur()

    {

        return $this->adi . " - " . $this->səsÇıxar();

    }

}

// It sinifindən obyekt yaradırıq

$it = new It("Buldog");

// Miras alan sinifdən üst sinifin metodunu çağırırıq

echo $it->qulaqVur();

// Çıxış: "Buldog - Səs çıxır..."

Bu nümunədə, Heyvan sinfi üst sinifdir və It sinfi miras alan siniftir. It sinfi, Heyvan sinfinin bütün xüsusiyyətlərini və metodlarını miras alır.

Miras alınma, təkrarlanan kodların azaldılmasını təmin edir və yeni funksionalı əlavə etməni daha asan edir. Digər obyekt-orientasiyalı proqramlaşdırma prinsipləri ilə birləşdirilərək, kodunuzun daha təmiz, effektiv və idarə olunması asan olacaq.

**Çoxtipiklik (Polymorphism), obyekt-yönümlü proqramlaşdırmada (OOP) bir obyektin müxtəlif tipdə davranışlara sahib olmağı təmin edən mühüm bir prinsipdir.**

Bu prinsip, eyni adlı metodların fərqli siniflərdə fərqli işləri etməsinə imkan verir. Çoxtipiklik sayəsində, üst siniflərin və miras alan siniflərin birlikdə işləməsi asanlaşır və kodunuz daha təmiz və genişlənə bilən olur.

**Çoxtipiklik Nümunəsi:**

<?php

// Üst sinif (Base Class)

class Heyvan

{

    protected $adi;

    public function \_\_construct($adi)

    {

        $this->adi = $adi;

    }

    public function səsÇıxar()

    {

        return "Səs çıxır...";

    }

    public function getAdi()

    {

        return $this->adi;

    }

}

// Miras alan sinif 1 (Derived Class 1)

class It extends Heyvan

{

    public function \_\_construct($adi)

    {

        parent::\_\_construct($adi);

    }

    public function səsÇıxar()

    {

        return "hav-hav!";

    }

}

// Miras alan sinif 2 (Derived Class 2)

class Qus extends Heyvan

{

    public function \_\_construct($adi)

    {

        parent::\_\_construct($adi);

    }

    public function səsÇıxar()

    {

        return "Cik-cik!";

    }

}

// Polymorphism nümunəsi

$heyvanlar = array(new It("Buldog"), new Qus("Kanarya"));

foreach ($heyvanlar as $heyvan) {

    echo $heyvan->getAdi() . ": " . $heyvan->səsÇıxar() . "<br>";

}

Burada It sinfi Heyvan sinfindən gələn səsÇıxar() metodunu özü üçün əvəz edir və eyni adla fərqli iş görür. Eyni cür Quş sinfi də fərqli davranış üçün eyni adlı metodu özü üçün tənzimləyir. Bu, çoxtipikliyin prinsipinin göstəricisidir. OOP-da çoxtipikliyin mövcudluğu kodun daha təmiz, idarə olunan və genişlənən olmasına imkan verir.

**Mövzu 3: Formla işləmə**

Abstract Klasslar:

Abstract klasslar, əsasən, təyin edilmiş funksiyaların (metodların) təyinatını, lakin əsas təyinatları olmayan, başqa sözlə, "boş" metodları olan klasslardır. Bu klassları törədilən (miras alan) alt klasslar tamamlayaraq onların funksiyonallığını özəlləşdirirlər.

İşlətildikdə, abstract klasslar şəbəkəli bir şablon kimi xidmət edir, digər klasslara istifadə olunmağa imkan verir və onları hansısa bir təyin edilmiş arayüzə (interface) uyğunlaşdırır.

abstract class Shape {

    abstract public function calculateArea();

}

class Circle extends Shape {

    private $radius;

    public function \_\_construct($radius) {

        $this->radius = $radius;

    }

    public function calculateArea() {

        return pi() \* pow($this->radius, 2);

    }

}

**Interfeyslər**: Interfeys, bir sıra təyinatı olan metodlar və ya funksiyonlar qrupunu təmsil edir. Interfeys fərqli-fərqli siniflərin eyni metodlarını özündə cəmləşdirir. Klasslar birdən çox interfeysi eyni anda implement edə bilərlər və beləliklə, fərqli klasslar eyni funksiyonallığı həyata keçirə bilər.

public interface HeynvanXususiyyetleri()

{

    public function canFly();

}

class Qartal implements HeynvanXususiyyetleri

{

    public function canFly()

    {

        return "Ucur";

    }

}

interface Logger {

    public function log($message);

}

class FileLogger implements Logger {

    public function log($message) {

        // Fayla yazmaq əməliyyatları

    }

}

class DatabaseLogger implements Logger {

    public function log($message) {

        // Verilənlər bazasına yazmaq əməliyyatları

    }

}

Əsas fərqi, abstract klassların kod hissələrinin özü də təyin edilə bilir, lakin interfeyslər yalnız metod təyinatlarını təmin edir. Bir klass yalnız bir abstract klassı miras ala bilər, amma birdən çox interfeysi implement edə bilər.

Nəticədə, abstract klasslar daha çox kod törədilmiş funksiyonallığın paylaşılması üçün istifadə edilir, interfeyslər isə eyni təyinatları fərqli klasslar arasında mübadilə etmək üçün istifadə edilir.