Web programozás PHP-ben

2024 ősz

Oktató: Dr. Pál László, egyetemi docens Email: pallaszlo@uni.sapientia.ro

Tartalomjegyzék

1. fejezet

Bevezetés a webfejlesztésbe és a PHP-ba

1.1. A web működése, kliens-szerver modell

A világháló (World Wide Web) egy olyan információs rendszer, amely lehetővé teszi dokumentumok elérését és megtekintését az interneten keresztül. A web működése a kliens-szerver modellen alapul, amely egy elosztott alkalmazásstruktúra, mely feladatokat vagy munkaterheléseket oszt el a szolgáltatást nyújtók (szerverek) és szolgáltatást kérők (kliensek) között.

1.1.1. Kliens

A kliens általában egy webböngésző (pl. Google Chrome, Mozilla Firefox, Safari, Micro soft Edge), de lehet bármilyen más program is, amely képes HTTP kéréseket küldeni és a válaszokat értelmezni. A kliens felelős:

- A felhasználói interakciók kezeléséért (pl. URL beírása, linkre

kattintás) • HTTP kérések összeállításáért és elküldéséért

- A szervertől kapott válasz feldolgozásáért és megjelenítéséért
- Kliens oldali szkriptek futtatásáért (pl. JavaScript)

1.1.2. Szerver

A szerver egy olyan számítógép vagy program, amely szolgáltatásokat nyújt más progra mok vagy felhasználók számára. A webszerver fő feladatai:

- HTTP kérések fogadása és feldolgozása
- Megfelelő erőforrások (pl. HTML fájlok, képek) vagy dinamikusan generált tarta lom szolgáltatása
- Adatbázis-kezelés
- Szerveroldali szkriptek futtatása (pl. PHP)

- Biztonság és hozzáférés-vezérlés kezelése

Népszerű webszerverek: Apache HTTP Server, Nginx, Microsoft Internet Information Services (IIS).

1.1.3. HTTP protokoll

A Hypertext Transfer Protocol (HTTP) egy alkalmazási rétegbeli protokoll, amely lehe tővé teszi a kommunikációt a kliens és a szerver között. A protokoll főbb jellemzői:

- Állapotmentes: minden kérés független az előzőektől
- Kérés-válasz alapú: a kliens küld egy kérést, a szerver válaszol
- Szöveges: a kérések és válaszok ember által is olvasható formátumúak

Egy tipikus HTTP kérés felépítése:

```
GET / index . html HTTP /1.1
Host : www . example . com
User - Agent : Mozilla /5.0 ( Windows NT 10.0; Win64 ; x64 ; rv :88.0) Gecko /20100101 Firefox
    /88.0
Accept : text / html , application / xhtml + xml , application / xml ; q =0.9 , image / webp ,*/*; q
    =0.8
Accept - Language : en - US , en ; q =0.5
Accept - Encoding : gzip , deflate , br
Connection : keep - alive
```

1.1. Kódrészlet. Példa HTTP GET kérés

Egy tipikus HTTP válasz felépítése:

```
HTTP /1.1 200 OK
Date : Mon , 23 May 2024 22:38:34 GMT
Server : Apache /2.4.38 ( Unix ) OpenSSL /1.1.1 k PHP /8.0.3
Last - Modified : Wed , 08 Jan 2024 23:11:55 GMT
Content - Type : text / html ; charset = UTF -8

<! DOCTYPE html >
<html >
<head >
    <title > Example Page </ title >
</ head >
<body >
    <h1 > Hello , World ! </h1 >
</ body >
</ html >
```

1.2. Kódrészlet. Példa HTTP válasz

1.1.4. A weboldal betöltésének folyamata

A weboldal betöltése egy összetett folyamat, amely több lépésből áll. Az alábbiakban részletesen ismertetjük ezeket a lépéseket:

1. **URL megadása:**

• A felhasználó beír egy URL-t a böngészőbe vagy rákattint egy linkre. • A böngésző elkezdi a weboldal betöltési folyamatát.

2. **DNS lekérdezés:**

- A böngésző DNS (Domain Name System) lekérdezést végez.
- Célja: megtalálni a szerver IP címét a megadott domain név

alapján. 3. **TCP kapcsolat létesítése:**

- A böngésző TCP (Transmission Control Protocol) kapcsolatot létesít a szer verrel.
- Ez biztosítja a megbízható, sorrendhelyes adatátvitelt.

4. **HTTP kérés küldése:**

- A böngésző HTTP GET kérést küld a szervernek.
- A kérés tartalmazza a kért erőforrás elérési útját és egyéb információkat (pl. böngésző típusa, elfogadott tartalom típusok).

5. **Szerver válasza:**

- A szerver feldolgozza a kérést.
- Visszaküldi a választ, amely általában tartalmazza a kért HTML dokumentu mot.
- A válasz tartalmazhat egyéb információkat is, például sütiket vagy átirányítási utasításokat.

6. **HTML tartalom feldolgozása:**

- A böngésző elkezdi elemezni és megjeleníteni a HTML tartalmat.
- Közben azonosítja a külső erőforrásokra mutató hivatkozásokat (CSS, Ja vaScript, képek).

7. **További erőforrások letöltése:**

- A böngésző további HTTP kéréseket küld a külső erőforrásokért.
- Ezek lehetnek CSS fájlok, JavaScript szkriptek, képek, videók

stb.

8. **CSS feldolgozása:**

- A böngésző feldolgozza a CSS szabályokat.
- Alkalmazza a stílusokat a HTML elemekre.

9. **JavaScript végrehajtása:**

- A böngésző végrehajtja a letöltött JavaScript kódokat.
- Ez módosíthatja a DOM-ot, interaktivitást adhat az oldalhoz, vagy további tartalmat tölthet be.

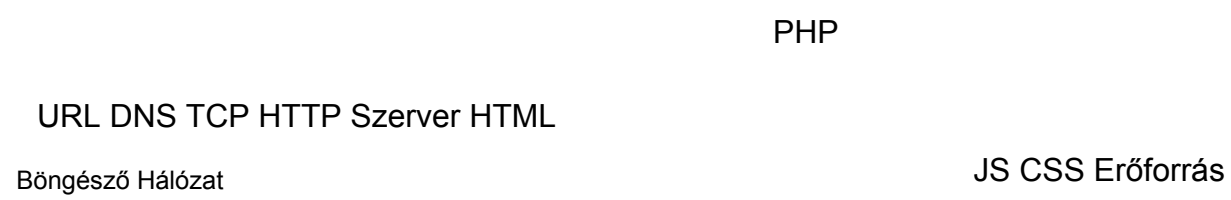
10. **Végső megjelenítés:**

    • A böngésző befejezi az oldal renderelését.

    • Megjeleníti a végső, formázott és interaktív weboldalt a felhasználónak.

Ez a folyamat általában másodpercek törtrésze alatt zajlik le, bár a pontos időtar tam függhet számos tényezőtől, mint például az internetkapcsolat sebessége, a szerver válaszideje, és a weboldal összetettsége.

Megjegyzés: PHP-alapú weboldalak esetén a szerver oldalon további lépések is tör ténnek a 4. és 5. pont között. A PHP szkript végrehajtása, adatbázis-lekérdezések, és a dinamikus tartalom generálása mind a szerveren történik, mielőtt a végső HTML válasz visszaküldésre kerülne a böngészőnek.



1.1. ábra. A weboldal betöltésének folyamata

1.2. PHP története, jellemzői

1.2.1. A PHP története

A PHP (PHP: Hypertext Preprocessor, eredetileg Personal Home Page Tools) egy szer veroldali szkriptnyelv, amelyet webalkalmazások fejlesztésére terveztek. A nyelv története 1994-ben kezdődött, amikor Rasmus Lerdorf létrehozta az első verziót. Főbb mérföldkövek:

- 1994: Rasmus Lerdorf kifejleszti az első verziót személyes használatra.
- 1995: PHP/FI (Personal Home Page / Forms Interpreter) - Az első nyilvánosan elérhető verzió.
- 1997: PHP/FI 2.0 - Körülbelül 50,000 domainben használták.
- 1998: PHP 3.0 - Andi Gutmans és Zeev Suraski újraírja a nyelv magját. A név "PHP: Hypertext Preprocessor"-ra változik.
- 2000: PHP 4.0 - Bevezetésre kerül a Zend Engine, jelentősen javítva a teljesítményt és a modularitást.
- 2004: PHP 5.0 - Fejlett objektumorientált programozási (OOP) támogatás, PDO absztrakciós réteg adatbázis-kezeléshez.
- 2015: PHP 7.0 - Jelentős teljesítményjavulás, típus-deklarációk, null coalescing operátor.

- 2020: PHP 8.0 - JIT (Just-In-Time) compiler, named arguments, attribútumok, union types.
- 2021: PHP 8.1 - Enumerációk, readonly properties, first-class callable szintaxis, fibers.
- 2022: PHP 8.2 - Readonly classes, null, false, és true mint standalone típusok, DNF (Disjunctive Normal Form) típusok.
- 2023: PHP 8.3 - Typed class constants, peremptory readonly classes, új [Override] attribútum.

A PHP 8.x sorozat számos jelentős újítást hozott a nyelvbe, amelyek javították a teljesítményt, a kód olvashatóságát és a típusbiztonságot:

- **JIT Compiler (8.0):** Jelentősen javítja a PHP kód futási sebességét bizonyos típusú alkalmazásoknál.
- **Named Arguments (8.0):** Lehetővé teszi a függvények paramétereinek név sze rinti megadását, javítva a kód olvashatóságát.
- **Attribútumok (8.0):** Metaadatok hozzáadása osztályokhoz, metódusokhoz és tulajdonságokhoz, ami hasznos például keretrendszerek fejlesztésénél.
- **Union Types (8.0):** Több lehetséges típus megadása egy változó vagy visszatérési érték számára.
- **Match Expression (8.0):** A switch utasítás egy kifejezőbb és biztonságosabb alternatívája.
- **Nullsafe Operator (8.0):** Egyszerűsíti a null ellenőrzéseket láncolható metódus hívásoknál.
- **Enumerations (8.1):** Erősen típusos felsorolások definiálása.
- **Readonly Properties (8.1):** Csak olvasható tulajdonságok definiálása osztályok ban.
- **First-class Callable Syntax (8.1):** Egyszerűbb szintaxis függvények és metódu sok referenciáinak létrehozására.
- **Fibers (8.1):** Alacsony szintű konkurrencia és aszinkronitás kezelése.
- **Readonly Classes (8.2):** Teljes osztályok definiálása csak olvasható tulajdonsá gokkal.
- **DNF Types (8.2):** Komplex típusdefiníciók létrehozása union és intersection tí pusok kombinálásával.
- **Typed Class Constants (8.3):** Típusdeklaráció hozzáadása osztálykonstansok hoz.
- **Override Attribute (8.3):** Explicit jelölése annak, hogy egy metódus felülír egy szülőosztály metódusát.

Ezek a fejlesztések mutatják, hogy a PHP folyamatosan fejlődik, hogy megfeleljen a modern webfejlesztés kihívásainak. A nyelv egyre inkább a típusbiztonságra, a teljesít ményre és a fejlesztői élmény javítására összpontosít, miközben megőrzi a PHP egyik fő erősségét: a könnyű tanulhatóságot és használatot.

1.2.2. A PHP főbb jellemzői

- **Szerveroldali végrehajtás:** A PHP kód a szerveren fut, csak az eredmény kerül elküldésre a kliensnek.
- **Beágyazhatóság HTML-be:** A PHP kód könnyen integrálható HTML doku mentumokba.
- **Platformfüggetlenség:** A PHP számos operációs rendszeren és webszerveren fut.
- **Széles körű adatbázis támogatás:** Támogatja a legtöbb népszerű adatbázis kezelő rendszert (MySQL, PostgreSQL, SQLite, Oracle, stb.).
- **Gazdag beépített függvénykészlet:** A PHP számos beépített függvénnyel ren delkezik különböző feladatokhoz (pl. sztringkezelés, fájlműveletek, hálózati kom munikáció).
- **Dinamikus típusosság:** A változók típusát nem kell előre deklarálni, a PHP automatikusan kezeli a típuskonverziókat.
- **Többféle programozási paradigma támogatása:** Támogatja a procedurális, objektumorientált és funkcionális programozást is.
- **Nagy és aktív közösség:** Rengeteg erőforrás, könyvtár és keretrendszer érhető el PHP-hez.
- **Nyílt forráskód:** A PHP ingyenesen használható és fejleszthető.

1.2.3. PHP vs. más szerveroldali technológiák

Érdemes röviden összehasonlítani a PHP-t néhány másik népszerű szerveroldali techno lógiával:

- **PHP vs. Node.js:**
  - PHP: Kifejezetten webes fejlesztésre tervezve, könnyű tanulhatóság, széles kö rű támogatás.
  - Node.js: JavaScript alapú, eseményvezérelt, aszinkron működés, jó teljesít mény real-time alkalmazásoknál.
- **PHP vs. Python:**
  - PHP: Webes fejlesztésre optimalizált, széles körű hosting támogatás.
  - Python: Általános célú nyelv, erős a tudományos számításokban és a gépi tanulásban.
- **PHP vs. Java (pl. JSP):**

- PHP: Gyorsabb fejlesztés, egyszerűbb telepítés és konfigurálás.
- Java: Erősebb típusosság, jobb teljesítmény nagyobb alkalmazásoknál, fejlet tebb többszálú működés.

1.3. Fejlesztői környezet beállítása (XAMPP/WAMP)

A PHP fejlesztéshez szükségünk van egy webszerverre, PHP értelmezőre és általában egy adatbázis-kezelő rendszerre. Bár ezeket egyenként is telepíthetjük, léteznek kényelmes, előre összeállított csomagok, amelyek mindent tartalmaznak, amire szükségünk lehet a fejlesztéshez.

1.3.1. XAMPP

A XAMPP egy ingyenes, nyílt forráskódú, platformfüggetlen webszerver megoldáscsomag, amelynek fő komponensei:

• **X:** Cross-platform (többplatformos)

• **A:** Apache HTTP Server

• **M:** MariaDB (MySQL fork)

• **P:** PHP

• **P:** Perl

XAMPP telepítése és használata:

1. Töltsd le a XAMPP-ot a hivatalos oldalról: https://www.apachefriends.org/
2. Futtasd a telepítőt és kövesd az utasításokat. Válaszd ki a kívánt komponenseket (általában az alapértelmezett beállítások megfelelőek).
3. A telepítés után indítsd el a XAMPP Control Panel-t.
4. Indítsd el az Apache és MySQL szolgáltatásokat a Control Panel-ben.
5. Nyisd meg a böngésződben a http://localhost címet a telepítés teszteléséhez. Ha minden rendben ment, egy üdvözlő oldalt fogsz látni.
6. A PHP fájljaidat a "htdocs" mappába kell elhelyezned (általában C:\xampp\htdocs Windows-on, vagy /opt/lampp/htdocs Linux-on).

1.3.2. WAMP

A WAMP egy Windows-specifikus webfejlesztői környezet,

komponensei: • **W:** Windows operációs rendszer

• **A:** Apache HTTP Server

• **M:** MySQL

 • **P:** PHP

A WAMP telepítése hasonló a XAMPP-hoz, de csak Windows rendszereken használ ható.

1.3.3. Kódszerkesztési lehetőségek

A PHP kód írásához számos lehetőség áll rendelkezésünkre. Az egyszerű szövegszer kesztőktől kezdve a komplex integrált fejlesztői környezetekig (IDE) széles a választék. Nézzünk meg néhány népszerű opciót:

Egyszerű szövegszerkesztők

- **Notepad++ (Windows):** Ingyenes, könnyen használható, szintaxis kiemeléssel.
- **Sublime Text (Többplatformos):** Gyors, testreszabható, fizetős de korlátlanul használható próbaverzióval.
- **Atom (Többplatformos):** Nyílt forráskódú, nagyon testreszabható, GitHub in tegrációval.
- **Visual Studio Code (Többplatformos):** Microsoft által fejlesztett, ingyenes, kiterjedt funkcionalitással és bővítmény-támogatással.

Integrált fejlesztői környezetek (IDE-k)

- **PhpStorm (Többplatformos):** JetBrains által fejlesztett, kifejezetten PHP-hoz. Számos fejlett funkcióval rendelkezik, de fizetős.
- **NetBeans (Többplatformos):** Ingyenes, nyílt forráskódú, támogatja a PHP mel lett más nyelveket is.
- **Eclipse PDT (Többplatformos):** Az Eclipse IDE PHP Development Tools (PDT) kiegészítővel. Ingyenes és nyílt forráskódú.

Online fejlesztői környezetek

- **Repl.it:** Böngészőből használható fejlesztői környezet, amely támogatja a PHP-t és sok más nyelvet.
- **CodePen:** Elsősorban frontend fejlesztésre, de PHP kód írására és tesztelésére is alkalmas.

Kódszerkesztő választása

A megfelelő kódszerkesztő vagy IDE választása nagyban függ a személyes preferenciáktól és a projekt igényeitől. Kezdőknek ajánlott egy egyszerűbb szövegszerkesztővel kezdeni, mint a Notepad++ vagy a Visual Studio Code, majd ahogy fejlődnek a készségek, érdemes lehet egy komplexebb IDE-re váltani.

Néhány fontos szempont a választáshoz:

- **Szintaxis kiemelés:** Segít a kód olvashatóságában és a hibák gyorsabb észlelésé ben.
- **Kód kiegészítés:** Gyorsítja a kódolást és segít elkerülni az elgépeléseket.
- **Hibakeresés (debugging):** Lehetővé teszi a kód lépésenkénti végrehajtását és a változók vizsgálatát.
- **Verziókezelés integrációja:** Megkönnyíti a Git vagy más verziókezelő rendszerek használatát.
- **Testreszabhatóság:** Lehetőség a környezet személyre szabására és bővítmények használatára.

Példa: Visual Studio Code beállítása PHP fejlesztéshez

A Visual Studio Code egy népszerű és sokoldalú választás PHP fejlesztéshez. Íme néhány lépés a beállításához:

1. Töltsd le és telepítsd a Visual Studio Code-ot a hivatalos

weboldalról. 2. Nyisd meg a VS Code-ot, és telepítsd a PHP

IntelliSense bővítményt. 3. (Opcionális) Telepítsd a PHP Debug

bővítményt a hibakereséshez. 4. Állítsd be a PHP végrehajtható fájl

elérési útját a beállításokban. 5. Hozz létre egy új .php fájlt és kezdj el

kódolni!

A Visual Studio Code számos hasznos funkciót kínál, mint például az integrált termi nál, git verziókövetés, és élő szerverkörnyezet bővítmények PHP-hoz.

1.3.4. Fejlesztői környezet tesztelése

A fejlesztői környezet beállítása után érdemes tesztelni, hogy minden megfelelően működik e. Hozzunk létre egy egyszerű PHP fájlt a htdocs mappában:

```
<? php
      phpinfo () ;
? >
```

1.3. Kódrészlet. phpinfo.php

Ezt a fájlt elmentjük ‘phpinfo.phpnéven, majd böngészőben megnyitjuk a http://localhost/phpinfo.php címet. Ha minden rendben van, egy részletes informá ciós oldalt fogunk látni a PHP konfigurációnkról.

1.4. Első PHP program

Most, hogy beállítottuk és teszteltük a fejlesztői környezetet, írjuk meg az első PHP programunkat! Ez a program bemutatja a PHP néhány alapvető funkcióját. Hozzunk létre egy ‘hello.php‘fájlt a ‘htdocs‘mappában:


```
<! DOCTYPE html >
< head >
        < meta charset ="UTF -8">
        < meta name =" viewport " content =" width =device -width , initial - scale =1.0 ">
        < title > My First PHP Program </ title >
</ head >
< body >
        <h1 > Welcome to PHP ! </ h1 >

        <? php
                // Ez egy egysoros komment

                /*
                Ez egy tö bbsoros
                komment blokk
                */

                // Vá ltoz ók és echo haszn á lata
                $name = " User ";
                $age = 25;
                echo "<p>Hello , $name ! You are $age years old . </p>";

                // Felt é teles utas ítás
                if ( $age >= 18) {
                        echo "<p>You are an adult . </p>";
                } else {
                        echo "<p>You are a minor . </p>";
                }

                // Tömbök
                $fruits = array (" apple ", " pear ", " banana ", " orange ") ;
                echo "<p>My favorite fruit is: " . $fruits [2] . " </p>";

                // Ciklus
                echo "<ul >";
                foreach ( $fruits as $fruit ) {
                        echo "<li >$fruit </li >";
                }
                echo " </ul >";

                // Fü ggv ény defini álása és haszn á lata
                function add ( $a , $b ) {
                        return $a + $b ;
                }

                $result = add (5 , 3) ;
                echo "<p >5 + 3 = $result </p>";

                // Dá tum és idő
                echo "<p> Today ’s date : " . date ("Y-m-d") . " </p>";
                echo "<p> Current time : " . date ("H:i:s") . " </p>";
        ? >
```

```
</ body >
</ html >
```

1.4. Kódrészlet. hello.php

Ebben a példában láthatjuk a PHP számos alapvető elemét:

• **PHP tagek:** A PHP kód <?php és ?> tagek közé kerül.

• **Kommentek:** Egysoros (//) és többsoros (/* */) kommentek használata. • **Változók:** A $ jellel definiáljuk őket, dinamikus típusúak.

• **Echo:** Kiírás a kimenetre.

• **Sztringek:** Idézőjelek között, változókat közvetlenül beilleszthetünk. • **Feltételes utasítások:** if-else szerkezet.

• **Tömbök:** Az array() függvénnyel vagy [] szintaxissal hozhatjuk létre. • **Ciklusok:** foreach ciklus tömbök bejárására.

• **Függvények:** Saját függvények definiálása és használata.

• **Beépített függvények:** Például a date() függvény használata. 15

2. fejezet

PHP alapok

2.1. Szintaxis, változók, adattípusok

2.1.1. PHP kód szintaxisa és beágyazása HTML-be

A PHP kód általában HTML-be ágyazva jelenik meg, ami lehetővé teszi a dinamikus tartalom generálását statikus weboldalakon. A PHP kód a <?php kezdő határoló és a ?> záró határoló között helyezkedik el. Egy PHP kódblokk tehát a következőképpen néz ki:

```
<? php
// PHP kód ide ker ül
```

```
? >
```

2.1. Kódrészlet. PHP kód határolók

Nézzünk egy egyszerű példát a PHP kód HTML-be ágyazására:

```
<! DOCTYPE html >
< html >
< head >
    < title > PHP p é lda </ title >
</ head >
< body >
    <h1 > Ü dv ö z ö lj ü k ! </ h1 >
    <? php
    $name = " Felhaszn áló";
    echo "<p>Hello , $name ! </p>";
    ? >
</ body >
</ html >
```

2.2. Kódrészlet. PHP kód beágyazása HTML-be

Ebben a példában láthatjuk, hogyan kombinálódik a statikus HTML tartalom a dina mikus PHP kóddal. A PHP blokkon belül definiálunk egy változót, majd kiírjuk annak értékét. Ez a megközelítés lehetővé teszi a dinamikus tartalomgenerálást, ami a PHP egyik fő erőssége a webes alkalmazások fejlesztésében.

Fontos megjegyezni, hogy a PHP utasításokat pontosvesszővel (;) kell lezárni. Ez lehetővé teszi több utasítás egy sorba írását, bár a kód olvashatósága szempontjából általában jobb gyakorlat minden utasítást új sorba írni.

2.1.2. Változók

A PHP-ban a változókat a $ jellel kezdjük. A változónevek kis- és nagybetű érzékenyek, és csak betűkkel vagy alulvonással kezdődhetnek, amit betűk, számok és alulvonások követhetnek. Az alábbi példa bemutatja a változók deklarálását és használatát:

```
<? php
$name = " John ";
$age = 30;
$is_student = false ;

echo $name ; // Kimenet : John
echo $Age ; // Hiba : Undefined variable
? >
```

2.3. Kódrészlet. Változók deklarálása és használata

A PHP dinamikus és gyenge típusossága miatt egy változó típusa megváltozhat a program futása során, és a nyelv automatikusan konvertálja a típusokat, ha szükséges. Ezt szemlélteti a következő példa:

```
<? php
$var = "42";
echo $var + 8; // Kimenet : 50 (a string automatikusan int -té konvert ál ódik)

$var = " Hello ";
```

```
$var = $var + 1; // Figyelmeztet és, de a mű velet vé grehajt ódik echo $var ; // Kimenet : 1 (a
string 0 -vá konvert álódik , majd hozz áadó dik 1)
? >
```

2.4. Kódrészlet. Dinamikus típusosság és automatikus konverzió

Ez a viselkedés növeli a rugalmasságot, de potenciális hibaforrás is lehet, ezért fontos tisztában lenni a típuskonverziók szabályaival.

A PHP nyelvben a kis- és nagybetű érzékenység különböző elemekre másképp vonat kozik, ami fontos szempont a kód írása és hibakeresése során:

- **Változónevek**: Kis- és nagybetű érzékenyek
- **Függvénynevek**: Nem kis- és nagybetű érzékenyek
- **Konstansok**: Hagyományosan nagybetűsek, de valójában kis- és nagybetű érzéke nyek
- **Kulcsszavak** (pl. if, else, while): Nem kis- és nagybetű érzékenyek

Példa a kis- és nagybetű érzékenységre:

```
<? php
$name = " John ";
$Name = " Jane ";
echo $name ; // Kimenet : John
echo $Name ; // Kimenet : Jane

function sayHello () {
        echo " Hello !";
```



```
}
SAYHELLO () ; // Működik , mert a fü ggvé nynevek nem kis - és nagybet ű érzé kenyek

define (" MY_CONSTANT ", 42) ;
echo MY_CONSTANT ; // Működik
echo my_constant ; // Működik , de nem ajá nlott
? >
```

Ez a viselkedés lehetővé teszi a rugalmas kódírást, de egyben potenciális hibaforrás is lehet, ezért fontos a következetes névkonvenciók használata.

2.1.3. PHP adattípusok

A PHP egy dinamikusan és gyengén típusos nyelv. Ez azt jelenti, hogy egy változó típusa a hozzárendelt érték alapján automatikusan meghatározódik, és a típusok között automatikus konverzió történhet. A PHP-ban a következő alaptípusok léteznek:

1. **Integer** (**egész szám**): Előjeles egész számok. Értéktartománya platformfüggő, 32 bites rendszereken általában -2,147,483,648 és 2,147,483,647 között van.
2. **Float** (**lebegőpontos szám**): Tizedes törtek. Pontossága platformfüggő, általá ban a 14 tizedesjegyig pontos.
3. **String** (**karakterlánc**): Szöveges adatok. Mérete csak a rendelkezésre álló

memó ria mennyisége által korlátozott.

4. **Boolean** (**logikai érték**): Igaz (**TRUE**) vagy hamis (**FALSE**) értékek. 5.

Array (**tömb**): Értékek rendezett gyűjteménye.

6. **Object** (**objektum**): Osztályok példányai.

7. **NULL**: Speciális érték, ami a "nincs érték" állapotot jelöli.

8. **Resource** (**erőforrás**): Külső erőforrásokra (pl. fájlok, adatbázis-kapcsolatok) mutató speciális változók.

Nézzünk példákat ezekre az adattípusokra:

```
<? php
$intVar = 42; // Integer
$floatVar = 3.14; // Float
$stringVar = "Hello , World !"; // String
$boolVar = true ; // Boolean
$arrayVar = [1 , 2 , 3]; // Array
$nullVar = NULL ; // NULL

class Car {
        public $brand ;
}
$objectVar = new Car () ; // Object

$resourceVar = fopen (" example .txt", "r") ; // Resource
? >
```

2.5. Kódrészlet. PHP alaptípusok példái

2.1.4. Konstansok

A konstansok olyan azonosítók, amelyek értéke nem változhat a program futása során. Konstansokat kétféleképpen definiálhatunk:

1. A define() függvénnyel

2. A const kulcsszóval

Az alábbi példa bemutatja mindkét módszert:

```
<? php
define ("PI", 3.14) ;
const MAX_VALUE = 100;

echo PI ; // Kimenet : 3.14
echo MAX_VALUE ; // Kimenet : 100
? >
```

2.6. Kódrészlet. Konstansok definiálása

A define() függvény és a const kulcsszó közötti fő különbségek:

- A define() futási időben értékelődik ki, míg a const fordítási időben.

- A define() bárhol használható a kódban, míg a const csak a legfelső szinten vagy osztályokban.
- A define() használható dinamikus nevekkel és értékekkel, a const nem.

2.1.5. Kiíratások a kimenetre

A PHP több módszert kínál az adatok kiíratására. A leggyakrabban használt módszerek: • **echo**: Több érték kiírására is használható

- **print**: Egyetlen érték kiírására szolgál
- **printf**: Formázott szöveg kiírására alkalmas
- **print_r**: Strukturált kiíratásra használható (pl. tömbök esetén) •

var_dump: Részletes típus és érték információt ad

Az alábbi példa bemutatja ezeket a kiíratási módszereket:

```
<? php
$name = " John ";
$age = 30;

// echo - több értéket is kií rhatunk vessz ővel elvá lasztva echo " Name : ", $name
, ", Age : ", $age , "<br >";

// print - csak egy értéket ír ki
print "Hello , $name ! <br >";

// printf - form á zott kiírás
```



```
printf (" Name : %s, Age : %d<br >", $name , $age ) ;

// print_r - struktur ált kiíratás ( hasznos tömbök és objektumok eset én) $arr = ['apple ', 'banana ',
'cherry '];
print_r ( $arr ) ;

// var_dump - ré szletes inform ációt ad a vá ltoz óról
var_dump ( $name ) ;
? >
```

2.7. Kódrészlet. Különböző kiíratási módszerek

Az echo és print közötti fő különbségek:

- Az echo nem ad vissza értéket, míg a print mindig 1-et ad vissza.
- Az echo képes több paramétert fogadni, a print csak egyet.
- Az echo általában gyorsabb, különösen több érték kiírásakor.

2.1.6. Típusokkal kapcsolatos függvények

A PHP számos beépített függvényt kínál a változók típusának ellenőrzésére és kezelésére. A legfontosabb függvények:

- Típusellenőrző függvények: is_int(), is_float(), is_string(), is_array(), is_object(), is_null(), is_resource()
- Típuskonverziós függvények: intval(), floatval(), strval()
- Típus lekérdezése: gettype()
- Típus beállítása: settype()

Az alábbi példa bemutatja ezen függvények használatát:

```
<? php
$value = "42";

// Típus ellen őrzése
var_dump ( is_string ( $value ) ) ; // bool ( true )
var_dump ( is_int ( $value ) ) ; // bool ( false )
var_dump ( is_numeric ( $value ) ) ; // bool ( true )

// Tí puskonverzi ó
$int_value = intval ( $value ) ;
var_dump ( $int_value ) ; // int (42)

// Típus lek é rdez ése
echo gettype ( $value ) ; // string

// Vá ltoz ó típus ának beállítása
settype ( $value , " integer ") ;
var_dump ( $value ) ; // int (42)
? >
```

2.8. Kódrészlet. Típusellenőrző és típuskezelő függvények

Ezek a függvények különösen hasznosak lehetnek a hibakeresés során, illetve amikor biztosak akarunk lenni egy változó típusában egy művelet végrehajtása előtt. A gyenge típusosság miatt ezek a függvények segíthetnek elkerülni a nem kívánt típuskonverziókat és az ezekből eredő hibákat.

2.2. Sztringkezelés

A sztringek (karakterláncok) kezelése a PHP egyik erőssége és gyakran használt funkciója. A PHP rugalmas és sokrétű eszköztárat biztosít a sztringek manipulálására, ami lehetővé teszi a fejlesztők számára, hogy hatékonyan dolgozzanak szöveges adatokkal.

2.2.1. Sztringek megadási módjai

PHP-ban négyféle módon adhatunk meg sztringeket, mindegyiknek megvan a maga sa játossága és használati területe:

1. **Aposztrófok (’...’)**: Egyszerű sztringekhez használjuk. Az aposztrófok között meg adott szöveg literális értékként kerül értelmezésre, ami azt jelenti, hogy a változók és a legtöbb escape karakter nem kerül feldolgozásra.

2. **Idézőjelek ("...")**: Az idézőjelek között megadott sztringekben a PHP értelmezi a változókat és az escape karaktereket. Ez lehetővé teszi a dinamikus sztringek egyszerű létrehozását.

3. **Heredoc**: A heredoc szintaxis többsoros sztringek megadására szolgál. Hasonlóan az idézőjeles sztringekhez, a heredoc is értelmezi a változókat és az escape karak tereket.

4. **Nowdoc**: A nowdoc szintaxis szintén többsoros sztringekhez használható, de az aposztrófos sztringekhez hasonlóan literális értékként kezeli a tartalmat, nem értel mezi a változókat és az escape karaktereket.

Nézzünk példákat ezekre a megadási módokra, kiemelve a közöttük lévő különbsége ket:

```
<? php
$name = " John ";
$age = 30;

// Aposztr ófok
echo ’Hello $name , you are $age years old.’;
// Kimenet : Hello $name , you are $age years old.

// Idéző jelek
echo " Hello $name , you are $age years old.";
// Kimenet : Hello John , you are 30 years old.

// Heredoc
echo <<< EOT
Hello $name ,
You are $age years old .
This is a multi - line string .
```



```
EOT ;
// Kimenet :
// Hello John ,
// You are 30 years old .
// This is a multi - line string .

// Nowdoc
echo <<< EOT
Hello $name ,
You are $age years old .
Variables are not interpreted here .
EOT ;
// Kimenet :
// Hello $name ,
// You are $age years old .
// Variables are not interpreted here .
? >
```

2.9. Kódrészlet. Sztringek megadási módjai és különbségeik

Fontos megjegyezni, hogy az aposztrófos és a nowdoc szintaxis általában gyorsabb feldolgozást tesz lehetővé, mivel a PHP-nek nem kell a változókat és az escape karakte reket értelmezni. Azonban ez a teljesítménybeli különbség csak nagyon nagy mennyiségű szöveg esetén válik észrevehetővé.

2.2.2. Változók automatikus behelyettesítése (variable interpo lation)

A változók automatikus behelyettesítése, vagy más néven interpolációja, lehetővé teszi a változók értékének közvetlen beillesztését a sztringekbe. Ez a funkció azonban nem minden megadási módnál működik egyformán:

- **Aposztrófok (’...’)**: Nem támogatják a változó interpolációt. A változónevek lite rális szövegként jelennek meg.
- **Idézőjelek ("...")**: Támogatják a változó interpolációt.
- **Heredoc**: Támogatja a változó interpolációt, hasonlóan az idézőjeles sztringekhez.
- **Nowdoc**: Nem támogatja a változó interpolációt, hasonlóan az aposztrófos sztrin gekhez.

Példák a változók behelyettesítésére és a különbségekre:

```
<? php
$fruit = " alma ";
$count = 5;

// Aposztr ó fokkal - nincs interpol áció
echo ’Van $count darab $fruit .’;
// Kimenet : Van $count darab $fruit .

// Idéző jelekkel - van interpol áció
echo "Van $count darab $fruit .";
// Kimenet : Van 5 darab alma .
```



```
// Komplexebb kifejez ések
echo "A gyümölcs ök száma: { $count + 1}";
// Kimenet : A gyümölcs ök száma: 6

// Tö mbelem interpol ációja
$fruits = [’alma ’, ’körte ’, ’szilva ’];
echo "A kedvenc gyümölcs öm: { $fruits [0]} ";
// Kimenet : A kedvenc gyümölcsöm: alma

// Objektum tulajdons ág interpol ációja
$obj = new stdClass () ;
$obj - > name = " John ";
echo "A nevem {$obj - > name }";
// Kimenet : A nevem John
? >
```

2.10. Kódrészlet. Változók behelyettesítése különböző megadási módokban

Az interpoláció használatakor fontos figyelni a szintaxisra, különösen összetettebb ki fejezések esetén. A kapcsos zárójelek {} használata segít egyértelműsíteni a PHP számára, hogy mit kell értelmezni változóként vagy kifejezésként.

2.2.3. Escape karakterek és kezelésük

Az escape karakterek speciális jelentéssel bíró karakterek a sztringekben, amelyek lehető vé teszik olyan karakterek beszúrását, amelyeket egyébként nehéz vagy lehetetlen lenne közvetlenül megadni. PHP-ban a leggyakrabban használt escape karakter a backslash (\).

A leggyakoribb escape szekvenciák:

• \n - Új sor

• \r - Kocsi vissza (carriage return)

• \t - Tabulátor

• \$ - Dollárjel (változó interpoláció elkerülésére)

• \" - Idézőjel idézőjeles sztringben

• \’ - Aposztróf aposztrófos sztringben

• \\ - Backslash

Fontos különbség van az aposztrófos és az idézőjeles sztringek között az escape karak terek kezelésében:

• Aposztrófos sztringekben (’. . . ’) csak a \’ és \\ escape szekvenciák

értelmezettek. • Idézőjeles sztringekben (". . . ") minden escape szekvencia

értelmezésre kerül. Lássunk néhány példát:


```
<? php
// Idéző jeles sztringben
echo "Új sor :\ nTabul átor :\ tVissza \\"; // Kimenet : Új sor :(új sor) Tabul á tor :( tab) Vissza \

// Aposztr ófos sztringben
echo ’Új sor :\ nTabul átor :\ tVissza \\ ’; // Kimenet : Új sor :\ nTabul átor :\ tVissza \

// Vá ltoz ó interpol áció elker ülése
$name = " John ";
echo "A nevem \ $name "; // Kimenet : A nevem $name echo "A nevem $name "; //
Kimenet : A nevem John
```

```
// Idéző jelek haszn á lata sztringben
echo "Ő azt mondta : \" Szia !\" "; // Kimenet : Ő azt mondta : " Szia !" echo 'Ő azt mondta : " Szia !"
'; // Kimenet : Ő azt mondta : " Szia !"

// Aposztr ófok haszn á lata sztringben
echo "Ez egy aposztr óf: '"; // Kimenet : Ez egy aposztr óf: ' echo 'Ez egy aposztr óf: \ ''; // Kimenet
: Ez egy aposztr óf: ' ? >
```

2.11. Kódrészlet. Escape karakterek használata inline kimenetekkel Az escape karakterek használata különösen fontos lehet bizonyos helyzetekben:

- SQL lekérdezések írásakor, idézőjelek escapelése:

```
$query = " SELECT * FROM users WHERE name = \" John \"";
echo $query ; // Kimenet : SELECT * FROM users WHERE name = " John "
```

- JSON vagy más formátumú adatok generálásakor:

```
$json = "{\" name \": \" John \" , \" age \": 30}";
echo $json ; // Kimenet : {" name ": " John " , "age ": 30}
```

- Reguláris kifejezések írásakor, speciális karakterek escapelése:

```
$pattern = " /\\ bword \\b/";
echo $pattern ; // Kimenet : /\ bword \b/
```

Az escape karakterek helyes használata fontos a sztringek pontos formázásához és a nem kívánt értelmezések elkerüléséhez, különösen amikor változó interpolációval vagy speciális karakterekkel dolgozunk.

2.2.4. Karakterlánc műveletek és hasznos függvények

A PHP gazdag eszköztárat kínál a karakterláncok manipulálására, beleértve az alapvető műveleteket és a specializált függvényeket. Ez a rész áttekinti a legfontosabb sztringkeze lési technikákat, bemutatva mind a beépített operátorokat, mind a hasznos függvényeket. A leggyakrabban használt karakterlánc műveletek és függvények:

- **Összefűzés és módosítás:**

  - Összefűzés operátor (.)
  - Hozzáfűzés operátor (.=)
  - str_replace() - Helyettesítés
  - substr() - Részsztring kivágása

- **Karakterek elérése és módosítása:**

  - Indexelés ([])
  - strlen() - Sztring hosszának lekérdezése

- **Keresés és összehasonlítás:**
  - strpos() - Részsztring keresése
  - strcmp() - Sztring összehasonlítás
- **Formázás és átalakítás:**
  - strtolower() - Kisbetűssé alakítás
  - strtoupper() - Nagybetűssé alakítás
  - ucfirst() - Első betű nagybetűsítése
  - ucwords() - Minden szó első betűjének nagybetűsítése
  - trim() - Whitespace karakterek eltávolítása
  - sprintf() - Sztring formázás
- **Darabolás és összeállítás:**
  - explode() - Sztring darabolása
  - implode() - Sztring összeállítása tömbből
- **Speciális műveletek:**
  - preg_match() - Reguláris kifejezés illesztése

Lássunk példákat ezekre a műveletekre és függvényekre:

```
<? php
// Ö sszef űzés és módos ítás
$str1 = " Hello ";
$str2 = " World ";
$result = $str1 . " " . $str2 ; // Hello World
$str = " Hello ";
$str .= " World "; // Hello World
$new_str = str_replace (" World ", " PHP", $str ) ; // Hello PHP $sub = substr ( $str
, 0 , 5) ; // " Hello "

// Karakterek elérése és módosítása
$char = $str [0]; // 'H'
$str [0] = 'J'; // Jello World
$length = strlen ( $str ) ; // 11

// Keres és és ö sszehasonl ítás
```



```
$position = strpos ( $str , " World ") ; // 6
$comparison = strcmp (" apple ", " banana ") ; // Negat ív érték

// Form ázás és á talak ítás
echo strtolower ( $str ) ; // hello world
echo strtoupper ( $str ) ; // HELLO WORLD
echo ucfirst ( strtolower ( $str ) ) ; // Hello world
echo ucwords ( strtolower ( $str ) ) ; // Hello World
$trimmed = trim (" Hello ") ; // " Hello "
$formatted = sprintf (" Name : %s, Age: %d", " John ", 30) ;
```

```
// " Name : John , Age : 30"

// Darabol ás és ö sze állítás
$parts = explode (" ", $str ) ; // [" Hello " , " World "]
$array = [" Hello ", " World "];
$joined = implode (" ", $array ) ; // " Hello World "

// Speci ális mű veletek
$pattern = "/\b[A-Za -z0 -9._%+ -]+@[A-Za -z0 -9. -]+\.[A-Z|a-z]{2 ,}\ b/"; $email = "
user@example .com ";
if ( preg_match ( $pattern , $email ) ) {
        echo "Érvé nyes email cím";
}
? >
```

2.12. Kódrészlet. Karakterlánc műveletek és függvények példái

Ezek a műveletek és függvények alapvető eszközök a PHP-ben történő sztringkezelés hez. Néhány fontos megjegyzés:

- A PHP-ben a sztringek módosíthatók közvetlenül az indexek használatával, de óva tosan kell eljárni, mert a nem létező indexekre való hivatkozás hibát okozhat.
- Sok sztringkezelő függvénynek van többbájtos verziója (pl. mb_strlen(), mb_substr()), amelyek helyesen kezelik az UTF-8 kódolású szövegeket. Ezeket érdemes használni többnyelvű vagy nem latin betűs karaktereket tartalmazó szövegek esetén.
- A sprintf() függvény rendkívül hasznos komplex sztringek formázásához, különö sen amikor különböző típusú adatokat kell egy sztringbe illeszteni.
- Reguláris kifejezések (mint a preg_match() példában) nagyon erőteljes eszközök komplex sztring mintázatok keresésére és validálására, de használatuk némi gya korlatot igényel.

A sztringkezelési műveletek és függvények hatékony használata kulcsfontosságú a PHP programozásban, különösen webes alkalmazások fejlesztése során, ahol gyakran kell szö veges adatokat feldolgozni és manipulálni.

Ezek a függvények rendkívül hasznosak a mindennapi programozási feladatokban. Fontos megjegyezni, hogy sok sztringkezelő függvénynek van egy többbájtos verziója is (pl. mb_strlen(), mb_substr()), amelyek helyesen kezelik az UTF-8 kódolású szövege ket.

2.2.5. További fontos szempontok

- **Teljesítmény**: Nagy mennyiségű sztringmanipuláció esetén érdemes figyelni a me móriahasználatra. A PHP sztringjei belsőleg C karaktertömbökként vannak imp lementálva, így a nagy sztringek módosítása memóriaigényes lehet. Nagyobb adat mennyiségeknél fontoljuk meg a sztringek helyett a streamek használatát.

- **Többnyelvű alkalmazások**: UTF-8 kódolású sztringek kezeléséhez használjuk az mb_* függvényeket. Ezek helyesen kezelik a többbájtos karaktereket, ami elenged hetetlen a nem latin betűs nyelvek esetében.

- **Biztonság**: Felhasználói inputok kezelésekor mindig használjunk megfelelő szűrést és escapelést. Például HTML kontextusban használjuk a htmlspecialchars() függvényt az XSS (Cross-Site Scripting) támadások elkerülése érdekében.

- **Összehasonlítás**: Sztringek összehasonlításakor legyünk körültekintőek. Az == operátor típuskonverziót végez, míg az === szigorú összehasonlítást. Használjuk a strcmp() vagy strcasecmp() függvényeket a pontosabb összehasonlításhoz.

- **Kódolás**: Mindig legyünk tisztában a sztringjeink kódolásával. Az UTF-8 haszná lata ajánlott a legtöbb esetben, de néha szükség lehet konverzióra más kódolások között (pl. iconv() vagy mb_convert_encoding() függvényekkel).

A sztringek hatékony kezelése kulcsfontosságú a legtöbb PHP alkalmazásban. A fenti technikák, függvények és szempontok ismerete segít a tisztább, hatékonyabb és biztonsá gosabb kód írásában.

2.3. Operátorok

Az operátorok a PHP-ben kulcsfontosságú elemek, amelyek lehetővé teszik a programo zók számára, hogy különböző műveleteket végezzenek el változókon és értékeken. Az operátorok segítségével számításokat végezhetünk, összehasonlíthatunk értékeket, logi kai műveleteket hajthatunk végre, és még sok más műveletet is elvégezhetünk. A PHP számos operátort kínál, amelyek különböző kategóriákba sorolhatók funkcióik alapján.

2.3.1. Aritmetikai operátorok

Az aritmetikai operátorok a legalapvetőbb matematikai műveleteket teszik lehetővé. Ezek az operátorok számokkal dolgoznak, és a matematikában megszokott módon működnek.

- + (összeadás)
- - (kivonás)
- * (szorzás)
- / (osztás)
- % (maradékos osztás)
- ** (hatványozás)

Az aritmetikai operátorok használata egyszerű és intuitív, de fontos figyelni az

ope randusok típusára, különösen osztás esetén.

Példa:

```
$a = 10;
$b = 3;
echo $a + $b ; // Kimenet : 13
echo $a % $b ; // Kimenet : 1 (10 osztva 3 -mal , a marad ék 1) echo $a ** $b ; //
Kimenet : 1000 (10 a harmadikon )
```

2.13. Kódrészlet. Aritmetikai operátorok használata

2.3.2. Hozzárendelő operátorok

A hozzárendelő operátorok értéket adnak a változóknak. Az egyszerű értékadáson túl léteznek összetett hozzárendelő operátorok is, amelyek egy műveletet és egy értékadást kombinálnak.

- = (egyszerű hozzárendelés)
- +=, -=, *=, /=, %= (művelet és hozzárendelés)
- .= (összefűzés és hozzárendelés stringeknél)

Ezek az operátorok különösen hasznosak, amikor egy változó értékét annak aktuális értéke alapján módosítjuk.

Példa:

```
$a = 5;
$a += 3; // Ekvivalens : $a = $a + 3;
echo $a ; // Kimenet : 8

$str = " Hello ";
$str .= " World "; // String ö sszef űzés és hozz á rendel és echo $str ; //
Kimenet : " Hello World "
```

2.14. Kódrészlet. Hozzárendelő operátorok használata

2.3.3. Logikai operátorok

A logikai operátorok boolean értékekkel dolgoznak, és gyakran használatosak feltételes szerkezetekben. Ezek az operátorok lehetővé teszik komplex feltételek kialakítását.

- && (és)
- || (vagy)
- ! (nem)

Fontos megjegyezni, hogy a && és || operátorok rövid kiértékelést alkalmaznak, ami azt jelenti, hogy amint az eredmény ismert, a további kiértékelés leáll.

Példa:

```
$a = true ;
$b = false ;
var_dump ( $a && $b ) ; // Kimenet : bool ( false )
var_dump ( $a || $b ) ; // Kimenet : bool ( true )
var_dump (! $a ) ; // Kimenet : bool ( false )
```

2.15. Kódrészlet. Logikai operátorok használata

2.3.4. Összehasonlító operátorok

Az összehasonlító operátorok két érték összehasonlítására szolgálnak. Ezek az operátorok boolean értéket adnak vissza, ami jelzi, hogy az összehasonlítás igaz vagy hamis.

- == (egyenlő)
- === (azonos)
- != vagy <> (nem egyenlő)
- !== (nem azonos)
- <, >, <=, >= (kisebb, nagyobb, kisebb egyenlő, nagyobb

egyenlő) Fontos kiemelni a == és === közötti különbséget:

- == összehasonlítja az értékeket, de figyelmen kívül hagyja a

  típust • === összehasonlítja az értékeket és a típust is

Ez a különbség kritikus lehet bizonyos helyzetekben, különösen amikor különböző tí pusú értékeket hasonlítunk össze.

Példa:

```
$a = 5;
$b = "5";
var_dump ( $a == $b ) ; // Kimenet : bool ( true ) - az érték egyezik var_dump ( $a === $b ) ;
// Kimenet : bool ( false ) - a típus különbözik var_dump ( $a != "6") ; // Kimenet : bool ( true )
var_dump ( $a !== 5) ; // Kimenet : bool ( false )
```

2.16. Kódrészlet. Összehasonlító operátorok használata

Összehasonlító függvények: empty(), is_null(), isset()

A PHP több beépített függvényt kínál változók értékének és állapotának ellenőrzésére. Ezek a függvények különösen hasznosak feltételes szerkezetekben és adatvalidációnál.

- empty(): Ellenőrzi, hogy egy változó "üres"-e.
- is_null(): Ellenőrzi, hogy egy változó null értékű-e.

• isset(): Ellenőrzi, hogy egy változó be van-e állítva és nem null.

Fontos megjegyezni, hogy ezek a függvények eltérően viselkedhetnek különböző típusú és értékű változókkal.

Példa:

```
$emptyString = "";
$nullValue = null ;
$zeroInt = 0;
$falseBool = false ;
$nonEmptyString = " Hello ";

// empty () függv ény
var_dump ( empty ( $emptyString ) ) ; // bool ( true )
var_dump ( empty ( $nullValue ) ) ; // bool ( true )
var_dump ( empty ( $zeroInt ) ) ; // bool ( true )
var_dump ( empty ( $falseBool ) ) ; // bool ( true )
var_dump ( empty ( $nonEmptyString ) ) ; // bool ( false )

// is_null () függv ény
var_dump ( is_null ( $emptyString ) ) ; // bool ( false )
var_dump ( is_null ( $nullValue ) ) ; // bool ( true )
var_dump ( is_null ( $zeroInt ) ) ; // bool ( false )

// isset () függv ény
var_dump ( isset ( $emptyString ) ) ; // bool ( true )
var_dump ( isset ( $nullValue ) ) ; // bool ( false )
var_dump ( isset ( $undefinedVar ) ) ; // bool ( false )
```

2.17. Kódrészlet. Összehasonlító függvények használata

Fontos különbségek és megjegyzések:

- Az empty() függvény true értéket ad vissza üres sztringre, null értékre, 0 számér tékre és false logikai értékre is.
- Az is_null() csak akkor ad true értéket, ha a változó értéke pontosan null.
- Az isset() false értéket ad null értékre és nem definiált változókra, de true-t ad üres sztringre és 0 értékre.

Ezeknek a függvényeknek az eltérő viselkedése miatt fontos, hogy pontosan ismerjük a használatukat és azt, hogy milyen esetekben mely függvény a legalkalmasabb az adott ellenőrzés elvégzésére.

2.3.5. Feltételes operátor

A feltételes (ternáris) operátor egy kompakt módja az if-else szerkezet használatának. Ez az operátor különösen hasznos egyszerű feltételes értékadásoknál.

Szintaxis: feltétel ? érték_ha_igaz : érték_ha_hamis

Bár a feltételes operátor tömör és hatékony lehet, túlzott használata ronthatja a kód olvashatóságát, ezért érdemes mértékkel alkalmazni.

Példa:


```
$age = 20;
$status = ( $age >= 18) ? " feln őtt" : " kiskor ú";
echo $status ; // Kimenet : " feln őtt"

// Egym ásba á gyazott haszn álat (kerü lend ő a tú lzott haszn á lata ) $result = $a > $b ? "a
nagyobb " : ( $a < $b ? "b nagyobb " : " egyenl őek") ;
```

2.18. Kódrészlet. Feltételes operátor használata

2.3.6. Null coalescing operátor

A null coalescing operátor (??) a PHP 7-ben került bevezetésre, és jelentősen leegysze rűsíti a null ellenőrzéseket. Ez az operátor különösen hasznos, amikor alapértelmezett értéket akarunk használni, ha egy változó nem létezik vagy null értékű.

Példa egyszerű használatra:

```
$name = $firstName ?? " Guest ";
echo $name ; // Ha $firstName nem lé tezik vagy null , " Guest " lesz kií rva

// Ez ekvivalens a kö vetkez ővel:
$name = isset ( $firstName ) ? $firstName : " Guest ";
```

2.19. Kódrészlet. Null coalescing operátor egyszerű használata

Ez az egyszerű példa jól mutatja az operátor alapvető használatát. Most nézzünk egy összetettebb példát:

```
$username = $_GET ['user '] ?? $_POST ['user '] ?? 'nobody '; // Ez ekvivalens
a kö vetkez ővel:
// $username = isset ( $_GET [ ' user ']) ? $_GET [ ' user '] : ( isset ( $_POST [ ' user ']) ?
    $_POST [ ' user '] : 'nobody ');

$data = $result ['data '] ?? [];
```

2.20. Kódrészlet. Null coalescing operátor összetettebb használata

Ebben a példában látható, hogyan lehet az operátort használni többszörös ellenőrzésre vagy alapértelmezett érték beállítására összetettebb helyzetekben.

2.3.7. Spaceship operátor

A spaceship operátor (<=>), amely szintén a PHP 7-ben került bevezetésre, egy háromutas összehasonlító operátor. Ez az operátor különösen hasznos rendezési függvényekben és komplex összehasonlításoknál.

Az operátor összehasonlítja a bal és jobb oldali értékeket, és:

- -1-et ad vissza, ha a bal oldali érték kisebb
- 0-t ad vissza, ha a két érték egyenlő
- 1-et ad vissza, ha a bal oldali érték nagyobb

Példa:

```
echo 1 <= > 1; // Kimenet : 0 ( egyenl ő)
echo 1 <= > 2; // Kimenet : -1 (bal oldal kisebb )
echo 2 <= > 1; // Kimenet : 1 ( jobb oldal kisebb )

// Haszn álat rendez ésnél
$fruits = [’apple ’, ’banana ’, ’cherry ’];
usort ( $fruits , function ( $a , $b ) {
        return $a <= > $b ;
}) ;
print_r ( $fruits ) ;
  // Kimenet : Array ( [0] = > apple [1] = > banana [2] = > cherry )
```

2.21. Kódrészlet.

Spaceship operátor használata

Az operátorok helyes használata és megértése kulcsfontosságú a hatékony és olvasható PHP kód írásához. Minden operátornak megvan a maga specifikus használati területe, és a megfelelő operátor kiválasztása gyakran javíthatja a kód teljesítményét és érthetőségét.

2.4. Vezérlési szerkezetek

A vezérlési szerkezetek alapvető elemei a programozásnak, amelyek lehetővé teszik a prog ram végrehajtásának irányítását. PHP-ban számos vezérlési szerkezet áll rendelkezésre, amelyek segítségével feltételes végrehajtást, ciklusokat és más komplex programfolyama tokat valósíthatunk meg.

2.4.1. Utasítás és utasításblokk

A PHP programokban az utasítások az alapvető végrehajtási egységek. Egy utasítás általában egy műveletet hajt végre, és pontosvesszővel zárul. Az utasításblokk több utasítás csoportosítására szolgál, és kapcsos zárójelek közé zárjuk.

Szintaxis:

utasítás;

```
{
    utasítás1;
    utasítás2;
    // ...
}
```

Példa:

```
$total = 0; // Egyetlen utas ítás
```

```
{
        $price = 100;
        $tax = 0.2;
        $total = $price * (1 + $tax ) ;
} // Utas ítá sblokk
echo "Végö sszeg : $total ";
```

2.22. Kódrészlet. Utasítás és utasításblokk példa

2.4.2. If szerkezetek

Az if szerkezet lehetővé teszi a feltételes végrehajtást. PHP-ban többféle if szerkezetet használhatunk a program logikájának kialakítására.

Szintaxis:

```
// Egyszerű if
if (feltétel) {
        // kód, ha a feltétel igaz
}

// Kétágú if-else
if (feltétel) {
        // kód, ha a feltétel igaz
} else {
        // kód, ha a feltétel hamis
}

// Többágú if-elseif-else
if (feltétel1) {
        // kód, ha feltétel1 igaz
} elseif (feltétel2) {
        // kód, ha feltétel1 hamis, de feltétel2 igaz
} else {
        // kód, ha mindkét feltétel hamis
}
```

Példák:

```
$age = 18;
if ( $age >= 18) {
        echo "Ön nagykor ú.";
}
```

2.23. Kódrészlet. Egyszerű if szerkezet

```
$balance = 1000;
$withdrawal = 1200;
if ( $balance >= $withdrawal ) {
        echo " Sikeres tranzakci ó.";
        $balance -= $withdrawal ;
} else {
```

```
        echo " Nincs elegend ő fedezet .";
}
```

2.24. Kódrészlet. Kétágú if-else szerkezet

```
$grade = 75;
if ( $grade >= 90) {
        echo " Jeles ";
} elseif ( $grade >= 80) {
        echo "Jó";
} elseif ( $grade >= 70) {
```



```
        echo "Kö zepes ";
} elseif ( $grade >= 60) {
        echo "Elégséges ";
} else {
        echo "Elé gtelen ";
}
```

2.25. Kódrészlet. Többágú if-elseif-else szerkezet

Az egymásba ágyazott if szerkezetek lehetővé teszik komplex döntési fák kialakítását, de túlzott használatuk ronthatja a kód olvashatóságát. Íme egy példa:

```
$username = " user123 ";
$password = " pass123 ";
$isAdmin = false ;

if ( $username === " user123 ") {
        if ( $password === " pass123 ") {
                echo " Sikeres bejelentkez és. ";
                if ( $isAdmin ) {
                        echo "Üdvözöljük az admin fel ü leten !";
                } else {
                        echo "Üdvözöljük a felhaszn álói felü leten !";
                }
        } else {
                echo " Helytelen jelsz ó.";
        }
} else {
        echo " Felhaszn áló nem tal á lhat ó.";
}
```

2.26. Kódrészlet. Egymásba ágyazott if szerkezetek

Megjegyzés: Az egymásba ágyazott if szerkezetek használata ronthatja a kód olvas hatóságát. Komplex feltételek esetén érdemes megfontolni a kód átstrukturálását vagy függvények használatát.

2.4.3. Switch utasítás

A switch utasítás egy elegáns módja annak, hogy egy változó értékét összehasonlítsuk több lehetséges értékkel. Ez különösen hasznos, amikor sok lehetséges elágazást kell kezelnünk.

Szintaxis:

switch (kifejezés) {

```
    case érték1:
        // kód, ha kifejezés == érték1
        break;
    case érték2:
        // kód, ha kifejezés == érték2
        break;
    // ...
    default:
        // kód, ha egyik eset sem teljesül
}
```


Lássunk egy egyszerű példát:

```
$szin = " piros ";
switch ( $szin ) {
    case " piros ":
        echo "A szín piros .";
        break ;
    case "kék":
        echo "A szín kék.";
        break ;
    default :
        echo "A szín nem piros és nem kék.";
}
// Kimenet : A szín piros .
```

2.27. Kódrészlet. Egyszerű switch utasítás

Fontos megjegyezni, hogy a break utasítás nélkül a végrehajtás folytatódik a követke ző case-re. Ez lehetővé teszi több eset összevonását, ahogy az alábbi példában is látható:

```
$day = "hétfő";
switch ( $day ) {
    case "hétfő":
        echo "A hét els ő munkanapja .";
        break ;
    case " kedd ":
    case " szerda ":
    case "csütörtök":
        echo "Hétkö znap van .";
        break ;
    case "pé ntek ":
        echo "A hétvége kö zeleg !";
        break ;
    case " szombat ":
    case "vas á rnap ":
        echo "Hétvége van !";
        break ;
    default :
        echo "Érvé nytelen nap .";
}
```

2.28. Kódrészlet. Switch utasítás használata

2.4.4. A while ciklus

A while ciklus lehetővé teszi, hogy egy kódblokkot ismételjünk, amíg egy feltétel igaz.

Szintaxis:

```
while (feltétel) {
    // ismétlendő kód
}
```

Egyszerű példa:

```
$i = 1;
while ( $i <= 5) {
    echo $i . " ";
    $i ++;
}
// Kimenet : 1 2 3 4 5
```

2.29. Kódrészlet. Egyszerű while ciklus

Összetettebb példa:

```
$attempts = 0;
$max_attempts = 5;
while ( $attempts < $max_attempts ) {
    echo "Kísé rlet : " . ( $attempts + 1) . "\n";
    // Szimul á ljuk a sikertelen próbá lkoz ást
    $success = ( rand (0 , 1) == 1) ;
    if ( $success ) {
        echo " Sikeres mű velet !\n";
        break ;
    }
    $attempts ++;
}
if ( $attempts == $max_attempts ) {
    echo "Elérte a maxim ális próbá lkoz ások számát.";
}
```

2.30. Kódrészlet. Összetettebb while ciklus

2.4.5. A do...while ciklus

A do...while ciklus hasonló a while ciklushoz, de a feltétel ellenőrzése a ciklusmag végre hajtása után történik, így a ciklusmag legalább egyszer mindenképpen lefut.

Szintaxis:

```
do {
    // ismétlendő kód
} while (feltétel);
```

Példa:

```
$pin = "";
do {
        $pin = readline (" Adja meg a PIN kódot: ") ;
        if ( $pin != " 1234 ") {
                echo " Helytelen PIN . Próbálja újra .\n";
        }
} while ( $pin != " 1234 ") ;
echo " Sikeres bel épés!";
```

2.31. Kódrészlet. do...while ciklus használata

2.4.6. For ciklus

A for ciklus egy kompakt módot biztosít az ismétlődések végrehajtására, ahol az inicia lizálás, a feltétel és a léptetés egy sorban van megadva.

Szintaxis:


```
for (inicializálás; feltétel; léptetés) {
        // ismétlendő kód
}
```

A for ciklus három fő részből áll:

1. **Inicializálás:** Itt adjuk meg a kezdőértéket a ciklusváltozónak. Ez a rész csak egyszer fut le, a ciklus kezdetén.
2. **Feltétel:** Ez határozza meg, hogy a ciklus folytatódjon-e. Minden iteráció előtt ellenőrzésre kerül.
3. **Léptetés:** Ez a rész minden iteráció végén fut le, általában itt növeljük vagy csökkentjük a ciklusváltozót.

Egyszerű példa: számok kiírása növekvő sorrendben

```
for ( $i = 1; $i <= 5; $i ++) {
        echo $i . " ";
}
// Kimenet : 1 2 3 4 5
```

2.32. Kódrészlet. Egyszerű for ciklus

Csökkenő sorrendben (visszafele léptetés) való kiíratás:

```
for ( $i = 5; $i >= 1; $i - -) {
        echo $i . " ";
}
// Kimenet : 5 4 3 2 1
```

2.33. Kódrészlet. For ciklus visszafele léptetéssel

Összetettebb példa:

```
$total = 0;
$items = [10 , 20 , 30 , 40 , 50];
for ( $i = 0; $i < count ( $items ) ; $i ++) {
```

```
        $total += $items [ $i ];
        echo "Részö sszeg a(z) " . ( $i + 1) . ". elem után: $total \n"; }
echo "Végö sszeg : $total ";
```

2.34. Kódrészlet. Összetettebb for ciklus

2.4.7. Foreach ciklus

A foreach ciklus kifejezetten tömbök és objektumok bejárására szolgál. Ez a ciklus auto matikusan végigmegy a tömb vagy objektum elemein.

Szintaxis:

```
foreach ($tömb as $érték) {
        // kód minden elemre
}
```



```
foreach ($tömb as $kulcs => $érték) {
        // kód minden kulcs-érték párra
}
```

A foreach ciklusnak két változata van:

1. Az első változat csak az értékeket adja vissza.
2. A második változat a kulcsokat és az értékeket is visszaadja.

Egyszerű példa:

```
$colors = [" piros ", "kék", "zöld"];
foreach ( $colors as $color ) {
        echo $color . " ";
}
// Kimenet : piros kék zöld
```

2.35. Kódrészlet. Egyszerű foreach ciklus

Összetettebb példa:

```
$cart = [
        " alma " = > ["ár" = > 200 , " mennyis ég" = > 3] ,
        "banán" = > ["ár" = > 150 , " mennyis ég" = > 2] ,
        " narancs " = > ["ár" = > 220 , " mennyis ég" = > 4]
];

$total = 0;
foreach ( $cart as $item = > $details ) {
        $itemTotal = $details ['ár'] * $details ['mennyis ég'];
        $total += $itemTotal ;
        echo " $item : { $details [ ' mennyis ég ']} db , ö sszesen $itemTotal RON\n" ;
}
echo "Kos ár össz értéke: $total RON";
```

2.36. Kódrészlet. Összetettebb foreach ciklus

2.4.8. Ciklusvezérlő utasítások

A PHP két fontos ciklusvezérlő utasítást kínál: break és continue. Ezek segítenek a ciklusok finomhangolásában és a végrehajtás irányításában.

- break: Azonnal kilép a ciklusból
- continue: A ciklus következő iterációjára ugrik

Példa:

```
$numbers = [1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10];
$sum = 0;
foreach ( $numbers as $number ) {
        if ( $number % 2 != 0) {
                continue ; // A pá ratlan szá mokat kihagyjuk
        }
        $sum += $number ;
```



```
        if ( $sum > 20) {
                break ; // Megállunk , ha az ö sszeg meghaladja a 20 - at
        }
}
echo "A páros számok ö sszege (max. 20 - ig): $sum ";
```

2.37. Kódrészlet. Ciklusvezérlő utasítások használata

2.4.9. Egymásba ágyazott ciklusok

A PHP lehetővé teszi ciklusok egymásba ágyazását, ami komplex iterációs struktúrák létrehozását teszi lehetővé.

Egyszerű példa egymásba ágyazott ciklusokra:

```
for ( $i = 1; $i <= 3; $i ++) {
        for ( $j = 1; $j <= 3; $j ++) {
                echo "($i , $j) ";
        }
        echo "<br >";
}
// Kimenet :
// (1 , 1) (1 , 2) (1 , 3)
// (2 , 1) (2 , 2) (2 , 3)
// (3 , 1) (3 , 2) (3 , 3)
```

2.38. Kódrészlet. Egyszerű egymásba ágyazott ciklusok

Összetettebb példa tömbbel:

```
$products = [
        "Gyümölcsök" = > [" alma ", "banán", " narancs "] ,
        "Zöldségek" = > ["répa", " brokkoli ", " paradicsom "] ,
        " Tejterm ékek" = > ["tej", " sajt ", " joghurt "]
];
```

```
foreach ( $products as $category = > $items ) {
        echo " $category : <br >";
        foreach ( $items as $index = > $item ) {
                echo " " . ( $index + 1) . ". $item <br >";
        }
        echo "<br >";
}
```

2.39. Kódrészlet. Egymásba ágyazott ciklusok tömbbel

Az egymásba ágyazott ciklusok különösen hasznosak többdimenziós adatstruktúrák kezelésénél, de óvatosan kell használni őket, mert növelhetik a kód komplexitását és futási idejét.

2.4.10. Javaslatok és megjegyzések a ciklusok használatához

1. **while vs. for**: A while ciklust akkor érdemes használni, amikor nem tudjuk előre, hányszor kell ismételni a műveletet (pl. felhasználói input ellenőrzése). A for ciklus ideális, amikor ismert számú iterációt kell végrehajtani.

2. **Foreach előnyben részesítése**: Tömbök és objektumok bejárásánál mindig része sítsük előnyben a foreach ciklust. Gyorsabb és biztonságosabb, mint a hagyományos for ciklus.

3. **Végtelen ciklusok elkerülése**: Győződjünk meg róla, hogy a ciklusnak van kilé pési feltétele. Különösen while ciklusoknál fontos erre figyelni.

4. **Teljesítmény**: Nagy adathalmazoknál vagy erőforrás-igényes műveleteknél figyel jünk a ciklusok hatékonyságára. Kerüljük a szükségtelen iterációkat és a cikluson belüli bonyolult számításokat.

5. **Ciklusvezérlő utasítások megfontolt használata**: A break és continue utasí tások hasznosak lehetnek, de túlzott használatuk ronthatja a kód olvashatóságát. Használjuk őket mértékkel és megfelelő kommentezéssel.

6. **Egymásba ágyazott ciklusok optimalizálása**: Ha lehetséges, próbáljuk meg elkerülni a mély egymásba ágyazásokat. Gyakran a kód átalakítható hatékonyabb formába.

7. **Iterátorok használata**: Nagy adathalmazoknál vagy komplex objektumoknál ér demes megfontolni az iterátorok használatát a foreach ciklus helyett, mert kevesebb memóriát használnak.

2.5. Függvények

A függvények a PHP programozás alapvető építőkövei. Lehetővé teszik a kód újrafel használását, a program strukturálását és a komplexitás kezelését. Ebben a fejezetben részletesen áttekintjük a PHP függvények különböző aspektusait, a szintaxistól kezdve a speciális függvénytípusokig.

2.5.1. Függvények alapjai

Függvények szintaxisa

A PHP-ban a függvényeket a function kulcsszóval definiáljuk. A függvény definíciója tartalmazza a függvény nevét, a paraméterlistát (ha van), és a függvény törzsét.

Szintaxis:

```
function függvényNév(paraméter1, paraméter2, ...)
{
      // függvény törzse
      return visszatérési_érték;
}
```

Példa:

```
function greeting ( $name ) {
      return " Hell ó, $name !";
}
```



```
echo greeting ("János ") ; // Kimenet : Hell ó, János!
```

2.40. Kódrészlet. Egyszerű függvény definíció

Ebben a példában a greeting függvény egy $name paramétert fogad, és egy üdvözlő üzenetet ad vissza. A függvényt a "János" argumentummal hívjuk meg.

Függvényekkel kapcsolatos fontosabb szabályok

A PHP függvények használatakor néhány fontos szabályt kell szem előtt

tartanunk: • A függvénynevek nem lehetnek foglalt szavak (pl. if, else, while).

- A függvénynevek nem különböztetik meg a kis- és nagybetűket. Például a myFunction() és a MYFUNCTION() ugyanarra a függvényre hivatkozik.
- A függvényeket általában a hívás előtt kell definiálni, kivéve ha feltételes definíció ban vannak.
- A függvények túlterhelése (overloading) nem támogatott PHP-ban, azaz nem lehet több azonos nevű függvényt definiálni különböző paraméterlistákkal.

Eljárás vs. függvény viselkedés

A PHP-ban nincs szintaktikai különbség eljárás és függvény között. A különbség a vissza térési értékben rejlik:

- Ha egy függvény nem ad vissza értéket (nincs return utasítás), akkor eljárásként viselkedik.

- Ha egy függvény visszaad értéket (return utasítással), akkor függvényként viselke dik.

Példa:

```
function procedureExample ( $msg ) {
        echo $msg ;
}

function functionExample ( $a , $b ) {
        return $a + $b ;
}

procedureExample (" Hell ó!") ; // Kiírja: Hell ó!
$result = functionExample (3 , 4) ; // $result értéke 7 lesz
```

2.41. Kódrészlet.

Eljárás és függvény viselkedés

Ebben a példában a procedureExample eljárásként viselkedik, mert csak kiír egy üzenetet, de nem ad vissza értéket. A functionExample viszont függvényként viselkedik, mert visszaad egy értéket, amit eltárolhatunk egy változóban.

2.5.2. Paraméterek és visszatérési értékek

Paraméterek használata

A függvények paramétereket fogadhatnak, amelyek lehetnek kötelezőek vagy opcionálisak. A paraméterek lehetővé teszik, hogy adatokat adjunk át a függvénynek feldolgozásra.

Példa:

```
function greet ( $name , $greeting = " Hell ó") {
        return " $greeting , $name !";
}

echo greet (" Anna ") ; // Kimenet : Hell ó, Anna !
  echo greet ("Péter ", "Jó napot ") ; // Kimenet : Jó napot , Péter!
```

2.42. Kódrészlet.

Paraméterek használata

Ebben a példában a greet függvény két paramétert fogad: egy kötelező $name para métert és egy opcionális $greeting paramétert alapértelmezett "Helló" értékkel.

Alapértelmezett paraméter érték

Az opcionális paramétereknek adhatunk alapértelmezett értéket. Ezeknek a paraméte reknek a függvény definíciójában jobbra kell állniuk a kötelező paraméterektől.

Példa:

```
function power ( $base , $exponent = 2) {
        return $base ** $exponent ;
}

echo power (3) ; // Kimenet : 9 (3^2)
echo power (2 , 3) ; // Kimenet : 8 (2^3)
```

2.43. Kódrészlet. Alapértelmezett paraméter érték

Itt a power függvény egy kötelező $base paramétert és egy opcionális $exponent paramétert fogad, amelynek alapértelmezett értéke 2.

Paraméter típusok megadása (type hinting)

PHP 7.0-tól kezdve lehetőségünk van a paraméterek típusának explicit megadására. Ez segít a kód olvashatóságában és a hibák korai felderítésében.

Példa:

```
function add ( int $a , int $b ) : int {
        return $a + $b ;
}

echo add (5 , 3) ; // Kimenet : 8
// echo add ("5" , "3") ; // Hiba : Argument 1 must be of type int , string given
```

2.44. Kódrészlet. Paraméter típusok megadása

Ebben a példában az add függvény két egész számot (int) vár paraméterként, és garantálja, hogy az eredmény is egész szám lesz.

Visszatérési érték típusa

A függvény visszatérési értékének típusát is megadhatjuk, ami tovább növeli a kód biz tonságát és olvashatóságát.

Példa:

```
function getName () : string {
        return "János ";
}

$name = getName () ;
var_dump ( $name ) ; // string (6) "Já nos"
```

2.45. Kódrészlet. Visszatérési érték típusa

A getName függvény garantálja, hogy mindig string típusú értéket ad vissza.

Strict mode

A strict mode bekapcsolásával a PHP szigorúbban ellenőrzi a típusokat, ami segít a típushibák elkerülésében.

Példa:

```
declare ( strict_types =1) ;

function add ( int $a , int $b ) : int {
        return $a + $b ;
}

echo add (5 , 3) ; // Működik
// echo add ("5" , "3") ; // Fatal error : Uncaught TypeError
```

2.46. Kódrészlet.

Strict mode használata

Strict módban a PHP nem végez automatikus típuskonverziót, így a string típusú "5" és "3" argumentumok használata hibát eredményez.

Referencia szerinti paraméter átadás

Alapértelmezetten a PHP érték szerint adja át a paramétereket. A referencia szerinti átadáshoz használjuk a & jelet, ami lehetővé teszi, hogy a függvény módosítsa az eredeti változó értékét.

Példa:

```
function increment (& $number ) {
        $number ++;
}

$a = 5;
increment ( $a ) ;
```



```
echo $a ; // Kimenet : 6
```

2.47. Kódrészlet. Referencia szerinti paraméter átadás

Ebben a példában az increment függvény módosítja az $a változó értékét a referencia szerinti paraméterátadás miatt.

2.5.3. Függvények hatóköre és változók láthatósága

Lokális változók

A függvényen belül definiált változók alapértelmezetten lokálisak, csak a függvényen belül érhetők el.

Példa:

```
function example () {
        $local = " Csak itt lá that ó";
        echo $local ;
}

example () ; // Kimenet : Csak itt lá that ó
// echo $local ; // Hiba : Undefined variable
```

2.48. Kódrészlet. Lokális változók

Globális változók

A global kulcsszóval elérhetünk globális változókat a függvényen belül.

Példa:

```
$x = 5;

function useGlobal () {
        global $x ;
        echo $x ;
}

useGlobal () ; // Kimenet : 5
```

2.49. Kódrészlet. Globális változók használata

Statikus változók

A static kulcsszóval definiált változók megtartják értéküket a függvényhívások között.

Példa:

```
function counter () {
        static $count = 0;
        return ++ $count ;
}

echo counter () ; // Kimenet : 1
```



```
echo counter () ; // Kimenet : 2
echo counter () ; // Kimenet : 3
```

2.50. Kódrészlet. Statikus változók

2.5.4. Speciális függvénytípusok

Névtelen függvények

A névtelen függvények (anonim függvények) olyan függvények, amelyeknek nincs nevük. Használhatók változóhoz rendelésre, függvény paraméterként, vagy visszatérési értékként.

Példa:

```
$greet = function ( $name ) {
        return " Hell ó, $name !";
};

echo $greet (" Anna ") ; // Kimenet : Hell ó, Anna !
```

2.51. Kódrészlet. Névtelen függvény változóhoz rendelve

A névtelen függvényeket használhatjuk callback-ként:

```
$numbers = [1 , 2 , 3 , 4 , 5];
$evens = array_filter ( $numbers , function ( $n ) {
        return $n % 2 == 0;
}) ;
print_r ( $evens ) ; // Array ( [1] = > 2 [3] = > 4 )
```

2.52. Kódrészlet. Névtelen függvény callback-ként

Closure

A closure egy speciális névtelen függvény, amely hozzáfér a környezeti változókhoz (glo bális hatókör), akkor is, ha azok kívül esnek a függvény hatókörén.

Szintaxis:

```
$closure = function([paraméterek]) use ([külső változók]) { //
        Függvény törzse
};
```

Ahol:

- **function**: Ez az anonim függvény deklarálása.
- **paraméterek**: Opcionális. Ezek a függvény által fogadott paraméterek. • **use**:

  Ez a kulcsszó biztosítja, hogy a függvény hozzáférjen a külső változókhoz. •

  függvény törzse: Itt hajtódik végre a closure logikája.

Példa:


```
// Egy vá ltoz ó a külső kö rnyezetben
$message = 'Hello ';

// Egy closure , ami hozz áfér a külső vá ltoz óhoz
$greet = function () use ( $message ) {
        echo $message ;
};

$greet () ; // Kimenet : Hello
```

2.53. Kódrészlet. Closure példa

A fenti példában a use kulcsszóval érjük el a külső hatókörben lévő $message változót, így az elérhető lesz a closure számára.

A closure függvények paramétereket is fogadhatnak, ahogyan azt az alábbi példában látjuk:

```
$multiplier = 2;
```

```php
$multiply = function ( $value ) use ( $multiplier ) {
        return $value * $multiplier ;
};

echo $multiply (5) ; // Kimenet : 10
```

2.54. Kódrészlet. Closure paraméterekkel

A Closure előnyei:

- **Callback-ek**: A closure-k különösen hasznosak callback függvényként, például az array_map() esetében.
- **Állapotmegőrzés**: A closure-k lehetővé teszik a környezeti változók megőrzését és módosítását.
- **Egyszerűség**: Lehetővé teszik a függvények dinamikus, rövid formában történő definiálását.

Arrow function

Az arrow function (lambda függvény) a closure-ök rövidebb, tömörebb szintaxisa, amely PHP 7.4-től érhető el és automatikusan öröklik a külső változókat a use kulcsszó hasz nálata nélkül

Szintaxis:

$lambdaFüggvény = fn(paraméterek) => kifejezés;

Ahol:

- **fn**: Az arrow function kulcsszava.
- **paraméterek**: A függvény által fogadott paraméterek (opcionális). •
- **=>**: A nyíl operátor, amely elválasztja a paramétereket a kifejezéstől.

- **kifejezés**: Egyetlen kifejezés, amelynek értéke lesz a függvény visszatérési

értéke. Az arrow function használatának szabályai és tulajdonságai:

- Mindig egyetlen kifejezést tartalmaznak, amely egyben a visszatérési

érték. • Nincs szükség explicit return kulcsszóra.

- Automatikusan öröklik a külső hatókör változóit érték szerint, nincs szükség use kulcsszóra.
- Nem módosíthatják a külső változók értékét.
- Nem tartalmazhatnak utasításokat vagy többsoros kódot.
- Használhatók osztálymetódusokban, és hozzáférnek a $this változóhoz.

Példa:

```
$message = 'Hello ';

$greet = fn ( $name ) = > $message . ', ' . $name ;

echo $greet ('World ') ; // Kimenet : Hello , World
```

2.55. Kódrészlet. Arrow function példa

A fenti példában a $message változó automatikusan elérhető az arrow függvényben anélkül, hogy explicit módon át kellene adni a use kulcsszóval.

A következő példában egy tipikus használatát láthatjuk mint callback függvény:

```
$numbers = [1 , 2 , 3 , 4 , 5];
$multiplier = 2;

$doubled = array_map ( fn ( $n ) = > $n * $multiplier , $numbers ) ; print_r ( $doubled ) ; // Array (
[0] = > 2 [1] = > 4 [2] = > 6 [3] = > 8 [4] = > 10 )

// Ö sszehasonl ítás a hagyom á nyos closure szintaxissal :
$doubledClosure = array_map ( function ( $n ) use ( $multiplier ) { return $n *
    $multiplier ;
} , $numbers ) ;
```

2.56. Kódrészlet. Arrow függvény mint callback

Az arrow függvények különösen hasznosak olyan helyzetekben, ahol rövid, egyszerű callback függvényekre van szükség, például tömbműveleteknél vagy eseménykezelőknél. Tömörségük és olvashatóságuk miatt gyakran előnyben részesítik őket a hagyományos anonim függvényekkel szemben egyszerű műveletek esetén.

Variadic függvény

A variadic függvények olyan függvények, amelyek képesek változó számú argumentumot fogadni. PHP 5.6-tól kezdve a három pont (...) operátorral lehet ilyen függvényeket definiálni.

Szintaxis:


```
function functionName(...$params) {
    // function body
}
```

Ahol:

- **...**: A variadic operátor, amely jelzi, hogy a függvény változó számú argumentumot fogad.
- **$params**: Egy tömb, amely tartalmazza az összes átadott argumentumot.

Példa:

```php
function sum (... $numbers ) {
    return array_sum ( $numbers ) ;
}

echo sum (1 , 2 , 3 , 4) ; // Kimenet : 10
echo sum (5 , 10 , 15) ; // Kimenet : 30
```

2.57. Kódrészlet. Egyszerű variadic függvény

A variadic függvények használata különösen ajánlott a következő esetekben:

1. **Ismeretlen számú paraméter kezelése**: Amikor előre nem tudható, hány argu mentumot fog kapni a függvény.

```php
function concatenate (... $strings ) {
    return implode (', ', $strings ) ;
}

echo concatenate ('alma ', 'ban án', 'citrom ') ; // Kimenet : alma , banán, citrom
echo concatenate ('egy ', 'kett ő') ; // Kimenet : egy , kett ő
```

2.58. Kódrészlet. Ismeretlen számú paraméter kezelése

2. **Függvényhívások továbbítása**: Amikor egy függvény az összes kapott argumen tumot tovább akarja adni egy másik függvénynek.

```php
function log_and_run ( $func , ... $args ) {
    echo " Calling function with " . count ( $args ) . " arguments .\n";
    return $func (... $args ) ;
}

function multiply ( $a , $b ) {
    return $a * $b ;
}

echo log_and_run ('multiply ', 4 , 5) ;
// Kimenet :
// Calling function with 2 arguments .
// 20
```

2.59. Kódrészlet. Függvényhívások továbbítása

3. **Opcionális paraméterek kezelése**: Amikor néhány kötelező paraméter után tet szőleges számú opcionális paramétert szeretnénk fogadni.

```php
function makeList ( $listType , ... $items ) {
    $html = "<$listType >\n";
    foreach ( $items as $item ) {
        $html .= " <li >$item </li >\n";
    }
    $html .= " </ $listType >";
    return $html ;
}

echo makeList ('ul ', 'Apple ', 'Banana ', 'Cherry ') ;
// Kimenet :
// <ul >
// <li >Apple </li >
```

```
// <li >Banana </li >
// <li >Cherry </li >
// </ul >
```

2.60. Kódrészlet. Opcionális paraméterek kezelése

A variadic függvények használatának előnyei:

- Rugalmasabb függvénydefiníciók
- Tisztább kód, nincs szükség a func_get_args() függvényre
- Jobb olvashatóság és önmagyarázó kód
- Könnyebb paraméter továbbítás más függvényeknek

Fontos megjegyezni, hogy a variadic paraméternek mindig az utolsónak kell lennie a paraméterlistában. Emellett, ha típusdeklarációt használunk, az a tömb minden elemére vonatkozni fog.

```
function sumIntegers ( int ... $numbers ) {
        return array_sum ( $numbers ) ;
}

echo sumIntegers (1 , 2 , 3) ; // Működik
// echo sumIntegers (1 , 2 , '3 '); // TypeError : Argument 3 must be of type int , string given
```

2.61. Kódrészlet. Típusdeklaráció variadic függvénynél

2.6. Tömbök

A tömbök a PHP egyik legalapvetőbb és legsokoldalúbb adatstruktúrái. Lehetővé teszik különböző típusú adatok tárolását. A tömbök használata rendkívül elterjedt a PHP programozásban, mivel rugalmas módot biztosítanak az adatok szervezésére és kezelésére.

2.6.1. Tömbök fajtái

PHP-ban két fő tömbtípus létezik, amelyeket különböző esetekben használhatjuk:

- **Indexelt tömbök**: Olyan tömbök, ahol a kulcsok automatikusan hozzárendelt egész számok. Ezek a tömbök hasonlítanak a legtöbb programozási nyelv hagyomá nyos tömbjeihez. Indexelt tömböket akkor használunk, amikor az elemek sorrendje fontos, és nincs szükség egyedi azonosítókra az elemekhez.
- **Asszociatív tömbök**: Olyan tömbök, amelyek kulcs-érték párokat tárolnak. Ez azt jelenti, hogy minden értékhez egy egyedi kulcs tartozik, amely segítségével könnyen hozzáférhetünk az adott értékhez. A kulcsok lehetnek string-ek vagy szá mok, míg az értékek bármilyen típusúak lehetnek (string, integer, float,

boolean, vagy akár más tömbök vagy objektumok).

Fontos megjegyezni, hogy PHP-ban valójában minden tömb asszociatív a háttérben, az indexelt tömbök csupán speciális esetei az asszociatív tömböknek, ahol a kulcsok egész számok.

2.6.2. Indexelt tömbök

Szintaxis és létrehozás

Indexelt tömböket többféleképpen hozhatunk létre. A rövidített szintaxis (szögletes zá rójelek használata) PHP 5.4-től kezdve érhető el, és általában ezt preferálják a modern PHP kódokban az olvashatóság és tömörség miatt. A hosszabb (klasszikus), array() függvényt használó szintaxis továbbra is érvényes, és régebbi kódokban gyakran találkoz hatunk vele.

Példa:

```
// Rövid í tett szintaxis
$fruits = [" apple ", " banana ", " cherry "];

// Hosszabb szintaxis
$vegetables = array (" carrot ", " broccoli ", " spinach ") ;

// Üres tömb lé trehoz ása és elemek hozz áadása
$colors = [];
$colors [] = "red";
$colors [] = " green ";
$colors [] = " blue ";
```

2.62. Kódrészlet. Indexelt tömbök létrehozása

Elemek elérése

Az indexelt tömbök elemeit a kulcsukkal (index) érhetjük el. Az indexelés 0-tól kezdődik, ami azt jelenti, hogy az első elem indexe 0, a másodiké 1, és így tovább:

```
echo $fruits [0]; // Kimenet : apple
echo $vegetables [1]; // Kimenet : broccoli
```

2.63. Kódrészlet. Indexelt tömbök elemeinek elérése

Fontos megjegyezni, hogy ha olyan indexet próbálunk elérni, ami nem létezik a tömb ben, akkor PHP Notice hibát kapunk. Ezért gyakran használjuk az isset() függvényt annak ellenőrzésére, hogy egy adott index létezik-e, mielőtt megpróbálnánk elérni:

```
if ( isset ( $fruits [3]) ) {
        echo $fruits [3];
} else {
        echo "A negyedik elem nem lé tezik .";
}
```

2.64. Kódrészlet. Biztonságos elem elérés

Tömbelemek törlése

Az unset() függvény használható egy vagy több tömbelem törlésére. Fontos megjegyezni, hogy az unset() nem indexeli újra a tömböt automatikusan.

Példa:

```
$fruits = [" apple ", " banana ", " cherry ", " date "];
unset ( $fruits [1]) ; // Törli a " banana " elemet
print_r ( $fruits ) ;
// Kimenet :
// Array
// (
// [0] = > apple
// [2] = > cherry
// [3] = > date
// )
```

2.65. Kódrészlet. Tömbelemek törlése unset() függvénnyel

Ha szükséges a tömb újraindexelése a törlés után, használhatjuk az array_values() függvényt:

```
$fruits = [" apple ", " banana ", " cherry ", " date "];
unset ( $fruits [1]) ;
$fruits = array_values ( $fruits ) ; // Ú jraindexel és
print_r ( $fruits ) ;
// Kimenet :
// Array
// (
// [0] = > apple
// [1] = > cherry
// [2] = > date
// )
```

2.66. Kódrészlet. Tömb újraindexelése törlés után

2.6.3. Asszociatív tömbök

Az asszociatív tömbök a PHP egyik leghatékonyabb eszközei, amellyel kulcs-érték párokat tárolhatunk.

Szintaxis

```
$tomb = array(
    "kulcs1" => "érték1",
    "kulcs2" => "érték2",
    // ... további kulcs-érték párok
);
```

Ahol:

- **array()** vagy []: Ez a konstrukció létrehoz egy új tömböt.
- **Kulcsok:** Bármely string vagy egész szám lehet, amelyek egyedileg azonosítják az értékeket. A kulcsoknak egyedinek kell lenniük a tömbön belül.
- **Értékek:** Bármilyen típusú adat lehet, például string, integer, float, boolean, vagy akár más tömb vagy objektum.

Példa:

```
// Rövid í tett szintaxis
$person = [
        " name " = > " John Doe",
        "age" = > 30 ,
        " city " = > "New York "
];

// Hosszabb szintaxis
$car = array (
        " brand " = > " Toyota ",
        " model " = > " Corolla ",
        " year " = > 2020
) ;
```

2.67. Kódrészlet. Asszociatív tömbök létrehozása

Az asszociatív tömbök különösen hasznosak, amikor komplex adatstruktúrákat aka runk létrehozni. Például egy személy vagy egy termék tulajdonságainak tárolására kivá lóan alkalmasak.

Elemek elérése

Az asszociatív tömbök elemeit a kulcsukkal érhetjük el. Ez intuitívabb módja lehet az adatok elérésének, mint az indexelt tömbök esetében:

```
echo $person [" name "]; // Kimenet : John Doe
echo $car [" brand "]; // Kimenet : Toyota
```

2.68. Kódrészlet. Asszociatív tömbök elemeinek elérése

Akárcsak az indexelt tömböknél, itt is fontos ellenőrizni a kulcs létezését az isset() vagy az array_key_exists() függvényekkel:

```
if ( array_key_exists ("age ", $person ) ) {
        echo "A szem ély é letkora : " . $person ["age"];
} else {
```



```
        echo "Az é letkor nincs megadva .";
}
```

2.69. Kódrészlet. Biztonságos elem elérés asszociatív tömbben

Az array_key_exists() függvény különösen hasznos lehet, ha kifejezetten a kulcs létezését akarjuk ellenőrizni, függetlenül attól, hogy az értéke null vagy sem.

2.6.4. Tömbök bejárása

A tömbök bejárása alapvető művelet a programozásban, és PHP-ban több módszer is rendelkezésünkre áll erre a célra.

Indexelt tömbök bejárása

Indexelt tömböket általában a for vagy foreach ciklussal járhatjuk be:

```
// for ciklussal
for ( $i = 0; $i < count ( $fruits ) ; $i ++) {
        echo $fruits [ $i ] . "<br >";
}

// foreach ciklussal
foreach ( $fruits as $fruit ) {
        echo $fruit . "<br >";
}
```

2.70. Kódrészlet. Indexelt tömbök bejárása

A for ciklus használata akkor lehet előnyös, ha szükségünk van az index értékére a cikluson belül. Azonban a foreach ciklus általában olvashatóbb és kevésbé hajlamos hibákra, különösen ha nem ismerjük a tömb pontos méretét.

Asszociatív tömbök bejárása

Asszociatív tömböket általában foreach ciklussal járjuk be, mivel ez lehetővé teszi mind a kulcs, mind az érték egyszerű elérését:

```
foreach ( $person as $key = > $value ) {
        echo " $key : $value <br >";
}
```

2.71. Kódrészlet. Asszociatív tömbök bejárása

Ez a módszer különösen hasznos, mert nem kell előre ismernünk a tömb kulcsait, és minden elemet egyszerűen elérhetünk.

2.6.5. Hasznos tömbfüggvények

PHP számos beépített függvényt kínál a tömbök kezelésére, amelyek megkönnyítik a prog ramozók munkáját. Íme néhány gyakran használt függvény részletesebb magyarázattal:

- count(): Megszámolja a tömb elemeit. Hasznos, amikor tudnunk kell, hány elem van egy tömbben, például ciklusok használata előtt.

- array_push(): Elemet ad hozzá a tömb végéhez. Ez a függvény különösen hasznos, amikor dinamikusan építünk fel egy tömböt.

- array_pop(): Eltávolítja és visszaadja a tömb utolsó elemét. Gyakran használják

verem (stack) adatstruktúra implementálásához.

- array_shift(): Eltávolítja és visszaadja a tömb első elemét. Ez a függvény hasznos lehet sor (queue) adatstruktúra implementálásához.
- array_unshift(): Elemet ad hozzá a tömb elejéhez. Ez megváltoztatja az összes meglévő numerikus kulcsot, ezért óvatosan kell használni.
- array_merge(): Összefűz két vagy több tömböt. Ez a függvény különösen hasznos, amikor több forrásból származó adatokat kell egyesíteni.
- in_array(): Ellenőrzi, hogy egy érték szerepel-e a tömbben. Ez a függvény hasznos lehet adatok validálásánál.
- array_key_exists(): Ellenőrzi, hogy egy kulcs létezik-e a tömbben. Ez biztonsá gosabb módja a kulcs létezésének ellenőrzésére, mint az isset(), különösen ha a kulcs értéke lehet null.

Ezek a függvények jelentősen megkönnyítik a tömbökkel való munkát, és sok eset ben hatékonyabb megoldást kínálnak, mint ha saját magunk implementálnánk ezeket a funkciókat.

2.6.6. Tömbök rendezése

A tömbök rendezése gyakori művelet a programozásban, és a PHP több beépített függ vényt is kínál erre a célra. Ezek a függvények különböző módon rendezik a tömböket, attól függően, hogy milyen típusú a tömb és milyen rendezési kritériumot szeretnénk alkalmazni:

- sort(): Növekvő sorrendbe rendezi a tömböt, és újraindexeli. Ez a függvény meg változtatja az eredeti tömböt.
- rsort(): Csökkenő sorrendbe rendezi a tömböt, és újraindexeli. Ez is megváltoz tatja az eredeti tömböt.
- asort(): Növekvő sorrendbe rendezi a tömböt az értékek alapján, megtartva a kulcs-érték megfeleltetéseket. Ez különösen hasznos asszociatív tömböknél.
- arsort(): Csökkenő sorrendbe rendezi a tömböt az értékek alapján, megtartva a kulcs-érték megfeleltetéseket.
- ksort(): Kulcsok szerint növekvő sorrendbe rendezi a tömböt. Ez hasznos lehet, ha az asszociatív tömb kulcsai szerint szeretnénk rendezni.
- krsort(): Kulcsok szerint csökkenő sorrendbe rendezi a tömböt.

Példa a rendezésre:


```
$numbers = [3 , 1 , 4 , 1 , 5 , 9 , 2 , 6 , 5 , 3 , 5];
sort ( $numbers ) ;
```

```
print_r ( $numbers ) ;
// Kimenet : Array ( [0] = > 1 [1] = > 1 [2] = > 2 [3] = > 3 [4] = > 3 [5] = > 4 [6] = > 5 [7] = > 5 [8] = > 5
    [9] = > 6 [10] = > 9 )

$fruits = [" banana " = > " yellow ", " apple " = > "red", " cherry " = > "red"]; asort ( $fruits ) ;
print_r ( $fruits ) ;
  // Kimenet : Array ( [ apple ] = > red [ cherry ] = > red [ banana ] = > yellow )
```

2.72. Kódrészlet.

Tömbök rendezése

Fontos megjegyezni, hogy ezek a függvények általában megváltoztatják az eredeti tömböt. Ha meg szeretnénk tartani az eredeti tömböt, érdemes először lemásolni azt.

2.6.7. array_map, array_walk, array_filter függvények

Ezek a függvények lehetővé teszik, hogy műveleteket végezzünk a tömb elemein, és külö nösen hasznosak funkcionális programozási stílusú kód írásánál.

array_map()

Az array_map() függvény egy megadott függvényt alkalmaz a tömb minden elemére, és visszaadja az eredményt egy új tömbben. Ez a függvény nem változtatja meg az eredeti tömböt.

```
$numbers = [1 , 2 , 3 , 4 , 5];
$squared = array_map ( function ( $n ) {
    return $n * $n ;
} , $numbers ) ;
print_r ( $squared ) ;
  // Kimenet : Array ( [0] = > 1 [1] = > 4 [2] = > 9 [3] = > 16 [4] = > 25 )
```

2.73. Kódrészlet.

array_map() használata

Az array_map() különösen hasznos, amikor minden tömbelem értékét ugyanazzal a művelettel szeretnénk módosítani. Gyakran használják adatok transzformálására vagy formázására.

array_walk()

Az array_walk() függvény egy felhasználó által megadott függvényt alkalmaz a tömb minden elemére, de az eredeti tömböt módosítja. Ez lehetővé teszi, hogy közvetlenül manipuláljuk a tömb elemeit.

```
$fruits = ["a" = > " apple ", "b" = > " banana "];
array_walk ( $fruits , function (& $value , $key ) {
    $value = strtoupper ( $value ) ;
}) ;
print_r ( $fruits ) ;
// Kimenet : Array ( [a] = > APPLE [b] = > BANANA )
```

2.74. Kódrészlet. array_walk() használata

Az array_walk() akkor lehet különösen hasznos, amikor a tömb elemeit helyben sze retnénk módosítani, vagy amikor szükségünk van a kulcsra is a módosítás során. Fontos megjegyezni, hogy az array_walk() visszatérési értéke mindig true, így a módosított tömböt közvetlenül kell használnunk.

array_filter()

Az array_filter() függvény a tömb elemeit egy megadott függvény alapján szűri. Csak azokat az elemeket tartja meg az eredmény tömbben, amelyekre a szűrő függvény true értéket ad vissza.

```
$numbers = [1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10];
$even = array_filter ( $numbers , function ( $n ) {
        return $n % 2 == 0;
}) ;
print_r ( $even ) ;
    // Kimenet : Array ( [1] = > 2 [3] = > 4 [5] = > 6 [7] = > 8 [9] = > 10 )
```

2.75. Kódrészlet.

array_filter() használata

Az array_filter() rendkívül hasznos, amikor egy tömb bizonyos feltételeknek meg felelő elemeit szeretnénk kiválogatni. Gyakran használják adatok tisztítására vagy speci fikus részhalmazok kinyerésére egy nagyobb adathalmazból.

Fontos megjegyezni, hogy az array_filter() megtartja az eredeti kulcsokat, ezért az eredmény tömb kulcsai nem feltétlenül lesznek folytonosak vagy nullától kezdődőek.

2.6.8. Többdimenziós tömbök

A PHP támogatja a többdimenziós tömböket, amelyek tulajdonképpen "tömbök tömbje". Ezek lehetővé teszik komplex adatstruktúrák létrehozását és kezelését.

Példa:

```
$matrix = [
        [1 , 2 , 3] ,
        [4 , 5 , 6] ,
        [7 , 8 , 9]
];

echo $matrix [1][2]; // Kimenet : 6

$users = [
        [" name " = > " John ", "age" = > 30] ,
        [" name " = > " Jane ", "age" = > 25]
];

echo $users [1][ " name "]; // Kimenet : Jane
```

2.76. Kódrészlet. Többdimenziós tömb

A többdimenziós tömbök különösen hasznosak lehetnek olyan helyzetekben, ahol táb lázatos adatokat vagy hierarchikus struktúrákat kell kezelni. Például egy sakktábla repre zentálása, vagy egy összetett JSON struktúra PHP-beli megfelelője lehet többdimenziós tömb.

A többdimenziós tömbök bejárásához általában egymásba ágyazott ciklusokat hasz nálunk:

```
foreach ( $users as $user ) {
        foreach ( $user as $key = > $value ) {
                echo " $key : $value <br >";
        }
        echo "<br >";
}
```

2.77. Kódrészlet. Többdimenziós tömb bejárása

Bár a többdimenziós tömbök nagyon rugalmasak, túlzott használatuk bonyolulttá teheti a kódot. Komplex adatstruktúrák esetén érdemes megfontolni osztályok vagy ob jektumok használatát a jobb strukturáltság és könnyebb kezelhetőség érdekében.

2.6.9. Javaslatok és megjegyzések tömbökkel kapcsolatban

A tömbök hatékony használata érdekében érdemes megfontolni a következő javaslatokat:

- **Beszédes kulcsnevek**: Használjon érthető, beszédes kulcsneveket asszociatív töm böknél. Ez javítja a kód olvashatóságát és csökkenti a hibák lehetőségét.
- **Memóriahasználat**: Legyen óvatos a nagy tömbök memóriahasználatával, külö nösen amikor másolja vagy módosítja azokat. Nagy adathalmazok esetén fontolja meg a generátorok vagy iterátorok használatát.
- **Foreach ciklus**: A foreach ciklus általában a leghatékonyabb és legolvashatóbb módja a tömbök bejárásának. Használja ezt, hacsak nincs specifikus oka más meg közelítésre.
- **Kulcs ellenőrzése**: Használja az array_key_exists() függvényt az isset() he lyett, ha kifejezetten a kulcs létezését akarja ellenőrizni, nem az értékét. Az isset() false értéket ad vissza, ha a kulcs létezik, de az értéke null.
- **Funkcionális megközelítés**: Az array_map(), array_walk() és array_filter() függvények hatékony eszközök a funkcionális programozási stílus alkalmazásához PHP-ban. Ezek gyakran vezetnek tisztább és könnyebben karbantartható kódhoz.
- **Többdimenziós tömbök**: Többdimenziós tömböknél ügyeljen a mélységre és a komplexitásra. Ha a struktúra túl bonyolulttá válik, fontolja meg objektumok hasz nálatát helyettük.
- **Típusbiztonság**: PHP 7.0-tól kezdve használhatja a típusos tömbdeklarációt (array típushint) a függvények paramétereinek és visszatérési értékeinek megadásakor, ami javítja a kód robusztusságát.
- **Tömb műveletek teljesítménye**: Nagyméretű tömbök esetén a beépített tömb függvények (array_merge, array_diff, stb.) általában hatékonyabbak, mint a kézzel írt ciklusok.

A tömbök a PHP egyik legerősebb és legsokoldalúbb eszközei. Helyes

használatuk kal komplex adatstruktúrákat hozhatunk létre és kezelhetünk hatékonyan. Ugyanakkor fontos, hogy mindig szem előtt tartsuk a kód olvashatóságát és karbantarthatóságát, különösen összetettebb tömb műveletek esetén.

2.7. Dátum- és időkezelés

A dátum- és időkezelés kritikus fontosságú számos alkalmazásban, a felhasználói inter fészektől kezdve az adatbázis-műveletekig és ütemezett feladatokig. A PHP robusztus eszközkészletet biztosít ezekhez a műveletekhez, beleértve mind a procedurális függvé nyeket, mind az objektum-orientált megközelítést.

2.7.1. Aktuális dátum és idő

Az aktuális dátum és idő lekérdezése az egyik legalapvetőbb művelet:

```
echo date ("Y-m-d H:i:s") ; // pl. 2024 -05 -15 14:30:00
echo time () ; // UNIX timestamp , pl. 1715123400
```

2.78. Kódrészlet. Aktuális dátum és idő lekérdezése

A date() függvény egy formázott stringet ad vissza, míg a time() az aktuális UNIX timestamp-et (a másodpercek számát 1970. január 1. óta) adja vissza. A UNIX times tamp széles körben használt, mert könnyen kezelhető és összehasonlítható.

2.7.2. Dátum formázása

A date() függvény lehetővé teszi a dátum tetszőleges formázását:

```
$timestamp = time () ;
echo date ("Y. F j. , l", $timestamp ) ; // pl. 2024. May 15. , Wednesday echo date ("Y-m-d",
$timestamp ) ; // pl. 2024 -05 -15 echo date ("d/m/Y H:i", $timestamp ) ; // pl. 15/05/2024 14:30
echo date ("j. n. Y", $timestamp ) ; // pl. 15. 5. 2024
```

2.79. Kódrészlet. Dátum formázása különböző módon

Néhány további gyakran használt formátum karakter:

- y: Kétjegyű évszám
- F: A hónap neve (pl. January)
- M: A hónap rövid neve (pl. Jan)
- j: A hónap napja vezető nulla nélkül (1-31)
- D: A nap rövid neve (pl. Mon)

• l: A nap teljes neve (pl. Monday)

• g: 12 órás formátumú óra vezető nulla nélkül (1-12)

• a: Kisbetűs ante meridiem és post meridiem (am vagy pm)

2.7.3. Dátum létrehozása és módosítása

Specifikus dátumot létrehozhatunk a strtotime() függvénnyel:

```
$date = strtotime (" 2024 -05 -15 14:30:00 ") ;
echo date ("Y-m-d H:i:s", $date ) ; // 2024 -05 -15 14:30:00

$nextWeek = strtotime ("+1 week ", $date ) ;
echo date ("Y-m-d H:i:s", $nextWeek ) ; // 2024 -05 -22 14:30:00

$lastDayOfMonth = strtotime (" last day of May 2024 ") ;
echo date ("Y-m-d", $lastDayOfMonth ) ; // 2024 -05 -31

$nextTuesday = strtotime (" next Tuesday ") ;
echo date ("Y-m-d", $nextTuesday ) ; // A kö vetkez ő kedd dá tuma
```

2.80.

Kódrészlet. Dátum létrehozása és módosítása

A strtotime() függvény rendkívül rugalmas, és számos természetes nyelvi kifejezést is elfogad. Azonban óvatosan kell használni, mert néha félreérthető lehet. Például a "next Monday" kifejezés a következő hétfőt jelenti, még akkor is, ha ma hétfő van.

2.7.4. DateTime osztály

PHP 5.2-től kezdve rendelkezésre áll a DateTime osztály, amely objektum-orientált meg közelítést kínál a dátum- és időkezeléshez:

```
$date = new DateTime (" 2024 -05 -15 14:30:00 ") ;
echo $date - > format ("Y-m-d H:i:s") ; // 2024 -05 -15 14:30:00

$date - > modify ("+1 month ") ;
echo $date - > format ("Y-m-d H:i:s") ; // 2024 -06 -15 14:30:00

$date - > add (new DateInterval ("P2D") ) ; // Hozz áad 2 napot echo $date - >
format ("Y-m-d H:i:s") ; // 2024 -06 -17 14:30:00

$date - > sub (new DateInterval (" PT3H ") ) ; // Kivon 3 órát
echo $date - > format ("Y-m-d H:i:s") ; // 2024 -06 -17 11:30:00
```

2.81.

Kódrészlet. DateTime osztály használata

A DateTime osztály számos előnnyel rendelkezik:

• Objektum-orientált, így könnyebben kezelhető komplex

alkalmazásokban • Beépített támogatás az időzónákhoz

- Könnyebb dátumok összehasonlítása és különbségek számítása
- Immutable verzió is elérhető (DateTimeImmutable), ami segít elkerülni a mellékha tásokat

2.7.5. DateInterval és DatePeriod osztályok

A DateInterval osztály időtartamok kezelésére szolgál, míg a DatePeriod ismétlődő időintervallumok kezelésére használható:


```
// DateInterval haszn á lata
$interval = new DateInterval (" P1Y2M3DT4H5M6S ") ; // 1 év, 2 hónap , 3 nap , 4 óra , 5 perc ,
    6 má sodperc
$date = new DateTime (" 2024 -01 -01 ") ;
$date - > add ( $interval ) ;
echo $date - > format ("Y-m-d H:i:s") ; // 2025 -03 -04 04:05:06

// DatePeriod haszn á lata
$start = new DateTime (" 2024 -01 -01 ") ;
$interval = new DateInterval ("P1D") ; // 1 nap
$end = new DateTime (" 2024 -01 -10 ") ;
$period = new DatePeriod ( $start , $interval , $end ) ;

foreach ( $period as $date ) {
      echo $date - > format ("Y-m-d") . "\n";
}
// Kiírja 2024 -01 -01 -től 2024 -01 -09 - ig minden nap dátumát
```

2.82. Kódrészlet. DateInterval és DatePeriod használata

2.7.6. Időzónák kezelése

Az időzónák kezelése kritikus fontosságú nemzetközi alkalmazásokban. A PHP DateTimeZone osztályt kínál erre a célra:

```
$date = new DateTime ("now", new DateTimeZone (" Europe / Budapest ") ) ; echo $date -
> format ("Y-m-d H:i:s") ; // Budapesti idő

$date - > setTimezone (new DateTimeZone (" America / New_York ") ) ; echo
$date - > format ("Y-m-d H:i:s") ; // New York -i idő

// Időzóna különbs ég kisz ámítása
$budapestZone = new DateTimeZone (" Europe / Budapest ") ;
$newYorkZone = new DateTimeZone (" America / New_York ") ;
$budapestTime = new DateTime (" now", $budapestZone ) ;
$newYorkTime = new DateTime ("now", $newYorkZone ) ;

$diff = $newYorkZone - > getOffset ( $budapestTime ) - $budapestZone - > getOffset (
    $budapestTime ) ;
echo "Időzóna különbs ég órá kban : " . ( $diff / 3600) ;
```

2.83. Kódrészlet. Időzónák kezelése

Az időzónák kezelésénél fontos figyelembe venni a nyári időszámítást is, ami

automa tikusan kezelve van a DateTime osztályban.

2.7.7. Dátumok összehasonlítása és különbségek számítása

Dátumok összehasonlítására és különbségek számítására több módszer is rendelkezésre áll:

```
$date1 = new DateTime (" 2024 -05 -15 ") ;
$date2 = new DateTime (" 2024 -06 -15 ") ;

if ( $date1 < $date2 ) {
        echo " date1 kor ábbi , mint date2 \n";
```



```
}

$diff = $date1 - > diff ( $date2 ) ;
echo $diff - > format ("%R%a nap \n") ; // +31 nap

echo "Év különbs ég: " . $diff - > y . "\n";
echo "Hónap különbs ég: " . $diff - > m . "\n";
echo "Nap különbs ég: " . $diff - > d . "\n";

// Ö sszes különbs ég napokban
echo "Ö sszes nap különbs ég: " . $diff - > days . "\n";
```

2.84. Kódrészlet. Dátumok összehasonlítása és különbségek számítása

A diff() metódus egy DateInterval objektumot ad vissza, ami részletes információt tartalmaz a két dátum közötti különbségről.

2.7.8. Gyakori problémák és megoldások

Helyi idő vs. szerver idő

Gyakori probléma, hogy a szerver időzónája eltér a kívánt helyi időtől. Ezt a date_default_timezone_sfüggvénnyel kezelhetjük:

```
date_default_timezone_set (" Europe / Budapest ") ;
echo date ("Y-m-d H:i:s") ; // Most már budapesti idő

// Vagy haszn á lhatjuk a DateTime oszt ályt explicit időzónával
$date = new DateTime ("now", new DateTimeZone (" Europe / Budapest ") ) ; echo $date -
> format ("Y-m-d H:i:s") ;
```

2.85. Kódrészlet. Alapértelmezett időzóna beállítása

Adatbázisban tárolt dátumok

Adatbázisokban gyakran YYYY-MM-DD HH:MM:SS formátumban tároljuk a dátumokat. Ezeket könnyen kezelhetjük a DateTime osztállyal:

```
$dbDate = " 2024 -05 -15 14:30:00 ";
$date = new DateTime ( $dbDate ) ;
echo $date - > format ("F j, Y, g:i a") ; // May 15 , 2024 , 2:30 pm

// Dátum visszaalak ítása adatb ázis form á tumba
```

```
echo $date - > format ("Y-m-d H:i:s") ; // 2024 -05 -15 14:30:00
```

2.86. Kódrészlet. Adatbázisban tárolt dátumok kezelése

Relatív dátumok kezelése

A PHP képes kezelni a relatív dátum kifejezéseket, de néha meglepő eredményeket pro dukálhat:

```
$now = new DateTime () ;

$lastDayOfMonth = new DateTime (" last day of this month ") ;
echo $lastDayOfMonth - > format ("Y-m-d") . "\n";
```



```
$nextMonday = new DateTime (" next Monday ") ;
echo $nextMonday - > format ("Y-m-d") . "\n";

// Vigy ázat : ez nem mindig a várt eredm ényt adja !
$firstDayOfNextMonth = new DateTime (" first day of next month ") ; echo
$firstDayOfNextMonth - > format ("Y-m-d") . "\n";
```

2.87. Kódrészlet. Relatív dátumok kezelése

A relatív dátumok használatakor mindig ellenőrizzük, hogy a kapott eredmény megfelel e az elvárásainknak.

Születésnap kiszámítása

```
function calculateAge ( $birthdate ) {
        $birth = new DateTime ( $birthdate ) ;
        $today = new DateTime ('today ') ;
        $age = $birth - > diff ( $today ) ->y ;
        return $age ;
}

echo calculateAge ('1990 -05 -15 ') ; // Kiírja az é letkort
```

2.88. Kódrészlet. Életkor kiszámítása

Munkanapokon alapuló számítások

```
function getNextWorkday ( $date ) {
        $date = new DateTime ( $date ) ;
        do {
                $date - > modify ('+1 day ') ;
        } while ( $date - > format ('N') >= 6) ; // 6 = szombat , 7 = vasá rnap return $date - >
        format ('Y-m-d') ;
}

echo getNextWorkday ('2024 -05 -15 ') ; // Kö vetkez ő munkanap
```

2.89. Kódrészlet. Következő munkanap kiszámítása

2.7.9. Teljesítmény megfontolások

Nagy mennyiségű dátumszámítás esetén a teljesítmény fontos szempont lehet:

- A time() és timestamp-alapú számítások általában gyorsabbak, de kevésbé ponto sak lehetnek (például szökőmásodpercek kezelésénél).
- A DateTime objektumok létrehozása és manipulálása erőforrás-igényesebb lehet, de pontosabb eredményt ad, különösen időzónák és nyári időszámítás kezelésénél.
- Nagy mennyiségű dátum generálásánál vagy számításnál fontolja meg a DatePeriod használatát az egyedi DateTime objektumok létrehozása helyett.

2.8. Gyakorló feladatok

2.8.1. Szintaxis, változók, adattípusok

1. Hozzon létre egy PHP szkriptet, amely deklarál és inicializál egy string, egy integer és egy boolean típusú változót, majd írja ki ezek értékét és típusát.
2. Írjon egy programot, amely demonstrálja a gyenge típusosság koncepcióját egy string és egy szám összeadásával. Magyarázza meg az eredményt.
3. Készítsen egy szkriptet, amely bemutatja a konstansok használatát. Definiáljon egy konstanst a define() függvénnyel és egyet a const kulcsszóval, majd próbálja meg módosítani ezeket. Kommentezze az eredményt.
4. Írjon egy programot, amely demonstrálja a különbséget a isset() és empty() függ vények között különböző típusú és értékű változókon.
5. Hozzon létre egy PHP oldalt, amely egy termék adatait (név, ár, készleten lévő mennyiség) tárolja változókban, majd ezeket megjeleníti egy HTML táblázatban.
6. Írjon egy programot, amely demostrálja a PHP heredoc és nowdoc szintaxisok kö zötti különbséget változók beágyazása szempontjából.
7. Készítsen egy szkriptet, amely bemutatja a változók hatáskörét (scope). Definiáljon egy globális változót, majd módosítsa azt egy függvényen belül a global kulcsszó használatával és anélkül. Magyarázza meg a különbséget.
8. Hozzon létre egy PHP oldalt, amely egy diák adatait (név, életkor, osztály, jegyek) tárolja különböző típusú változókban, majd formázottan megjeleníti ezeket.
9. Készítsen egy egyszerű árfolyam kalkulátort, amely egy fix árfolyamon (konstans) váltja át a felhasználó által megadott euró összeget forintra. Használjon típuskény szerítést a bevitt érték konvertálásához.

10. Készítsen egy egyszerű BMI (testtömegindex) kalkulátort, amely bekéri a felhasz náló magasságát és súlyát, kiszámolja a BMI értéket, majd szöveges értékelést ad az eredményről (pl. alulsúlyos, normál, túlsúlyos). Használjon konstansokat a BMI kategóriák határértékeihez.

2.8.2. Operátorok, vezérlési szerkezetek

1. Írjon egy programot, amely bekér két számot a felhasználótól, és elvégzi rajtuk az alapvető aritmetikai műveleteket (összeadás, kivonás, szorzás, osztás). Használja a megfelelő operátorokat, és kezelje le a nullával való osztás esetét.

2. Készítsen egy szkriptet, amely egy adott számról eldönti, hogy páros vagy páratlan. Használja a maradékos osztás operátort.

3. Írjon egy programot, amely generál egy véletlen számot 1 és 10 között, majd a felhasználónak ki kell találnia ezt a számot. A program adjon visszajelzést, hogy a tipp túl magas vagy túl alacsony.

4. Készítsen egy szkriptet, amely implementálja a "FizzBuzz" játékot: írja ki a számo kat 1-től 20-ig, de ha a szám osztható 3-mal, írja ki helyette, hogy "Fizz", ha 5-tel osztható, írja ki, hogy "Buzz", ha mindkettővel osztható, írja ki, hogy "FizzBuzz".

5. Írjon egy programot, amely bekér egy számot a felhasználótól, és kiírja annak a számnak a szorzótábláját 1-től 10-ig.

6. Készítsen egy egyszerű jelszó-erősség ellenőrzőt. A programnak ellenőriznie kell, hogy a jelszó legalább 8 karakter hosszú, tartalmaz nagybetűt, kisbetűt és számot. Használjon logikai operátorokat és reguláris kifejezéseket.

7. Implementáljon egy programot, amely szimulál egy egyszerű számkitaláló játékot. A program generáljon egy véletlen számot 1 és 100 között, majd a felhasználó próbálja kitalálni. A program adjon visszajelzést, hogy a tipp túl magas vagy túl alacsony, és számolja a próbálkozásokat.

8. Hozzon létre egy programot, amely szimulál egy egyszerű bankszámlát. A program kérjen be egy kezdeti egyenleget, majd egy cikluson belül kérjen be tranzakciókat (betét vagy kivét). Minden tranzakció után írja ki az új egyenleget. A program lépjen ki, ha az egyenleg 0 alá csökkenne.

9. Készítsen egy programot, amely kiszámítja egy romániai kisvállalkozás havi ÁFA (TVA) befizetési kötelezettségét. A program:

   - Kérje be a havi bevételt és a havi kiadásokat (RON-ban).
   - Számolja ki az ÁFA-t a bevételre (19%) és a kiadásokra (19%).
   - Határozza meg a befizetendő vagy visszaigényelhető ÁFA összegét.
   - Ha a cég bevétele meghaladja az 300,000 RON-t, figyelmeztessen a kötelező ÁFA-regisztrációra.

- Jelenítse meg az eredményeket, és adjon rövid szöveges értékelést a cég ÁFA helyzetéről.

Használjon függvényeket a számításokhoz és feltételes szerkezeteket az értékeléshez.

10. Készítsen egy egyszerű raktárkészlet-kezelő rendszert. A program tároljon termé keket és azok mennyiségét egy asszociatív tömbben. Implementáljon egy menü rendszert, ahol a felhasználó választhat termék hozzáadása, eltávolítása, készlet módosítása és leltár kilistázása között. Használjon switch szerkezetet a menüpon tok kezelésére.

2.8.3. Függvények

1. Írjon egy függvényt, amely egy számot kap paraméterként, és visszaadja annak az abszolút értékét. Ne használja a beépített abs() függvényt.

2. Készítsen egy függvényt, amely paraméterként egy szöveget és egy maximális hosszt kap, és visszaadja a szöveg rövidített verzióját, ha az hosszabb a megadott maxi mumnál. A rövidítés során az utolsó három karakter helyett "..." kerüljön a szöveg végére.

3. Implementáljon egy függvényt, amely egy stringet kap paraméterként, és vissza adja, hogy az palindrom-e (ugyanaz visszafelé olvasva, figyelmen kívül hagyva a szóközöket és az írásjeleket).

4. Írjon egy függvényt, amely egy számot kap paraméterként, és visszaadja annak a számnak a bináris reprezentációját string formájában. Ne használja a beépített decbin() függvényt.

5. Készítsen egy függvényt, amely egy szöveget kap paraméterként, és visszaadja a szövegben leggyakrabban előforduló karaktert és annak előfordulási számát.

6. Implementáljon egy függvényt, amely egy asszociatív tömböt kap paraméterként, ahol a kulcsok terméknevek, az értékek pedig árak. A függvény adja vissza a leg drágább termék nevét és árát. Használja a megfelelő beépített függvényeket.

7. Írjon egy függvényt, amely két dátumot kap paraméterként (string formátumban, pl. "2024-05-15"), és visszaadja a két dátum között eltelt munkanapok számát (a hétvégéket nem számolja).

8. Készítsen egy függvényt, amely egy asszociatív tömböt kap paraméterként, ahol a kulcsok terméknevek, az értékek pedig mennyiségek. A függvény rendezze a tömböt a mennyiségek szerint csökkenő sorrendbe, és adja vissza a rendezett tömböt.

9. Implementáljon egy függvényt, amely szimulálja egy egyszerű részvényportfólió ér tékének kiszámítását. A függvény fogadjon el egy asszociatív tömböt, ahol a

kulcsok a részvények nevei, az értékek pedig tömbök, amelyek tartalmazzák a részvények darabszámát és aktuális árfolyamát. A függvény számolja ki és adja vissza a teljes portfólió értékét.

10. Írjon egy függvényt, amely szimulálja egy egyszerű hitelkalkulátor működését. A függvény fogadja paraméterként a hitel összegét, a kamatrátát és a futamidőt hó napokban. A függvény számolja ki és adja vissza a havi törlesztőrészletet és a teljes visszafizetendő összeget.

2.8.4. Tömbök

1. Hozzon létre egy indexelt tömböt, amely tartalmazza a hét napjait. Írjon egy szkriptet, amely kiírja a tömb elemeit fordított sorrendben.

2. Készítsen egy asszociatív tömböt, amely országokat és azok fővárosait tartalmazza. Írjon egy függvényt, amely bekér egy országnevet a felhasználótól, és visszaadja annak fővárosát. Ha az ország nem található, adjon megfelelő hibaüzenetet.

3. Implementáljon egy függvényt, amely két tömböt kap paraméterként, és visszaadja a két tömb metszetét (azokat az elemeket, amelyek mindkét tömbben megtalálha tók). Használja a megfelelő beépített függvényeket.

4. Írjon egy függvényt, amely egy tömböt kap paraméterként, és visszaadja a tömb második legnagyobb elemét. Ne használja a beépített rendező függvényeket.

5. Készítsen egy programot, amely szimulál egy egyszerű TODO listát. Használjon tömböt a feladatok tárolására. A program tudjon új feladatot hozzáadni, feladatot törölni, feladatot késznek jelölni, és listázni a feladatokat állapot szerint (kész/fo lyamatban).

6. Írjon egy függvényt, amely egy számokból álló tömböt kap paraméterként, és vissza adja a tömb "csúcspontjait". Egy elem csúcspont, ha nagyobb mindkét szomszéd jánál.

7. Implementáljon egy függvényt, amely egy asszociatív tömböt kap paraméterként, ahol a kulcsok nevek, az értékek pedig életkorok. A függvény csoportosítsa az embereket korosztályok szerint (pl. 0-18, 19-30, 31-50, 50+), és adja vissza az eredményt egy új asszociatív tömbben.

8. Írjon egy programot, amely egy többdimenziós tömböt használ egy egyszerű mozi ülésrendjének reprezentálására. A program tudjon helyet foglalni, foglalást törölni, és kilistázni a szabad és foglalt helyeket.

9. Készítsen egy egyszerű raktárkészlet-kezelő rendszert tömbök segítségével. A rend szer tárolja a termékek nevét, árát és mennyiségét. Implementáljon funkciókat új termék hozzáadására, készlet módosítására, és készítsen egy riportot, amely kilis tázza a termékeket az összértékükkel együtt (ár * mennyiség).

10. Implementáljon egy egyszerű kosárlabda bajnokság pontozó rendszert. Használjon többdimenziós tömböt a csapatok és mérkőzések adatainak tárolására. A prog ram tudjon új mérkőzés eredményt rögzíteni, és generáljon egy rangsort a csapatok pontszáma alapján.

2.8.5. Dátum- és időkezelés

1. Írjon egy programot, amely kiírja az aktuális dátumot és időt különböző formátu mokban (pl. "2024-05-15", "15/05/2024", "May 15, 2024", "14:30:00").

2. Készítsen egy függvényt, amely kiszámolja és visszaadja egy személy életkorát évek ben. A függvény paraméterként kapja meg a születési dátumot.

3. Implementáljon egy programot, amely kiszámítja és kiírja az aktuális hét kezdő és záró dátumát (hétfő és vasárnap).

4. Írjon egy függvényt, amely kiszámolja és visszaadja két dátum között eltelt napok számát. A dátumokat string formátumban kapja a függvény.

5. Készítsen egy programot, amely kiírja az aktuális hónap naptárát. Emelje ki az aktuális napot.

6. Implementáljon egy függvényt, amely egy születési dátumot kap paraméterként, és visszaadja a személy korát években, valamint hogy hány nap van hátra a következő születésnapjáig.

7. Írjon egy függvényt, amely kiszámolja és visszaadja, hogy egy adott évben hány péntek 13-a van.

8. Készítsen egy függvényt, amely kiszámítja két időpont között eltelt időt órákban, percekben és másodpercben.

9. Készítsen egy függvényt, amely kiszámítja két időpont között eltelt időt órákban, percekben és másodpercekben. Az időpontokat string formátumban kapja a függ vény (pl. "14:30:00").

10. Implementáljon egy egyszerű szabadság-nyilvántartó rendszert. A program tárolja a dolgozók nevét és szabadságnapjaik számát. Tudjon új szabadságot rögzíteni (kezdő és záró dátummal), kiszámolni a felhasznált szabadságnapok számát (csak munkanapokat számolva), és kimutatást készíteni a fennmaradó szabadságokról. Vegye figyelembe az ünnepnapokat is (használjon egy előre definiált tömböt az ünnepnapok tárolására).

3. fejezet

Objektumorientált programozás PHP-ben

3.1. Bevezetés

Az objektumorientált programozás (OOP) egy olyan programozási paradigma, amely

a valós világ objektumainak modellezésére épül. A PHP nyelvben az OOP támogatása jelentős fejlődésen ment keresztül az évek során, és ma már a nyelv egyik központi elemévé vált.
Az OOP jellemzői:

- **Kód újrafelhasználhatóság:** Az osztályok és objektumok lehetővé teszik a kód moduláris felépítését, ami megkönnyíti a kód újrafelhasználását különböző projek tekben.
- **Karbantarthatóság:** A jól strukturált OOP kód könnyebben érthető és módosít ható, ami különösen fontos nagyobb projekteknél.
- **Skálázhatóság:** Az OOP elvek segítenek a nagyobb projektek kezelésében, lehe tővé téve a komplex rendszerek logikus felépítését.
- **Biztonság:** Az egységbezárás (encapsulation) segít az adatok védelmében, korlá tozva a közvetlen hozzáférést az osztály belső állapotához.
- **Együttműködés:** Az OOP szabványos struktúrát biztosít, ami megkönnyíti a csapatmunkát és a kód megosztását.

A PHP OOP támogatásának története:

- **PHP 3 (1998):** Bevezetésre került az osztályok és objektumok alapvető támo gatása. Ez még nagyon korlátozott volt, de lehetővé tette az OOP alapelveinek alkalmazását.
- **PHP 4 (2000):** Továbbfejlesztett OOP funkciók, beleértve a konstruktorokat és a statikus tagokat. Azonban még mindig hiányoztak olyan kulcsfontosságú elemek, mint a láthatósági módosítók.
- **PHP 5 (2004):** Jelentős újítások az OOP terén:

  - Láthatósági módosítók (public, protected, private)
  - Absztrakt osztályok és interfészek
  - Objektum klónozás (__clone() metódus)
  - Típusos metódus-paraméterek
  - Kivételkezelés (try-catch blokkok)
- **PHP 7 (2015):** További OOP fejlesztések:
  - Return type declarations
  - Anonymous classes
  - Null coalescing operator (??)
- **PHP 8 (2020):** Újabb OOP funkciók:
  - Named arguments
  - Attributes

– Constructor property promotion
– Match expressions

3.2. Osztályok és objektumok

3.2.1. Osztály definiálás, objektumok létrehozása

Az osztályok a PHP-ben az objektumorientált programozás alapvető építőelemei. Egy osztály tulajdonságok (változók) és metódusok (függvények) gyűjteménye, amely egy adott entitást vagy koncepciót modellez.

Szintaxis:

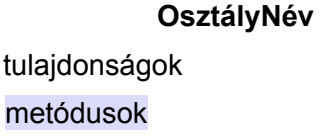
```
class OsztályNév {
    tulajdonságok
    metódusok
}
```

Ahol:

- OsztályNév: Az osztály neve, nagybetűvel kezdve (konvenció szerint).
- tulajdonságok: Az osztály adattagjai (változói), amelyek az objektum állapotát tárolják.
- metódusok: Az osztályhoz tartozó függvények, amelyek az objektum viselkedését definiálják.

A tulajdonságok és metódusok előtt láthatósági módosítókat (public, protected, private) használhatunk, hogy szabályozzuk a hozzáférést. A fenti szintaxisnak megfelelő UML (Unified Modelling Language) diagram a 3.1.Ábrán látható.

Az alábbi példában egy Student nevű osztályt láthatunk két adattaggal és egy me tódussal:



3.1. ábra. UML osztálydiagram

```
class Student {
    public $name ;
    public $age ;

    public function introduce () {
        return "Hello , my name is " . $this - > name . " and I'm " . $this - > age . " years old.";
    }
}
```

3.1. Kódrészlet. Egyszerű Student osztály

Objektumok létrehozására (példányosítás) a new operátort használjuk, az adattagok elérésére pedig a -> operátort:

Példa:

```
$student = new Student () ;
$student - > name = " John Doe ";
$student - > age = 20;

echo $student - > introduce () ;
// Kimenet : Hello , my name is John Doe and I'm 20 years old.
```

3.2. Kódrészlet.

Student osztály példányosítása

3.2.2. Tulajdonságok, metódusok, láthatóság

Az osztályok tulajdonságokkal (változók) és metódusokkal (függvények) rendelkeznek. A láthatósági módosítók szabályozzák ezek hozzáférhetőségét. Láthatósági módosítók:

- public: Mindenhonnan hozzáférhető
- protected: Az osztályon belülről és a leszármazott osztályokból érhető el
- private: Csak az osztályon belülről érhető el

Példa a Student osztály különböző láthatóságú tagokkal:

```
class Student {
    private string $id ;
    public string $name ;
    protected int $age ;

    public function __construct ( string $id , string $name , int $age ) { $this - > id = $id ;
```



```
        $this - > name = $name ;
        $this - > age = $age ;
    }

    public function introduce () : string {
        return "Hello , my name is " . $this - > name . " and I'm " . $this - > age . " years old.";
    }

    protected function getAge () : int {
        return $this - > age ;
    }

    private function getId () : string {
        return $this - > id ;
    }

    public function getInfo () : string {
        return " Name : " . $this - > name . ", Age: " . $this - > getAge () . " , ID: " . $this - > getId
            () ;
```

```
    }
}
```

3.3. Kódrészlet. Student osztály különböző láthatóságú tagokkal

A fenti osztály használata és a láthatósági szabályok demonstrálása:

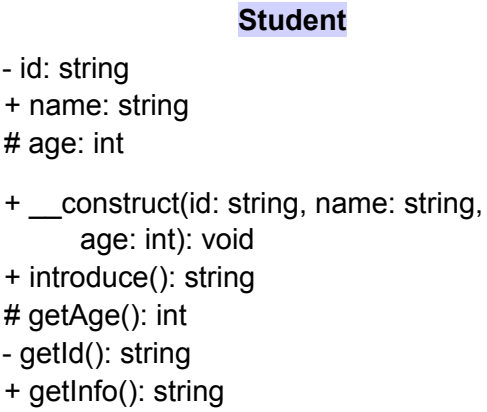
```
$student = new Student (" John Doe", 20 , " S12345 ") ;

echo $student - > name ; // Működik , mert public
echo $student - > introduce () ; // Működik , mert public metódus echo $student - >
getInfo () ; // Működik , mert public metódus

// echo $student - >age ; // Hiba : Cannot access protected property // echo $student - > getAge
(); // Hiba : Cannot access protected method // echo $student - >id; // Hiba : Cannot access
private property // echo $student - > getId (); // Hiba : Cannot access private method
```

3.4. Kódrészlet. Student osztály használata és láthatósági

szabályok Ebben a példában:

- A name tulajdonság és az introduce() metódus nyilvános, így kívülről is elérhetőek.
- Az age tulajdonság és a getAge() metódus védett, így csak az osztályon belül és a leszármazott osztályokban használhatók.
- A id tulajdonság és a getId() metódus privát, így csak az osztályon belül érhetők el.
- A getInfo() metódus nyilvános, de belül használja a védett és privát metódusokat, demonstrálva, hogyan lehet biztonságosan hozzáférni a védett és privát tagokhoz.

Ez a struktúra biztosítja, hogy az osztály belső működése el legyen rejtve a külvilág elől, miközben szabályozott hozzáférést biztosít a szükséges funkciókhoz és adatokhoz. Az osztály UML diagramaja a 3.2.Ábrán látható.



3.2. ábra. A Student osztály UML diagramja

3.2.3. Konstruktor, destruktor

A konstruktor és destruktor speciális metódusok az osztályokban, amelyek az objektum életciklusának kulcsfontosságú pontjain hívódnak meg.

Konstruktor

A konstruktor (__construct()) egy speciális metódus, amely az objektum létrehozásakor automatikusan meghívódik. Fő feladata az objektum inicializálása, kezdeti állapotának beállítása.

Példa konstruktor használatára a Student osztályban:

```
class Student
{
	private string $id ;
	private string $name ;
	private int $age ;

	public function __construct ( string $id , string $name , int $age ) {
		$this - > id = $id ;
		$this - > name = $name ;
		$this - > age = $age ;
		echo "New Student object created : {$this - > name } <br >";
	}

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

	public function getAge () : int
	{
		return $this - > age ;
	}

	public function getId () : string
	{
		return $this - > id ;
```



```
	}

	public function introduce () : string
	{
		return "Hi , I'm {$this - > name } , {$this - >age} years old , student ID: {$this - >id }.";
	}
}

$student = new Student (" S12345 ", " Alice Johnson ", 20) ;
echo $student - > introduce () ;
// Kimenet :
// New Student object created : Alice Johnson
// Hi , I'm Alice Johnson , 20 years old , student ID: S12345 .
```

3.5. Kódrészlet.

Student osztály konstruktorral és típusokkal

Ebben a példában:

• A konstruktor három paramétert fogad: $id, $name, és$age.

• Mindegyik paraméternek és tulajdonságnak megadtuk a típusát (string,

int). • A getter metódusok visszatérési típusát is deklaráltuk.

Destruktor

A destruktor (__destruct()) az objektum megsemmisítésekor hívódik meg, általában amikor az objektumra már nincs több hivatkozás vagy a script véget ér. Használható erőforrások felszabadítására vagy befejező műveletek végrehajtására.

Példa destruktor használatára:

```
class Student
{
        private string $id ;
        private string $name ;
        private int $age ;

        public function __construct ( string $id , string $name , int $age ,) {
                $this - > id = $id ;
                $this - > name = $name ;
                $this - > age = $age ;
                echo " Student {$this - > name } created . <br >";
        }

        public function __destruct ()
        {
                echo " Student {$this - > name } object is being destroyed . <br >"; }

        // ... (többi met ódus ugyanaz , mint az előző példában)
}

function createAndDestroyStudent () : void
{
        $student = new Student (" S67890 ", " Bob Smith ", 22) ;
```



```
        echo $student - > introduce () . "<br >";
        // A fü ggv ény végén a $student objektum megsemmis ül
}

createAndDestroyStudent () ;
echo "End of script . <br >";

// Kimenet :
// Student Bob Smith created .
// Hi , I'm Bob Smith , 22 years old , student ID: S67890 . // Student Bob Smith
object is being destroyed .
// End of script .
```

3.6. Kódrészlet. Student osztály destruktorral

Ebben a példában:

• A destruktor kiír egy üzenetet, amikor az objektum megsemmisül.

• A createAndDestroyStudent() függvény demonstrálja, hogy a destruktor auto matikusan meghívódik, amikor az objektum kikerül a hatókörből.

• Figyeljük meg, hogy a destruktor a script vége előtt hívódik meg, amikor a $student objektum megsemmisül.

A konstruktor és destruktor használata segít az objektumok megfelelő inicializálásá ban és erőforrásaik felszabadításában, ami különösen fontos lehet nagyobb alkalmazások nál vagy erőforrás-igényes objektumoknál.

3.2.4. Statikus adattagok és metódusok

A statikus tagok (tulajdonságok és metódusok) az osztályhoz tartoznak, nem pedig az osz tály egy konkrét példányához. Ezeket a static kulcsszóval deklaráljuk. Statikus tagokat az osztály nevével és a hatókör (::) operátorral érhetünk el, objektum példányosítása nélkül.

Példa statikus tagok használatára a Student osztályban:

```
class Student
{
        private string $name ;
        private int $age ;
        private string $id ;
        private static int $studentCount = 0;
        private static string $schoolName = "PHP High School ";

        public function __construct ( string $name , int $age )
        {
                $this - > name = $name ;
                $this - > age = $age ;
                self :: $studentCount ++;
                $this - > id = "S" . str_pad ( self :: $studentCount , 5 , "0",
                        STR_PAD_LEFT ) ;
        }

        public static function getStudentCount () : int
```



```
        {
                return self :: $studentCount ;
        }

        public static function getSchoolName () : string
        {
                return self :: $schoolName ;
        }

        public static function setSchoolName ( string $name ) : void {
                self :: $schoolName = $name ;
        }

        public function getInfo () : string
        {
                return " Name : {$this - > name } , Age : {$this - >age} , ID: {$this - >id} ";
        }
```

```
}

// Statikus tagok haszn á lata
echo " School Name : " . Student :: getSchoolName () . "<br >"; echo " Initial Student Count : " .
Student :: getStudentCount () . "<br >";

$student1 = new Student (" Alice ", 20) ;
$student2 = new Student ("Bob", 22) ;

echo " Updated Student Count : " . Student :: getStudentCount () . "<br >"; echo $student1 - >
getInfo () . "<br >";
echo $student2 - > getInfo () . "<br >";

Student :: setSchoolName ("PHP Advanced Academy ") ;
echo "New School Name : " . Student :: getSchoolName () . "<br >";

// Kimenet :
// School Name : PHP High School
// Initial Student Count : 0
// Updated Student Count : 2
// Name : Alice , Age : 20 , ID: S00001
// Name : Bob , Age : 22 , ID: S00002
// New School Name : PHP Advanced Academy
```

3.7. Kódrészlet. Student osztály statikus tagokkal

Ebben a példában:

- $studentCount egy statikus tulajdonság, amely nyomon követi a létrehozott diákok számát.
- $schoolName egy statikus tulajdonság, amely az iskola nevét tárolja.
- A konstruktor növeli a $studentCount értékét és generál egy egyedi azonosítót minden új diák számára.
- getStudentCount() és getSchoolName() statikus metódusok, amelyek visszaadják a statikus tulajdonságok értékeit.

- setSchoolName() egy statikus metódus, amely lehetővé teszi az iskola nevének módosítását.
- A self:: kulcsszót használjuk a statikus tagok osztályon belüli eléréséhez.

Statikus tagok használatának előnyei:

- Nem igényelnek objektum példányosítást, így gyorsabban elérhetők és kevesebb memóriát használnak.
- Alkalmasak globális állapotok vagy konstansok tárolására egy osztályon belül.
- Hasznos segédfüggvények (utility functions) implementálására, amelyek nem függ nek az objektum állapotától.

Fontos megjegyezni, hogy a statikus tagok nem férnek hozzá a nem statikus tagokhoz közvetlenül, mivel nincs $this objektum kontextus a statikus

metódusokban.

3.2.5. Az stdClass osztály

A stdClass egy beépített üres osztály a PHP-ben, amit gyakran használnak dinamikus objektumok létrehozására. Ez az osztály nem tartalmaz előre definiált metódusokat vagy tulajdonságokat, de lehetővé teszi, hogy dinamikusan adjunk hozzá tulajdonságokat futási időben.

Példák a stdClass használatára:

```
// 1. Üres objektum lé trehoz ása és tulajdons ágok hozz áadása $student = new
stdClass () ;
$student - > name = " John Doe ";
$student - > age = 20;
$student - > id = " S00003 ";

echo " Student : " . $student - > name . ", Age: " . $student - > age . ", ID: " . $student - > id ;
// Kimenet : Student : John Doe , Age: 20 , ID: S00003

// 2. Tömbb ől objektum lé trehoz ása
$studentArray = [
        " name " = > " Jane Smith ",
        "age" = > 22 ,
        "id" = > " S00004 "
];

$studentObject = ( object ) $studentArray ;

echo " Student : " . $studentObject - > name . ", Age: " . $studentObject - > age . ", ID: " .
      $studentObject - > id ;
// Kimenet : Student : Jane Smith , Age: 22 , ID: S00004
```

3.8. Kódrészlet. stdClass használata és dinamikus

tulajdonságok A stdClass használatának előnyei:

- Rugalmas: Könnyen hozzáadhatunk vagy eltávolíthatunk tulajdonságokat futási időben.

- Könnyű használni: Nem igényel előzetes osztálydefiníciót.
- Hasznos adatátvitelre: Gyakran használják API-k válaszainak vagy konfigurációs adatok tárolására.
- Gyors prototípuskészítés: Ideális gyors, dinamikus objektumok létrehozására fej lesztés közben.

Hátrányai:

- Nincs típusellenőrzés: Könnyen hibákhoz vezethet, ha nem vagyunk óvatosak.
- Nehezebb karbantartani: Nagyobb projekteknél az explicit osztálydefiníciók általá ban jobb választások.

• Nincs metódus támogatás: Csak tulajdonságokat tárolhatunk, metódusokat

nem. **3.3. Öröklődés**

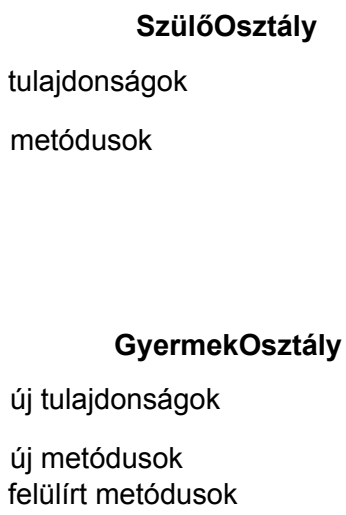
3.3.1. Mi az öröklődés?

Az öröklődés az objektumorientált programozás egyik alapvető koncepciója, amely lehe tővé teszi, hogy egy új osztályt egy meglévő osztály alapján hozzunk létre. Az új osztály (leszármazott vagy gyermek osztály) örökli a meglévő osztály (szülő vagy alap osztály) tulajdonságait és metódusait.

Szintaxis:

```
class GyermekOsztály extends SzülőOsztály
{
    // Új tulajdonságok és metódusok
    // Meglévő metódusok felülírása
}
```

Két osztály közötti öröklődést a 3.3.Ábra szemlélteti. A kapcsolatot a gyerekosztálytól az ősosztály felé mutató *folytonos* vonallal jelöljük.



3.3. ábra. Öröklődés UML diagramja

Példa öröklődésre: egy Person osztályból származtatjuk a Student osztályt

```
class Person
{
    protected string $name ;
    protected int $age ;

    public function __construct ( string $name , int $age )
    {
        $this - > name = $name ;
        $this - > age = $age ;
    }
```

```
        public function introduce () : string
        {
                return "Hi , I'm {$this - > name } and I'm {$this - >age} years old."; }
}

class Student extends Person
{
        private string $id ;

        public function __construct ( string $id , string $name , int $age ) {
                parent :: __construct ( $name , $age ) ;
                $this - > id = $id ;
        }

        public function introduce () : string
        {
                return parent :: introduce () . " My student ID is {$this - > studentId }.";
        }

        public function study () : string
        {
                return "{$this - > name } is studying .";
        }
}

$person = new Person (" John ", 30) ;
echo $person - > introduce () ;
// Kimenet : Hi , I'm John and I'm 30 years old.

$student = new Student (" S12345 ", " Alice ", 20) ;
echo $student - > introduce () ;
// Kimenet : Hi , I'm Alice and I'm 20 years old. My student ID is S12345 .
echo $student - > study () ;
// Kimenet : Alice is studying .
```

3.9. Kódrészlet. Öröklődés példa: Person és Student osztályok

A fenti példában:

• A Student osztály örökli a Person osztály tulajdonságait és metódusait. • A Student osztály kibővíti a Person osztályt egy új $studentId tulajdonsággal.

• Az introduce() metódus felül van írva a Student osztályban, de felhasználja a szülő osztály implementációját is.

• A study() egy új metódus, amely csak a Student osztályban létezik.

3.3.2. A parent:: használata

A parent:: kulcsszó lehetővé teszi, hogy a gyermek osztályból elérjük a szülő osztály metódusait és tulajdonságait. Ez különösen hasznos, amikor felül szeretnénk írni egy metódust, de meg szeretnénk tartani az eredeti funkcionalitás egy részét is. Lássuk, hogyan használtuk a parent:: kulcsszót a Student példában:

1. **Konstruktor hívásában:**

```
public function __construct ( string $id , string $name , int $age ) {
        parent :: __construct ( $name , $age ) ;
        $this - > id = $id ;
}
```

Itt a parent::__construct(*name,*age) a Person osztály konstruktorát hívja, ini cializálva az öröklött $name és $age tulajdonságokat. Ezt a gyermek osztály konst ruktorának elején helyeztük el, mielőtt az új $id tulajdonságot inicializáltuk volna.

2. **Metódus felülírásakor:**

```
public function introduce () : string
{
        return parent :: introduce () . " My student ID is {$this - >id }.";
}
```

Itt a parent::introduce() meghívja a Person osztály introduce() metódusát, majd az eredményhez hozzáfűzzük a hallgatói azonosítót. Ez lehetővé teszi, hogy kibővítsük a szülő osztály funkcionalitását anélkül, hogy teljesen újra kellene írnunk azt.

A parent:: használatának előnyei:

• Lehetővé teszi a szülő osztály funkcionalitásának

újrafelhasználását. • Csökkenti a kód duplikációt.

• Segít fenntartani a konzisztenciát az osztályhierarchiában.

• Lehetővé teszi a fokozatos módosítást és kiterjesztést.

Fontos megjegyzések:

• A parent:: csak olyan metódusokra és tulajdonságokra hivatkozhat, amelyek public vagy protected láthatósággal rendelkeznek a szülő osztályban. A private tagok nem érhetők el a gyermek osztályból, még a parent:: kulcsszóval sem.

• A study() metódus nem használja a parent:: kulcsszót, mert ez egy teljesen új metódus a Student osztályban, amely nem létezik a Person osztályban.

• A protected láthatóságú $name és $age tulajdonságok elérhetők a Student osz tályban, ezért tudjuk használni őket a study() metódusban.

3.3.3. Interfész

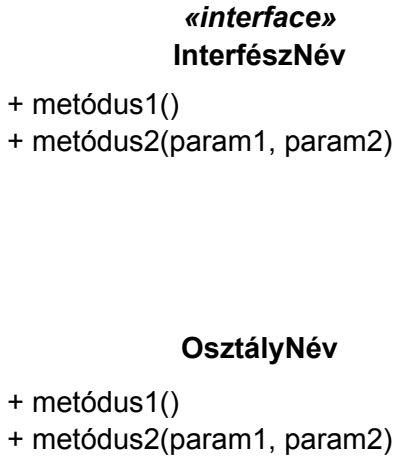
Az interfész egyfajta "szerződés", amely meghatározza, hogy egy osztálynak milyen nyil vános metódusokat kell implementálnia. Az interfészek lehetővé teszik, hogy közös vi selkedést definiáljunk különböző osztályok között anélkül, hogy azok között öröklődési kapcsolat lenne.

Szintaxis:

```
interface InterfészNév
{
    public function metódus1();
```

```
    public function metódus2($param1, $param2);
}

class OsztályNév implements InterfészNév
{
    public function metódus1()
    {
        // Implementáció
    }
    public function metódus2($param1, $param2)
    {
        // Implementáció
    }
}
```

Az interfész definiálásához az interface kulcsszót használjuk. Az osztályok az inter fész megvalósítást az implements kulcsszóval jelzik. A fenti szintaxisnak megfelelő UML diageam a 3.4.Ábrán látható. A kapcsolatot a megvalósító osztálytól az interfész felé mutató *szaggatott* nyillal jelöljük.



3.4. ábra. Interfész és azt megvalósító osztály UML diagramja

Szabályok