**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**IP COLEGIUL „IULIA HASDEU” DIN CAHUL**

**CATEDRA TIC**

**Asistență pentru programarea server-side**

**PHP**

****

**CAHUL 2023**

**IP Colegiul „Iulia Hasdeu” Din Cahul**

**Autor**

Pădure Gheorghe IP Colegiul „Iulia Hasdeu” din Cahul

**Aprobat de**

Consiliul metodico-ştiinţific al IP CIHC

**Recenzenți:**

Pîrvan Evghenii, profesor informatică, grad didactic superior, IP Colegiul „Iulia Hasdeu” din Cahul

Pădure Maria profesoară informatică, grad didactic II, IP Colegiul „Iulia Hasdeu” din Cahul

**Cuprins**

[**Modulul 1: Limbaje de programare pentru server-side** 5](#_Toc147094268)

[**1.1 Ce este PHP?** 6](#_Toc147094269)

[**1.2 Istoria PHP** 6](#_Toc147094270)

[**1.3 Instalarea PHP** 7](#_Toc147094271)

[**1.4 Primii pași în PHP** 7](#_Toc147094272)

[**1.5 Tipuri de date** 7](#_Toc147094273)

[**1.6 Variabile în PHP** 10](#_Toc147094274)

[**1.7 Constante în PHP** 11](#_Toc147094275)

[**1.8 Operatori** 12](#_Toc147094276)

[**1.9 Expresii** 14](#_Toc147094277)

[**1.10 Comentarii** 16](#_Toc147094278)

[**1.11 Array-uri** 17](#_Toc147094279)

[**Test de evaluare** 21](#_Toc147094280)

[**Probleme practice** 24](#_Toc147094281)

[**Modulul 2: Instrucțiuni** 25](#_Toc147094282)

[**2.1 Instrucțiuni condiționale:** 25](#_Toc147094283)

[**2.2 Instrucțiuni repetitive:** 26](#_Toc147094284)

[**2.3 Instrucțiuni de salt** 26](#_Toc147094285)

[**1.4 Instrucțiuni de includere și evaluare a fișierelor:** 27](#_Toc147094286)

[**Test de evaluare** 28](#_Toc147094287)

[**Probleme practice** 29](#_Toc147094288)

[**Modulul 3: Funcții** 31](#_Toc147094289)

[**3.1 Ce este o funcție în PHP?** 32](#_Toc147094290)

[**3.2 Definirea unei funcții în PHP** 32](#_Toc147094291)

[**3.3 Parametrii funcției** 33](#_Toc147094292)

[**3.4 Tipurile de returnare** 33](#_Toc147094293)

[**3.5 Funcții recursive** 34](#_Toc147094294)

[**3.5 Funcții anonime** 35](#_Toc147094295)

[**3.6 Funcțiile arrow** 36](#_Toc147094296)

[**3.7 Definirea tipurilor de date pentru parametrii funcțiilor și valorile returnate** 38](#_Toc147094297)

[**Test de evaluare** 39](#_Toc147094298)

[**Probleme practice** 41](#_Toc147094299)

[**Modulul 4: Lucrul cu directoare și fișiere** 43](#_Toc147094300)

[**1. Funcții de lucru cu directoare și fișiere** 43](#_Toc147094301)

[**2. Funcții de lucru cu fișiere** 44](#_Toc147094302)

[**3. Încărcarea fișierelor** 44](#_Toc147094303)

[**Test de evaluare** 45](#_Toc147094304)

[**Probleme practice** 46](#_Toc147094305)

[**Modulul 5: Formulare** 48](#_Toc147094306)

[**5.1. Crearea un Formular HTML** 48](#_Toc147094307)

[**5.2. Transmiterea Datelor folosind Metoda GET** 50](#_Toc147094308)

[**5.3 Transmiterea Datelor folosind Metoda POST** 50](#_Toc147094309)

[**5.4 Validarea datelor.** 51](#_Toc147094310)

# **Modulul 1: Limbaje de programare pentru server-side**

**Unităţi de conţinut**

1. Limbaje de programare pentru server. Caracteristicile de bază. Dezvoltarea limbajului.
2. Server Web local.
3. Editoare de cod.
4. Sintaxa limbajului.
5. Tipuri de date.
6. Variabile și constate.
7. Operatori.
8. Expresii.
9. Comentarii.

**Abilități**

A1. Selectarea limbajului de scriptare.

A2. Instalarea aplicațiilor necesare pentru serverul local.

A3. Configurarea serverului local.

A4. Scrierea scripturilor.

A5. Executarea scripturilor.

A6. Comentarea scripturilor.

A7. Depanarea scripturilor.

A8. Definirea variabilelor și constantelor în script.

A9. Scrierea expresiilor utilizând variabile, constante și operatori.

**Cuvinte cheie:**

* **Variabile**: Elemente folosite pentru stocarea și manipularea datelor în PHP.
* **Operatori**: Simboluri și funcții utilizate pentru a efectua operații pe date.
* **Expresii**: Combinații de variabile, constante și operatori care produc un rezultat.
* **Constante**: Identificatori care reprezintă valori neschimbătoare în PHP.
* **Comentarii**: Text adăugat în cod pentru a explica sau documenta funcționalitatea acestuia.
* **Atribuire**: Procesul de asociere a unei valori unei variabile folosind operatorul **=**.
* **Egalitate strictă**: Verificarea dacă două valori sunt egale și au același tip de date.
* **Operatori logici**: Simboluri utilizate pentru a efectua operații logice, cum ar fi **&&**, **||** și **!**.
* **Concatenare**: Operația de lipire a două sau mai multe șiruri de caractere.
* **Comentarii de documentare**: Comentarii speciale utilizate pentru a documenta funcțiile, metodele și clasele într-un format specific pentru generarea automată a documentației.

## **1.1 Ce este PHP?**

**PHP** (Hypertext Preprocessor) este un limbaj de programare open-source și interpretat, specializat în dezvoltarea web. A fost creat inițial de Rasmus Lerdorf în anii 1990 și este extrem de popular în dezvoltarea site-urilor web din cauza simplității sale, a versatilității și a capacității de a se integra ușor cu HTML.

PHP este folosit pentru a crea pagini web dinamice, adică pagini care pot genera conținut diferit în funcție de datele de intrare sau alte condiții. Acesta funcționează pe server și generează HTML, care este apoi trimis către browserul utilizatorului. Principalele utilizări ale PHP includ:

* Generarea dinamică a conținutului web: PHP permite adăugarea de date dinamice în pagini web, cum ar fi informații din baze de date sau rezultatele unor calcule.
* Gestionarea formularelor web: PHP poate procesa datele trimise de utilizatori prin formulare web, efectuând validări și acțiuni în funcție de acestea.
* Interacțiunea cu baze de date: PHP poate comunica cu baze de date pentru a stoca și recupera date. MySQL este o bază de date frecvent utilizată în combinație cu PHP.
* Dezvoltarea aplicațiilor web complete: PHP poate fi folosit pentru a construi aplicații web complexe, cum ar fi sistemele de gestionare a conținutului (CMS) sau platformele de comerț electronic.

## **1.2 Istoria PHP**

PHP a apărut inițial în 1994 ca un set simplu de scripturi pentru gestionarea paginilor web personale ale lui Rasmus Lerdorf. Cu timpul, a evoluat într-un limbaj de programare robust și a fost lansat public pentru prima dată în 1995 sub numele de "Personal Home Page Tools" (PHP Tools).

Principalele etape din istoria PHP includ:

* PHP/FI (Forms Interpreter): Prima versiune a limbajului, dezvoltată de Lerdorf, a fost axată pe interpretarea formularilor web.
* PHP 2.0: Prima versiune oficială, lansată în 1997, a introdus funcționalități mai avansate și a devenit populară în comunitatea dezvoltatorilor web.
* PHP 3.0: A adus suport pentru dezvoltarea orientată pe obiect (OOP) și a îmbunătățit gestionarea sesiunilor.
* PHP 4.0: A introdus o arhitectură complet nouă și suport pentru multiple tipuri de obiecte.
* PHP 5.0: A adus o serie de îmbunătățiri majore, inclusiv suport pentru OOP îmbunătățit, gestionarea erorilor îmbunătățită și extensii pentru lucrul cu baze de date.
* PHP 7.0: O versiune importantă care a adus performanțe semnificativ îmbunătățite și alte caracteristici noi.
* PHP 8.0: Versiunea cea mai recentă la momentul tăierii cunoștințelor mele, PHP 8.0, a adus mai multe caracteristici noi, inclusiv tipuri de argumente și întoarceri, atribuirea ternară, și multe alte îmbunătățiri semnificative.

## **1.3 Instalarea PHP**

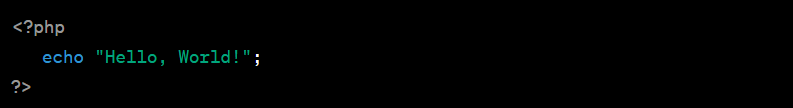
Pentru a începe să lucrăm cu PHP, trebuie să aveți un mediu de dezvoltare configurat. Există mai multe moduri de a instala PHP, inclusiv:

* **XAMPP/WAMP/MAMP**: Acestea sunt pachete care conțin PHP, server web (de obicei Apache), MySQL și alte componente necesare pentru dezvoltarea web. Sunt ușor de instalat și pot fi folosite pe platforme precum Windows, macOS și Linux.
* **Instalare manuală**: Puteți descărca și instala PHP manual pe serverul dvs. web sau pe computerul local, configurând serverul web și PHP separat.

După instalare, puteți verifica dacă PHP este funcțional prin crearea unui fișier PHP simplu și vizualizând rezultatul într-un browser web.

## **1.4 Primii pași în PHP**

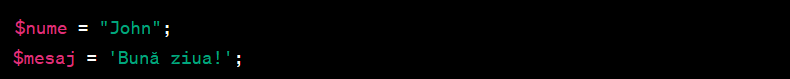
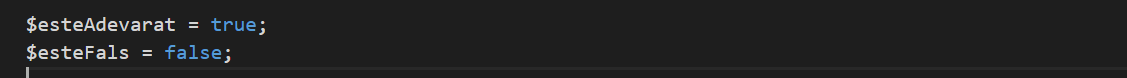
* **Structura unui script PHP**: Un script PHP începe cu **<?php** și se termină cu **?>**. Între aceste etichete, puteți introduce cod PHP.
* **Editorul de cod**: Puteți utiliza orice editor de text pentru a crea scripturile PHP, dar există și editori specializați pentru dezvoltare web care oferă funcționalități de evidențiere a sintaxei.
* **Rularea într-un browser**: Pentru a vedea rezultatele unui script PHP, trebuie să-l deschideți într-un browser web, fie pe un server web configurat, fie pe computerul local.
* **"Hello, World!" în PHP**: Un exemplu clasic este afișarea mesajului "Hello, World!" în PHP, pentru a vă asigura că totul funcționează corect.



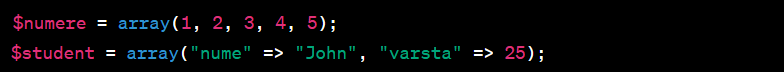
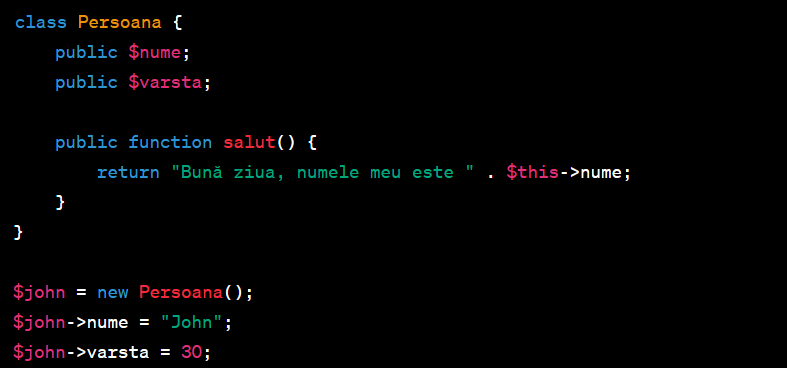
## **1.5 Tipuri de date**

Tipurile de date sunt un aspect fundamental în programarea PHP. Iată mai multe detalii despre tipurile de date și cum să le utilizați în PHP:

***1. Tipuri de date scalare***

* **int (integer)**: Acest tip de date este folosit pentru a stoca numere întregi, pozitive sau negative. De exemplu: 
* **float (floating-point)**: Acesta este folosit pentru numere zecimale (cu virgulă mobilă). De exemplu: 
* **string**: Tipul de date string este folosit pentru șiruri de caractere, cum ar fi textul. Puteți defini șiruri de caractere folosind ghilimele duble sau ghilimele simple: 
* **bool (boolean)**: Acesta este folosit pentru valori de adevăr (true) sau fals (false). De exemplu: 

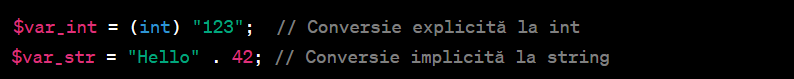
***2. Tipuri de date compuse***

* **array**: Un array este o colecție de valori indexate. Puteți avea array-uri indexate numeric sau asociative (cu chei în loc de indecși numerici): 
* **object**: Tipul de date obiect este utilizat pentru a crea obiecte în cadrul programării orientate pe obiect (OOP). Obiectele sunt instanțe ale claselor și pot avea proprietăți și metode. De exemplu: 

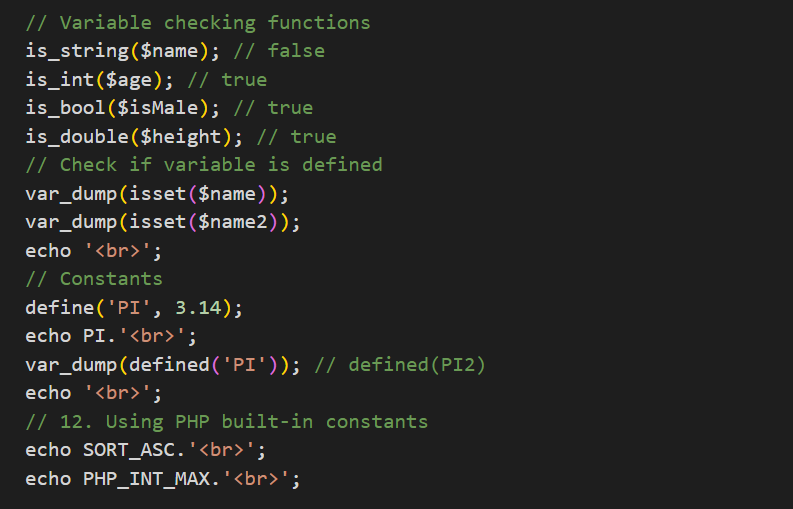
***3. Tipuri de date speciale***

* **resource**: Acest tip de date este utilizat pentru a reprezenta resurse externe, cum ar fi manipulatoarele pentru fișiere sau conexiuni la baze de date. De obicei, nu este necesar să lucrați direct cu tipul de date resource, deoarece este utilizat intern de funcțiile PHP pentru a gestiona resurse externe.
* **NULL**: Tipul de date NULL este utilizat pentru a indica că o variabilă nu are o valoare sau că este neinicializată.

***4. Conversii între tipuri de date***

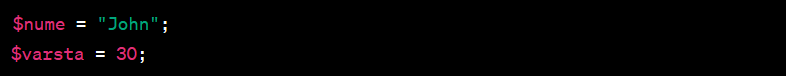
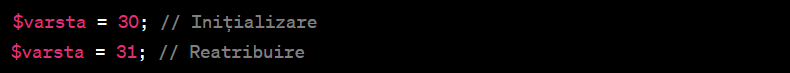
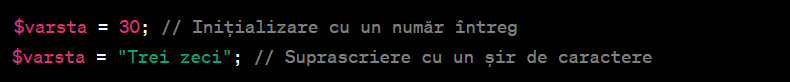
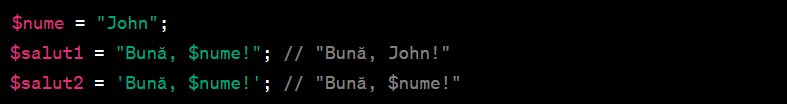
PHP permite conversii între tipuri de date, fie prin conversii explicite, fie prin conversii implicite. De exemplu: 

***5. Funcții pentru lucrul cu tipuri de date***

PHP oferă o serie de funcții pentru a lucra cu tipurile de date, cum ar fi **gettype()** pentru a obține tipul unei variabile, **is\_int()** pentru a verifica dacă o variabilă este de tip integer, **is\_string()** pentru a verifica dacă este un string, și altele. Aceste funcții sunt utile pentru gestionarea și validarea tipurilor de date în scripturile dvs.

## **1.6 Variabile în PHP**

Variabilele sunt folosite pentru a stoca și manipula date într-un program PHP. Ele sunt containeri care pot păstra valori de diferite tipuri de date, cum ar fi numere întregi, șiruri de caractere, valori boolene sau chiar obiecte complexe. Aici sunt câteva aspecte importante despre variabile în PHP:

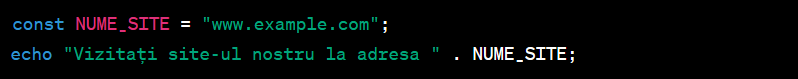
* **Declararea variabilelor**: În PHP, variabilele trebuie declarate înainte de a fi utilizate. Declararea se face prin prefixarea numelui variabilei cu simbolul **$**. De exemplu:
* **Tipuri de date flexibile**: PHP este un limbaj cu tipuri de date flexibile, ceea ce înseamnă că nu trebuie să specificați tipul de date al unei variabile la declarare. Tipul de date al unei variabile este determinat automat în funcție de valoarea pe care o conține.
* **Reatribuirea variabilelor**: Puteți reatribui o variabilă cu o altă valoare mai târziu în codul dvs. De exemplu: 
* **Suprascrierea variabilelor**: Puteți suprascrie valoarea unei variabile cu o altă valoare de un alt tip de date. De exemplu: 
* **Diferența dintre ghilimele duble și simple**: Când atribuiți valori șirurilor de caractere, puteți utiliza atât ghilimele duble (") cât și ghilimele simple ('). Ghilimelele duble interpretează variabilele și caracterele de escape, în timp ce ghilimelele simple tratează textul ca text pur. 

## **1.7 Constante în PHP**

Constantele sunt asemănătoare cu variabilele, dar acestea nu pot fi schimbate odată ce sunt definite. Ele sunt utile pentru a stoca valori constante care nu trebuie să fie modificate în timpul execuției programului. Aici sunt câteva aspecte importante despre constante în PHP:

* **Declararea constantelor**: Constantele sunt definite folosind funcția **define()** sau cu ajutorul operatorului **const**. De exemplu: Utilizând **define()**:   
  Utilizând **const**:  
  
* **Numele constantelor:** Convenția pentru numele constantelor este să fie în majuscule, cu cuvintele separate de subliniere. De exemplu: **NUME\_CONSTANTA**.
* **Accesarea constantelor:** Pentru a accesa o constantă, utilizați numele constantă fără simbolul **$**. De exemplu: 
* **Diferențe între variabile și constante**: Variabilele pot fi reatribuite cu alte valori, în timp ce constantele nu pot fi modificate după ce sunt definite. De asemenea, variabilele necesită simbolul **$** în fața numelui, în timp ce constantele nu.
* **Utilizarea constantelor**: Constantele sunt utile pentru a stoca valori fixe, cum ar fi adrese URL, chei de acces API sau orice altă valoare care trebuie să rămână constantă în timpul execuției programului.

Exemplu de utilizare a unei constante:



Variabilele și constantele sunt esențiale în dezvoltarea PHP, deoarece vă permit să stocați și să manipulați date în mod dinamic sau să definiți valori constante necesare în codul dvs. Alegerea între variabile și constante depinde de necesitățile specifice ale proiectului dvs. și de natura datelor pe care le manipulați.

## **1.8 Operatori**

În PHP, există o varietate de operatori care permit manipularea și evaluarea datelor în diferite moduri. Iată o listă cuprinzătoare a operatorilor din PHP:

**Operatori Aritmetici:**

1. **+** - Adunare
2. **-** - Scădere
3. **\*** - Înmulțire
4. **/** - Împărțire
5. **%** - Restul împărțirii (modulo)
6. **\*\*** - Exponențiere (ridicare la putere)

**Operatori de Asignare:**

1. **=** - Atribuire
2. **+=** - Adunare și atribuire
3. **-=** - Scădere și atribuire
4. **\*=** - Înmulțire și atribuire
5. **/=** - Împărțire și atribuire
6. **%=** - Modulo și atribuire
7. **\*\*=** - Exponențiere și atribuire

**Operatori de Comparare:**

1. **==** - Egal (verifică valoarea, nu tipul)
2. **===** - Strict egal (verifică valoarea și tipul)
3. **!=** sau **<>** - Diferit (verifică valoarea, nu tipul)
4. **!==** - Strict diferit (verifică valoarea și tipul)
5. **<** - Mai mic decât
6. **>** - Mai mare decât
7. **<=** - Mai mic sau egal
8. **>=** - Mai mare sau egal

**Operatori Logici:**

1. **&&** sau **and** - Și (ambii condiționali trebuie să fie adevărați)
2. **||** sau **or** - Sau (cel puțin unul dintre condiționali trebuie să fie adevărat)
3. **!** sau **not** - Negare (inversarea valorii logice)

**Operatori de Incrementare și Decrementare:**

1. **++** - Incrementare cu 1
2. **--** - Decrementare cu 1
3. **$a++** sau **++$a** - Diferență între post-incrementare și pre-incrementare (valoarea înainte sau după incrementare)

**Operatori de Șiruri:**

1. **.** - Concatenare (unirea a două șiruri)
2. **.=** - Concatenare și atribuire

**Operatori de Array:**

1. **+** - Unirea a două array-uri (duplicarea cheilor existente)
2. **==** - Egalitate a valorilor array-urilor
3. **===** - Egalitate strictă a array-urilor (compară și tipurile)
4. **!=** sau **<>** - Diferență a valorilor array-urilor
5. **!==** - Diferență strictă a array-urilor (compară și tipurile)

**Operatori de Control:**

1. **?:** - Operator ternar (expresie de decizie)
2. **??** - Operator de coalescență nulă (returnează prima valoare non-nulă dintr-o listă de expresii)
3. **[]** - Operator de indexare (pentru accesa elemente din array-uri și obiecte)
4. **::** - Operator de rezoluție a scopului (pentru accesarea membrilor statici ai unei clase)

**Alți Operatori:**

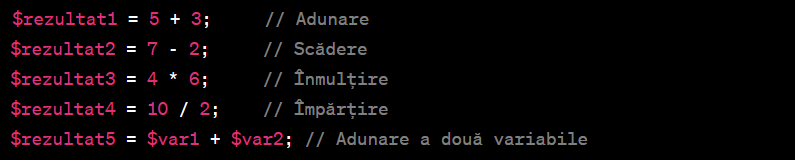
1. **new** - Operator pentru crearea unui nou obiect
2. **instanceof** - Operator pentru verificarea tipului de obiect
3. **yield** - Operator utilizat în generatoare pentru a produce valori
4. **yield from** - Operator folosit în generatoare pentru a delega yield-ul la un alt generator
5. **(...) => ...** - Operator de funcții anonime (arrow function) introdus în PHP 7.4

Aceasta este o prezentare generală a operatorilor în PHP. Este important să înțelegeți cum să utilizați acești operatori pentru a scrie cod eficient și clar în PHP.

## **1.9 Expresii**

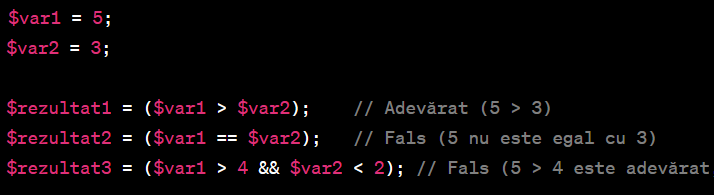
**1. Expresii aritmetice**

Expresiile aritmetice sunt utilizate pentru a efectua operații matematice, cum ar fi adunarea, scăderea, înmulțirea și împărțirea. Acestea pot conține numere, variabile și operatori aritmetici. Exemple de expresii aritmetice:



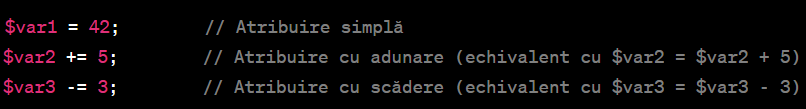
**2. Expresii logice**

Expresiile logice sunt utilizate pentru a evalua condiții și returnează rezultate booleane (adevărat sau fals). Acestea pot conține operatori de comparație, operatori logici și valori booleane. Exemple de expresii logice:



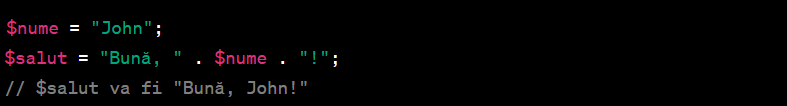
**3. Expresii de atribuire**

Expresiile de atribuire sunt utilizate pentru a atribui valori variabilelor. Acestea pot conține operatori de atribuire precum **=** sau **+=**. Exemple de expresii de atribuire:



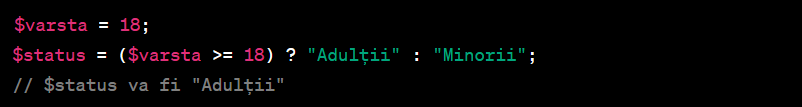
**4. Expresii de concatenare**

Expresiile de concatenare sunt utilizate pentru a lipi (concatena) șiruri de caractere. Acestea folosesc operatorul de concatenare **.** pentru a combina șiruri de caractere. Exemple de expresii de concatenare:



**5. Expresii de evaluare ternare**

Expresiile de evaluare ternare sunt utilizate pentru a lua decizii bazate pe condiții. Acestea au forma **condiție ? valoare1 : valoare2** și returnează **valoare1** dacă condiția este adevărată și **valoare2** în caz contrar. Exemplu:



**6. Expresii de evaluare a funcțiilor**

Expresiile de evaluare a funcțiilor sunt utilizate pentru a apela funcții și a returna rezultatele acestora. Acestea pot conține funcții predefinite sau funcții definite de utilizator. Exemplu:



**7. Expresii complexe**

Expresiile pot fi complexe și pot conține mai multe componente, inclusiv operații aritmetice, logice, de atribuire și funcții. Este important să aveți grijă la ordinea operațiilor și la paranteze pentru a asigura evaluarea corectă a expresiilor.



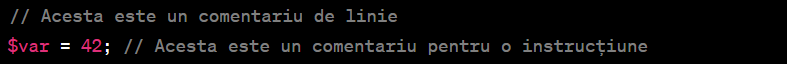
Expresiile sunt fundamentale în programarea PHP și sunt utilizate în mod extensiv pentru a efectua calcule, pentru a controla fluxul programului și pentru a evalua condiții. Înțelegerea corectă a expresiilor este esențială pentru dezvoltarea eficientă a codului PHP.

## **1.10 Comentarii**

Comentariile sunt porțiuni de text într-un program PHP care nu sunt interpretate și executate de către interpretorul PHP. Acestea sunt folosite pentru a adăuga explicații, documentație și note pentru dezvoltatori în codul sursă. Comentariile sunt importante pentru a face codul mai ușor de înțeles, de întreținut și de partajat cu alți dezvoltatori. Iată mai multe detalii despre comentarii în PHP:

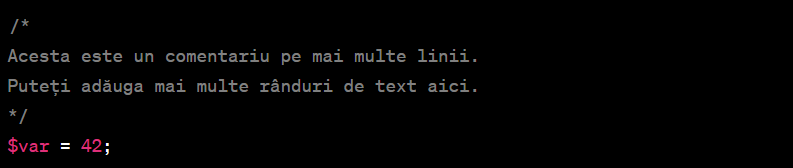
**1. Comentarii de linie**

Comentariile de linie sunt utilizate pentru a adăuga explicații pe o singură linie și încep cu **//**. Orice text de la **//** până la sfârșitul liniei este considerat comentariu și este ignorat de interpretorul PHP. Exemple:



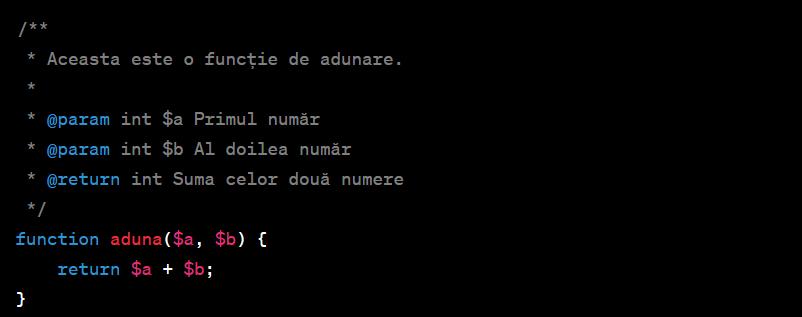
**2. Comentarii pe mai multe linii**

Comentariile pe mai multe linii sunt utilizate pentru a adăuga explicații pe mai multe linii și sunt delimitate de **/\*** și **\*/**. Orice text inclus între acești delimitatori este considerat comentariu și este ignorat de interpretorul PHP. Exemple:



**3. Comentarii de documentare**

Comentariile de documentare sunt utilizate pentru a documenta funcții, metode și clase într-un format specific. Acestea sunt utile pentru generarea automată a documentației. Comentariile de documentare încep cu **/\*\*** și pot conține etichete speciale precum **@param** și **@return**. Exemplu:



## **1.11 Array-uri**

Un array (sau tablou în română) este o structură de date în programare care permite stocarea și gestionarea unui set de valori sau elemente sub un singur nume. În PHP, array-urile sunt o parte esențială a limbajului și pot fi utilizate pentru a organiza și manipula date într-o varietate de moduri. Iată tot ce trebuie să știți despre array-urile în PHP:

**Array-uri Indexate Numeric (Iterative)**

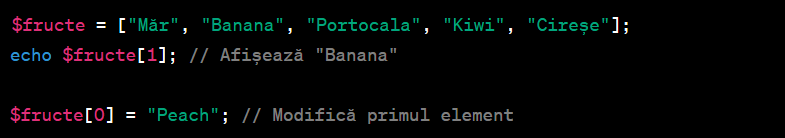
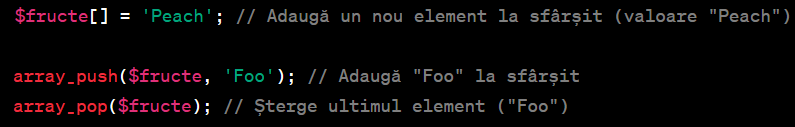
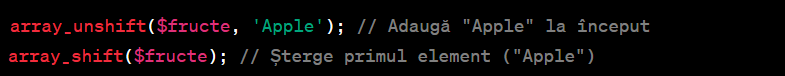
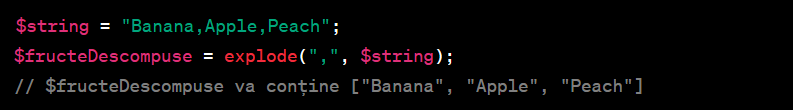
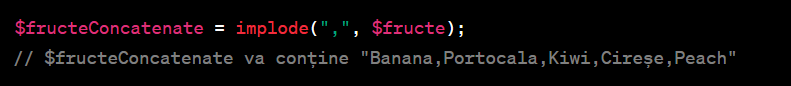
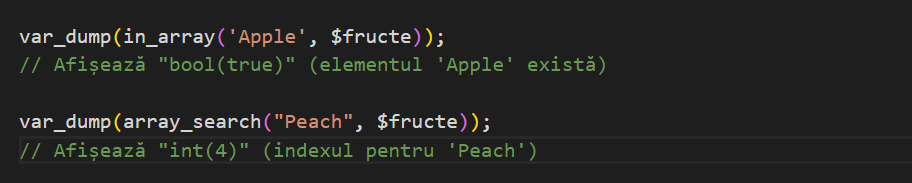
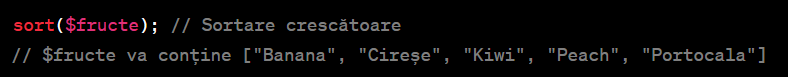
1. **Declararea și Inițializarea Array-urilor**:
   * Array-urile pot fi declarate și inițializate folosind sintaxa cu colțuri pătrate **[]** sau funcția **array()**.
2. **Accesarea și Modificarea Elementelor**:
   * Elementele din array-uri indexate numeric pot fi accesate și modificate folosind indici numerici. Indicii încep de la 0 și continuă în ordine numerică.
3. **Verificarea Existentei Elementelor**:
   * Puteți verifica dacă un element există într-un array la un anumit indice utilizând funcția **isset()**.
4. **Obținerea Lungimii Array-ului**:
   * Pentru a obține numărul de elemente dintr-un array, folosiți funcția **count()**.
5. **Adăugarea și Ștergerea Elementelor**:
   * Puteți adăuga un element la sfârșitul unui array folosind operatorul **[]** sau funcția **array\_push()**. Pentru a șterge ultimul element dintr-un array, utilizați funcția **array\_pop()**.
6. **Adăugarea și Ștergerea Elementelor la Începutul Array-ului**:
   * Puteți adăuga un element la începutul unui array cu funcția **array\_unshift()**, iar pentru a elimina primul element, utilizați funcția **array\_shift()**.
7. **Descompunerea unui Șir de Caractere într-un Array**:
   * Funcția **explode()** este folosită pentru a separa un șir de caractere în funcție de un delimiter și pentru a crea un array cu părțile separate.
8. **Concatenarea Elementelor unui Array într-un Șir de Caractere**:
   * Folosiți funcția **implode()** (sau **join()**) pentru a concatena elementele unui array într-un șir de caractere folosind un delimiter specificat.
9. **Verificarea Existentei unui Element într-un Array și Aflarea Indicelui său**:
   * Funcția **in\_array()** vă permite să verificați dacă un element există într-un array, iar funcția **array\_search()** vă ajută să aflați indicele sau cheia corespunzătoare a unui element.
10. **Sortarea unui Array**:
    * Puteți sorta un array folosind funcțiile **sort()** sau **rsort()** pentru sortarea crescătoare sau descrescătoare, respectiv.

**Array-uri Asociative**

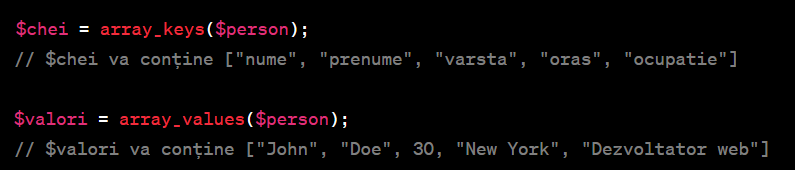
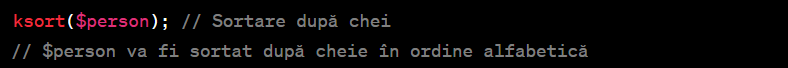
1. **Declararea și Inițializarea Array-urilor Asociative**:
   * Array-urile asociative sunt definite prin perechi cheie => valoare, în care cheile pot fi personalizate și nu sunt limitate la valori numerice.
2. **Accesarea și Modificarea Valorilor prin Chei**:
   * Elementele din array-uri asociative sunt accesate și modificate prin specificarea cheii asociate valorii.
3. **Verificarea Existentei unei Chei într-un Array Asociativ**:
   * Puteți verifica dacă o cheie există într-un array asociativ utilizând funcția **isset()** sau prin folosirea operatorului **??=** (disponibil în PHP 7.4) pentru a furniza o valoare implicită.
4. **Obținerea Listelor de Chei și Valori dintr-un Array Asociativ**:
   * Funcțiile **array\_keys()** și **array\_values()** vă permit să obțineți lista de chei sau valori dintr-un array asociativ.
5. **Sortarea Array-urilor Asociative**:
   * Puteți sorta array-uri asociative folosind funcții precum **ksort()** (sortare după chei), **krsort()** (sortare descrescătoare după chei), **asort()** (sortare după valori) sau **arsort()** (sortare descrescătoare după valori).

Array-urile sunt structuri de date esențiale în PHP, utilizate frecvent în dezvoltarea de aplicații web și pentru gestionarea și organizarea datelor. Ele pot fi utilizate în diverse moduri pentru a stoca și manipula informații în cadrul aplicațiilor PHP.

**Array-uri Indexate Numeric (Iterative)**

1. **Accesarea și Modificarea Elementelor**: 
2. **Verificarea Existentei Elementelor:**
3. **Adăugarea și Ștergerea Elementelor: **
4. **Adăugarea și Ștergerea Elementelor la Începutul Array-ului: **
5. **Descompunerea unui Șir de Caractere într-un Array: **
6. **Concatenarea Elementelor unui Array într-un Șir de Caractere: **
7. **Verificarea Existentei unui Element într-un Array și Aflarea Indicelui său: **
8. **Sortarea unui Array: **

**Array-uri Asociative**

1. **Accesarea și Modificarea Valorilor prin Chei**: 
2. **Verificarea Existentei unei Chei într-un Array Asociativ**: 
3. **Obținerea Listelor de Chei și Valori dintr-un Array Asociativ**: 
4. **Sortarea Array-urilor Asociative**: 

## **Test de evaluare**

**1. Ce semnificație are operatorul === în PHP?**

a) Efectuează o adunare

b) Verifică egalitatea strictă

c) Verifică egalitatea fără a verifica tipul

d) Nu este un operator valid

**2. Cum se definește o constantă în PHP?**

a) Cu operatorul =

b) Cu operatorul ==

c) Cu funcția define()

d) Cu funcția const()

**3. Care este operatorul pentru adunare în PHP?**

a) +

b) -

c) /

d) \*

**4. Ce reprezintă expresiile logice în PHP?**

a) Calcule matematice

b) Evaluarea condițiilor

c) Concatenarea șirurilor de caractere

d) Atributele funcțiilor

**5. Care este scopul principal al comentariilor de documentare în PHP?**

a) Atribuirea valorilor la variabile

b) Scrierea codului sursă

c) Documentarea funcțiilor, metodelor și claselor

d) Evaluarea expresiilor logice

**6. Cum se marchează un comentariu de documentare în PHP?**

a) Cu //

b) Cu /\* \*/

c) Cu /\*\* \*/

d) Cu #

**7. Ce face operatorul % în PHP?**

a) Înmulțește două valori

b) Returnează restul unei împărțiri

c) Scade două valori

d) Concatenează două șiruri de caractere

**8. Care este tipul de date folosit pentru stocarea numerelor întregi în PHP?**

a) int

b) float

c) string

d) bool

**9. Cum se pot adăuga comentarii pe mai multe linii în PHP?**

a) Cu //

b) Cu /\* \*/

c) Cu /\*\* \*/

d) Cu #

**10. Ce se întâmplă atunci când se utilizează operatorul de atribuire (=) în PHP?**

a) Se efectuează o adunare

b) Se atribuie o valoare unei variabile

c) Se compară două valori

d) Se concatenează șiruri de caractere

**11. Care este operatorul de concatenare în PHP?**

a) +

b) -

c) .

d) /

**12. Ce tip de date poate stoca o variabilă în PHP?**

a) Numere întregi și șiruri de caractere

b) Numere întregi și booleeni

c) Toate tipurile de date

d) Doar șiruri de caractere

**13. Cum se poate inversa valoarea unei variabile booleane în PHP?**

a) Cu operatorul &&

b) Cu operatorul ||

c) Cu operatorul !

d) Cu operatorul ==

**14. Care dintre următoarele tipuri de date nu este scalabil în PHP?**

a) int (integer)

b) float (floating-point)

c) array (tablou)

d) object (obiect)

**15. Care este scopul principal al comentariilor temporare în PHP?**

a) Documentarea funcțiilor și claselor

b) Dezactivarea temporară a codului

c) Adăugarea explicațiilor pentru dezvoltatori

d) Evaluarea expresiilor logice

**Răspunsuri:**

b) Verifică egalitatea strictă

c) Cu funcția define()

a) +

b) Evaluarea condițiilor

c) Documentarea funcțiilor, metodelor și claselor

c) Cu /\*\* \*/

b) Returnează restul unei împărțiri

a) int

b) Cu /\* \*/

b) Se atribuie o valoare unei variabile

c) .

c) Toate tipurile de date

c) Cu operatorul !

d) object (obiect)

b) Dezactivarea temporară a codului

## **Probleme practice**

1. Creați o variabilă pentru fiecare tip de date în PHP (***integer, float, string, boolean***) și atribuiți-le valori.
2. Declarați o variabilă și afișați tipul de date al acelei variabile.
3. Declarați o variabilă de șir de caractere și afișați lungimea acestui șir.
4. Combinați două șiruri de caractere diferite într-unul singur și afișați rezultatul.
5. Declarați un tablou cu 5 elemente și afișați-l folosind o buclă for.

# **Modulul 2: Instrucțiuni**

**Unităţi de conţinut**

1. Instrucțiuni condiționale.
2. Instrucțiuni repetitive.
3. Instrucțiuni de salt.
4. Instrucțiuni de includere și evaluare a fișierelor.

**Abilități**

A10. Prelucrarea datelor cu ajutorul instrucțiunilor condiționale, repetitive și de salt.

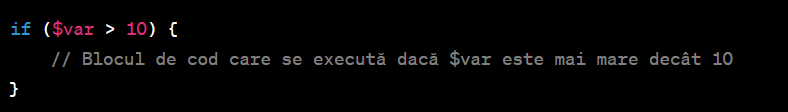
A11. Includerea fișierelor într-un document Web.

**Cuvinte cheie:**

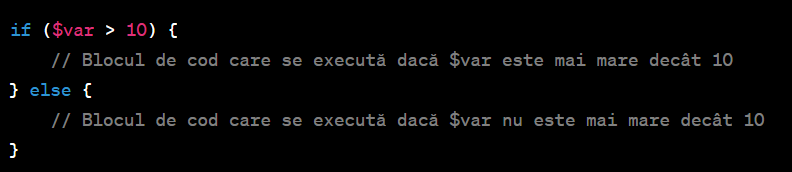
## **2.1 Instrucțiuni condiționale:**

Instrucțiunile condiționale sunt utilizate în programare pentru a permite executarea unor bucăți specifice de cod numai dacă o anumită condiție este adevărată. În PHP, cele mai frecvente instrucțiuni condiționale sunt "if", "else" și "elseif".

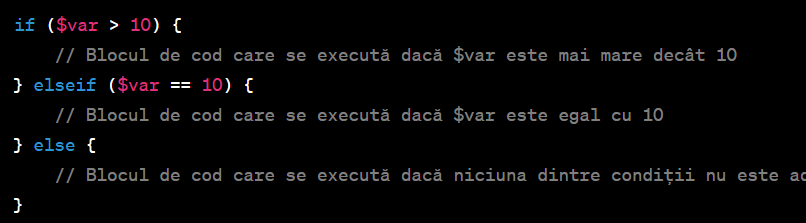
* **if**: Instrucțiunea "if" este folosită pentru a verifica o condiție. Dacă condiția este adevărată, se execută un bloc de cod specificat. De exemplu:



**else**: Instrucțiunea "else" este folosită în combinație cu "if" pentru a furniza un bloc de cod care se execută atunci când condiția "if" este falsă. De exemplu:



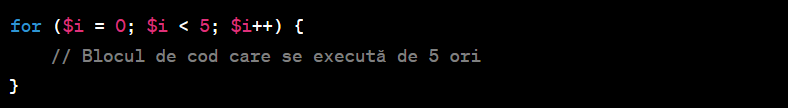
**elseif**: Instrucțiunea "elseif" poate fi utilizată pentru a verifica mai multe condiții într-o succesiune. Dacă prima condiție este falsă, se trece la următoarea condiție. De exemplu:



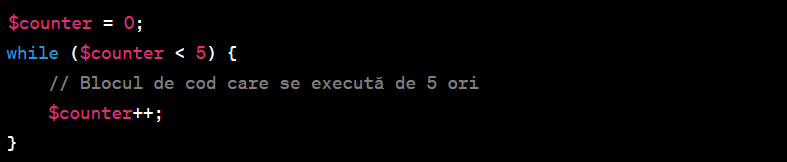
## **2.2 Instrucțiuni repetitive:**

Instrucțiunile repetitive (buclele) permit executarea repetată a unui bloc de cod. În PHP, cele mai comune tipuri de bucle sunt "for", "while" și "foreach".

* **for**: Instrucțiunea "for" este folosită atunci când cunoaștem numărul exact de iterații. Aceasta constă dintr-un inițializator, o condiție de oprire și un increment. De exemplu:



**while**: Instrucțiunea "while" este utilizată atunci când numărul de iterații depinde de o condiție specifică. Blocul de cod se execută atât timp cât condiția este adevărată. De exemplu:



## **2.3 Instrucțiuni de salt**

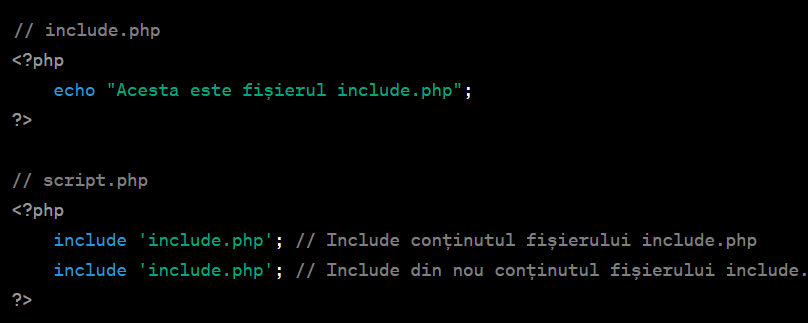
Instrucțiunile de salt sunt utilizate pentru a modifica fluxul normal de execuție al programului.

* **break**: Instrucțiunea "break" este folosită pentru a întrerupe execuția unei bucle (cum ar fi "for" sau "while") atunci când o anumită condiție este îndeplinită. Aceasta permite ieșirea prematură din buclă.
* **continue**: Instrucțiunea "continue" este utilizată pentru a sări la următoarea iterație a unei bucle atunci când o anumită condiție este îndeplinită. Aceasta permite sărirea peste restul codului din iterația curentă.

## **1.4 Instrucțiuni de includere și evaluare a fișierelor:**

Instrucțiunile de includere și evaluare a fișierelor sunt utilizate pentru a organiza și reutiliza codul într-un program.

* **include** și **require**: Aceste instrucțiuni permit includerea altor fișiere PHP într-un script principal. Diferența principală între ele este că "include" afișează o avertizare și continuă execuția dacă fișierul inclus nu poate fi găsit, în timp ce "require" generează o eroare fatală și oprește execuția scriptului.
* **include\_once** și **require\_once**: Aceste funcții sunt utilizate pentru a asigura că un fișier este inclus doar o dată într-un script, indiferent de câte ori este apelată. Aceasta previne includerea duplicatelor.
* **include** și **require** sunt folosite pentru a încorpora conținutul unui fișier într-un alt fișier PHP. De exemplu, **include 'fisier.php';** va include conținutul fișierului **fisier.php** în locul în care a fost apelată această instrucțiune. Dacă fișierul nu poate fi găsit, **include** va emite o avertizare, iar programul va continua să ruleze. Pe de altă parte, **require 'fisier.php';** face același lucru, dar dacă fișierul nu poate fi găsit, va genera o eroare fatală și va opri execuția programului.
* **include\_once** și **require\_once** sunt utilizate pentru a asigura că un fișier este inclus doar o dată în timpul execuției programului, chiar dacă este apelat mai multe ori. De exemplu, **include\_once 'fisier.php';** va include **fisier.php** doar prima dată când este apelat și va ignora apelurile ulterioare.



În acest exemplu, **include 'include.php';** este apelat de două ori, dar doar prima dată când este apelat, conținutul din **include.php** este inclus. Apelul ulterior nu va modifica comportamentul.

Instrucțiunile de includere și evaluare a fișierelor sunt utile pentru organizarea și reutilizarea codului, permițând împărțirea unui program mare în fișiere mai mici și gestionarea dependențelor. Acestea pot fi folosite, de asemenea, pentru a încorpora biblioteci externe sau funcționalități suplimentare într-un proiect PHP.

## **Test de evaluare**

**1. Care dintre următoarele nu este un tip scalar în PHP?** a) bool b) string c) array d) int

**2. Ce face instrucțiunea "else" într-o structură "if" în PHP?** a) Rulează întotdeauna blocul de cod asociat cu "if". b) Rulează atunci când condiția din "if" este adevărată. c) Rulează atunci când condiția din "if" este falsă. d) Nu are nicio funcție specifică.

**3. Ce funcție este utilizată pentru a obține tipul unei variabile în PHP?** a) getType() b) getTypeOf() c) typeof() d) gettype()

**4. Care dintre următoarele bucle este folosită atunci când numărul exact de iterații este cunoscut?** a) for b) while c) do-while d) foreach

**5. Care instrucțiune de salt este folosită pentru a ieși dintr-o buclă în PHP?** a) jump b) skip c) exit d) break

**6. Ce face funcția include\_once în PHP?** a) Include un fișier doar o dată în timpul execuției programului. b) Include un fișier de fiecare dată când este apelată. c) Include un fișier și îl evaluează înainte de a continua. d) Include un fișier și îl interpretează ca HTML.

**7. Ce tip de date este utilizat pentru stocarea unui set de valori asociate în PHP?** a) int b) string c) array d) float

**8. Cum se poate verifica dacă două variabile au aceeași valoare și același tip de date în PHP?** a) Utilizând operatorul **==** b) Utilizând operatorul **===** c) Utilizând operatorul **=** d) Nu se poate verifica acest lucru în PHP.

**9. Care funcție este utilizată pentru a afișa un mesaj în consola de dezvoltare (consola de devtools) dintr-un script PHP?** a) print() b) echo() c) console.log() d) debug()

**10. Ce face funcția unset() în PHP?** a) Dezactivează o variabilă. b) Șterge o variabilă. c) Eliberează memoria alocată pentru o variabilă. d) Schimbă valoarea unei variabile.

**11. Care este diferența dintre GET și POST în HTTP?** a) **GET** este folosit pentru a trimite date către server, iar **POST** pentru a primi date de la server. b) **GET** este mai securizat și protejat decât **POST**. c) **GET** atașează date la URL, în timp ce **POST** le transmite în corpul cererii. d) Nu există diferență între ele.

**12. Ce face funcția header() în PHP?** a) Afisează un antet HTTP personalizat. b) Redirectează utilizatorul la o altă pagină web. c) Schimbă fontul și stilul textului din pagină. d) Generează un antet JSON.

**13. Ce instrucțiune este folosită pentru a gestiona excepțiile în PHP?** a) try...catch b) if...else c) switch d) throw...catch

**14. Care este scopul unei sesiuni în PHP?** a) Să stocheze date pe server pentru utilizatori. b) Să permită utilizatorilor să-și schimbe parolele. c) Să transmită date prin URL. d) Să stocheze cookie-uri.

**15. Ce face funcția mysqli\_query() în PHP?** a) Realizează o interogare la o bază de date MySQL. b) Afisează un mesaj pe ecran. c) Crează o interfață grafică pentru utilizator. d) Șterge un fișier din sistemul de fișiere.

## **Probleme practice**

**Instrucțiuni condiționale:**

1. Scrieți un program PHP care verifică dacă un număr este par sau impar și afișează un mesaj corespunzător.
2. Implementați un script care verifică dacă un utilizator este autentificat și îl redirecționează către pagina de autentificare dacă nu este.
3. Creați un program care compară trei numere și afișează cel mai mare dintre ele.
4. Realizați un calculator simplu care efectuează operații matematice (adunare, scădere, înmulțire, împărțire) în funcție de input-ul utilizatorului.
5. Dezvoltați un script care determină dacă un anumit an este bisect sau nu, folosind instrucțiuni condiționale.

**Instrucțiuni repetitive:**

1. Scrieți un program care afișează toate numerele pare de la 1 la 100 folosind o buclă "for".
2. Implementați un script care calculează suma primelor 50 de numere naturale folosind o buclă "while".
3. Creați un joc simplu de ghicit numerele, în care calculatorul alege un număr aleatoriu, iar utilizatorul încearcă să ghicească numărul într-o buclă "do-while".
4. Realizați un program care afișează tabelul înmulțirii de la 1 la 10 folosind o buclă "for".
5. Dezvoltați un script care generează o secvență Fibonacci pentru primele n numere, unde n este introdus de utilizator.

**Instrucțiuni de salt:**

1. Scrieți un program care afișează toate numerele pare mai mici decât 20, utilizând o instrucțiune "break" pentru a opri bucla când un număr impar este întâlnit.
2. Implementați un script care calculează suma numerelor de la 1 la 100, dar sare peste numerele divizibile cu 5, folosind instrucțiunea "continue".
3. Creați un joc simplu de cuvinte încrucișate în care un utilizator începe să completeze cuvinte, dar poate să sara peste un cuvânt dacă se blochează.
4. Realizați un program care începe cu o listă de numere și utilizează o buclă "foreach" pentru a le afișa, dar folosește instrucțiunea "continue" pentru a omite numerele negative.
5. Dezvoltați un sistem de gestionare a utilizatorilor în care se utilizează o instrucțiune "break" pentru a ieși din buclă atunci când un utilizator specific este găsit.

# **Modulul 3: Funcții**

**Unităţi de conţinut**

1. Funcții predefinite.
2. Funcții definite de utilizator.
3. Funcții variabile.
4. Domeniul de vizibilitate a variabilelor.

**Abilități**

A12. Utilizarea funcțiilor predefinite.

A13. Definirea funcțiilor proprii.

A14. Utilizarea funcțiilor pentru prelucrarea datelor din documente web.

**Cuvinte cheie:**

1. **Funcții**: Blocuri de cod reutilizabile pentru realizarea unor acțiuni specifice.
2. **Parametri**: Variabile folosite pentru transmiterea datelor către funcții.
3. **Return**: Instrucțiune folosită pentru a furniza rezultatul unei funcții.
4. **Recursivitate**: Capacitatea unei funcții de a se apela pe sine.
5. **Tipuri de date**: Specificarea tipului de date așteptat pentru parametrii și valorile returnate în funcții.
6. **Funcții anonime**: Funcții fără nume, cunoscute și sub numele de funcții lambda.
7. **Funcții predefinite**: Funcții incluse în limbajul PHP pentru a efectua diverse operațiuni.
8. **Funcții arrow**: Funcții scurte și concise introduse în PHP 7.4.
9. **Namespace**: Mecanism pentru organizarea și izolarea funcțiilor și claselor în PHP.
10. **Validare de tipuri**: Verificarea și asigurarea corectitudinii tipurilor de date în funcții.

Funcțiile sunt componente esențiale în programarea PHP (și în multe alte limbaje de programare) și joacă un rol crucial în organizarea și structurarea codului. Ele permit gruparea instrucțiunilor în blocuri logice, reutilizarea codului și facilitarea întreținerii. În continuare, voi acoperi totul despre funcții în PHP.

## **3.1 Ce este o funcție în PHP?**

În PHP, o funcție este un bloc de cod care poate fi definit și apoi apelat de mai multe ori într-un program. Funcțiile sunt folosite pentru a grupa instrucțiunile într-un singur loc și pentru a le executa atunci când sunt necesare. O funcție poate accepta zero sau mai mulți parametri, efectua operații și returna un rezultat.

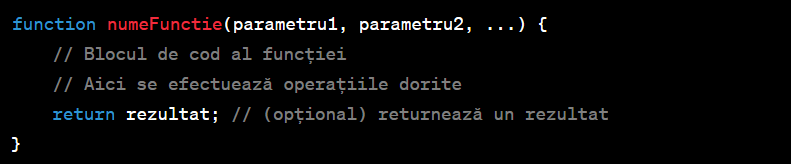
**De ce sunt funcțiile importante?**

Funcțiile au mai multe beneficii:

1. **Reutilizare**: Puteți defini o funcție o singură dată și apoi să o apelați de mai multe ori în diferite părți ale codului dvs. Acest lucru permite reutilizarea și evitarea duplicării codului.
2. **Structurare**: Funcțiile ajută la organizarea codului în blocuri logice mai mici și mai ușor de gestionat. Acestea fac codul mai ușor de citit și de înțeles.
3. **Întreținere**: Dacă aveți nevoie să faceți modificări la o funcție, trebuie să o actualizați doar o dată, iar modificările vor fi reflectate oriunde este apelată funcția.
4. **Abstractizare**: Funcțiile pot ascunde detaliile implementării și pot oferi o interfață simplificată pentru a interacționa cu codul.

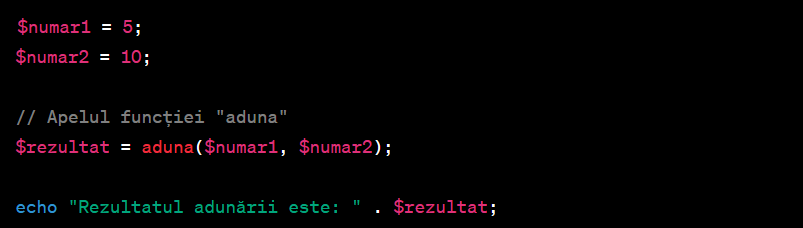
## **3.2 Definirea unei funcții în PHP**

O funcție în PHP este definită utilizând cuvântul cheie **function**, urmat de numele funcției și lista de parametri (opțională) între paranteze, precum și blocul de cod în interiorul acoladelor **{}**. Iată o sintaxă generală pentru definirea unei funcții:



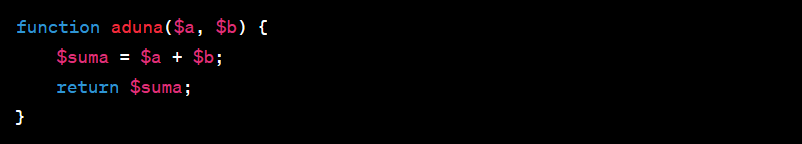
**Apelarea unei funcții**

După ce ați definit o funcție, o puteți apela în codul dvs. pentru a executa instrucțiunile din interiorul funcției. Apelul funcției se face folosind numele funcției urmat de paranteze **()**, în care puteți furniza argumentele corespunzătoare parametrilor funcției (dacă există).



## **3.3 Parametrii funcției**

Parametrii funcției sunt variabilele utilizate în interiorul funcției pentru a primi date de la apelator. Parametrii sunt specificați în lista de parametri la definirea funcției și pot fi utilizați în interiorul funcției ca oricare alte variabile.



În acest exemplu, **$a** și **$b** sunt parametrii funcției **aduna**, iar funcția returnează suma lor.

## **3.4 Tipurile de returnare**

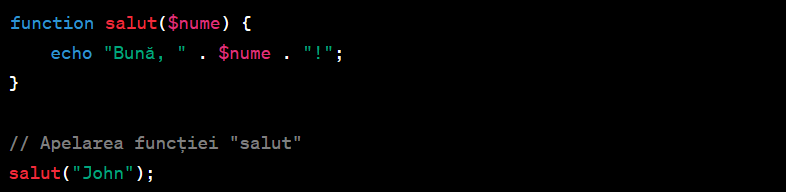
O funcție poate returna un rezultat utilizând instrucțiunea **return**. Aceasta permite funcției să furnizeze o valoare către locul în care este apelată. O funcție poate avea sau nu o instrucțiune **return**, iar în cazul în care lipsește sau dacă **return** este apelat fără argument, funcția va returna **null**.

**Funcții predefinite în PHP**

PHP dispune de numeroase funcții predefinite (built-in functions) care sunt disponibile pentru utilizare fără a necesita definirea lor. Acestea includ funcții pentru lucrul cu șiruri de caractere, lucrul cu fișiere, manipularea datelor și multe altele. Exemple de funcții predefinite includ **strlen()**, **file\_get\_contents()**, **date()**, **array\_push()**, **explode()** și multe altele.

**Funcții definite de utilizator**

În plus față de funcțiile predefinite, puteți defini propriile funcții în PHP, așa cum am arătat mai sus. Funcțiile definite de utilizator sunt funcții pe care le creați în codul dvs. pentru a îndeplini anumite sarcini sau pentru a organiza mai bine codul.

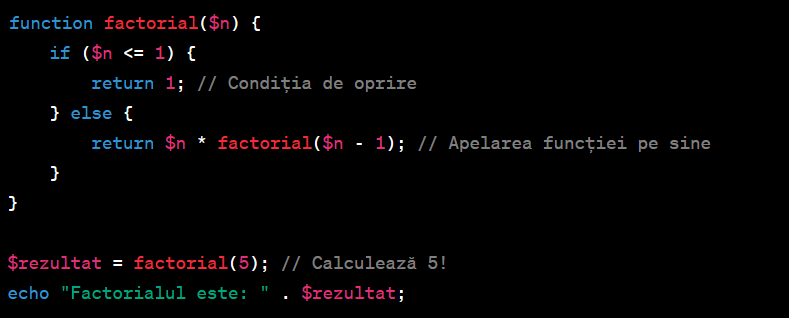


În acest exemplu, funcția **salut()** acceptă un parametru **$nume** și afișează un mesaj de salut utilizând acest parametru.

## **3.5 Funcții recursive**

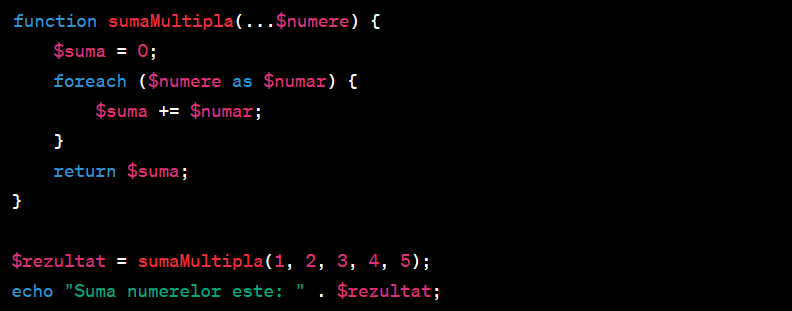
În PHP, puteți defini funcții recursive, adică funcții care se apelează pe sine. Acestea sunt utile pentru rezolvarea problemelor matematice, de listare sau de stivă care necesită o abordare recursivă. Este important să aveți o condiție de oprire într-o funcție recursivă pentru a evita bucle infinite.

**Exemplu de funcție recursivă**



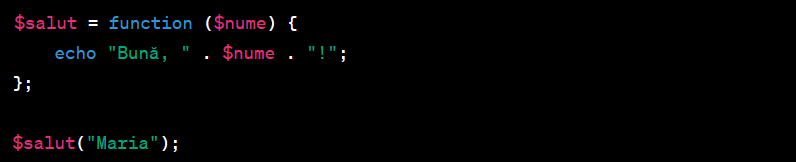
**Funcții cu mai mulți parametri**

Funcțiile în PHP pot avea mai mulți parametri. Puteți defini funcții cu un număr variabil de parametri utilizând operatorul **...** înaintea numelui parametrului. Acești parametri vor fi tratați ca un array în interiorul funcției.



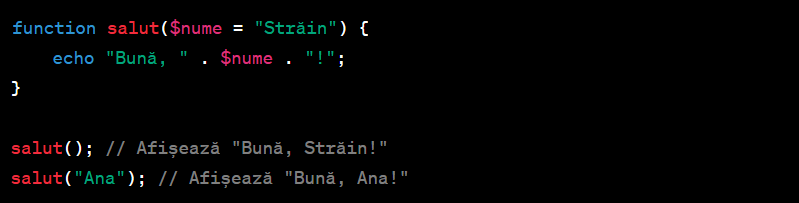
## **3.5 Funcții anonime**

PHP permite definirea funcțiilor anonime (sau funcții lambda) care pot fi stocate în variabile sau utilizate în alte funcții ca argumente. Acestea sunt utile în special în cazul funcțiilor callback și funcțiilor de înaltă ordine.



**Funcții cu valori implicite pentru parametri**

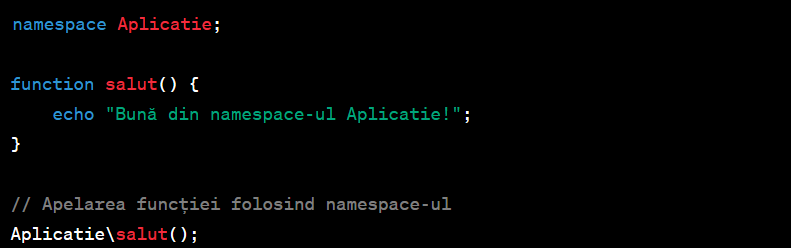
Puteți atribui valori implicite parametrilor funcției. Aceste valori vor fi folosite dacă apelatorul funcției nu furnizează un argument pentru parametrul respectiv.



**Namespace și funcții**

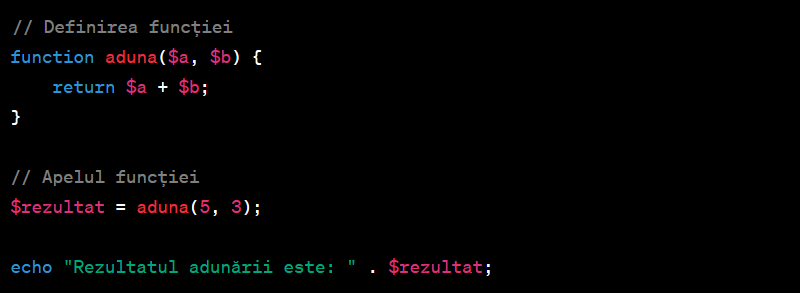
În PHP, puteți organiza funcțiile în namespace-uri pentru a evita conflictul de denumire. Acest lucru este util în dezvoltarea de aplicații complexe sau în colaborarea cu alte coduri.

**Exemplu de funcție dintr-un namespace**



**Folosirea funcțiilor într-un script PHP**

Funcțiile sunt esențiale pentru crearea oricărui script PHP. În mod obișnuit, un script PHP va consta în definiția și apelul funcțiilor pentru a realiza sarcinile necesare.



În acest exemplu, funcția **aduna()** este definită pentru a efectua o adunare, iar apoi este apelată pentru a calcula suma a două numere.

## **3.6 Funcțiile arrow**

Funcțiile arrow sunt o caracteristică relativ nouă introdusă în PHP 7.4 și oferă o sintaxă mai concisă și mai clară pentru definirea funcțiilor anonime (funcții lambda). Aceste funcții sunt cunoscute și sub numele de "arrow functions" sau "short functions". Funcțiile arrow au fost adăugate în PHP pentru a facilita definirea și utilizarea funcțiilor în mai multe contexte.

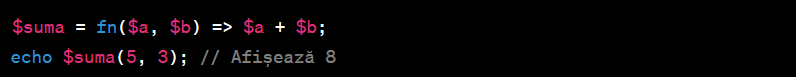
**Sintaxa funcțiilor arrow în PHP**

Sintaxa pentru definirea funcțiilor arrow este foarte simplă și constă într-o expresie de tip lambda:



* **$functie**: Variabila în care este stocată funcția arrow.
* **$parametru1, $parametru2, ...**: Parametrii funcției, similar cu funcțiile obișnuite.
* **expresie**: O singură expresie care reprezintă corpul funcției. Această expresie este evaluată și rezultatul este returnat implicit.

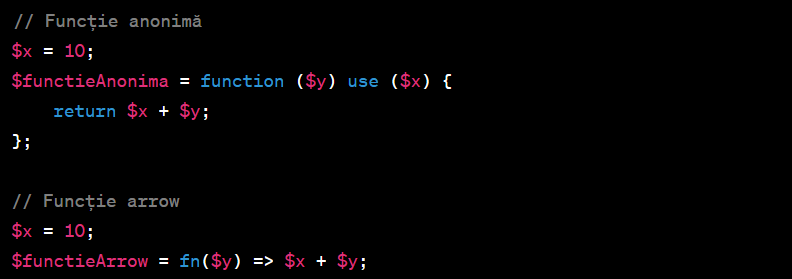
Iată un exemplu simplu:



În acest exemplu, am definit o funcție arrow care adună două numere.

**Avantajele funcțiilor arrow**

1. **Sintaxă concisă**: Funcțiile arrow au o sintaxă mai concisă decât funcțiile anonime obișnuite, ceea ce face codul mai ușor de citit și de scris.
2. **Nu este nevoie de cuvântul cheie use**: În funcțiile anonime, când doriți să utilizați variabile din afara funcției, trebuie să le adăugați în cadrul cuvântului cheie **use**. Cu funcțiile arrow, acest lucru nu mai este necesar, deoarece ele preiau automat variabilele din contextul în care sunt definite.



1. **Returnare implicită**: Funcțiile arrow returnează implicit rezultatul expresiei, fără a necesita utilizarea cuvântului cheie **return**.
2. **Pot fi utilizate în metode și funcții de înaltă ordine**: Funcțiile arrow pot fi utilizate în contexte în care aveți nevoie de funcții ca argumente pentru alte funcții sau metode.

**Limitări ale funcțiilor arrow**

Cu toate avantajele lor, funcțiile arrow au și unele limitări:

1. **Nu pot conține multiple instrucțiuni**: Funcțiile arrow sunt limitate la o singură expresie și nu pot conține mai multe instrucțiuni.
2. **Nu pot avea nume**: Funcțiile arrow sunt anonime și nu pot fi numite. Prin urmare, nu puteți apela o funcție arrow recursiv.

**Când să utilizați funcțiile arrow în PHP?**

Funcțiile arrow sunt utile în special pentru funcții simple, cu o singură expresie, în care nu este necesară logica complexă sau instrucțiuni multiple. Ele sunt folosite adesea în contexte de mapare și filtrare a tablourilor, în funcții callback și în alte situații în care vă doriți o sintaxă mai concisă.

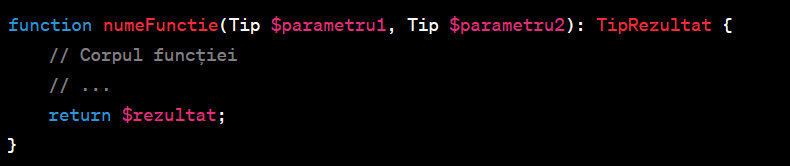
În concluzie, funcțiile arrow reprezintă o adiție utilă și modernă la limbajul PHP, permițându-vă să definiți funcții mai succinte și mai ușor de citit. Cu toate acestea, este important să înțelegeți limitările lor și să le utilizați în mod corespunzător în codul dvs.

## **3.7 Definirea tipurilor de date pentru parametrii funcțiilor și valorile returnate**

Începând cu PHP 7.0, limbajul PHP permite definirea tipurilor de date pentru parametrii funcțiilor și valorile returnate. Acest lucru ajută la îmbunătățirea clarității codului și la evitarea erorilor de tipuri. Iată mai multe detalii despre funcțiile cu tipuri de date în PHP:

**Sintaxa funcțiilor cu tipuri de date**

Pentru a defini tipurile de date pentru parametrii funcției, puteți utiliza sintaxa următoare:



În acest exemplu, am definit o funcție numită **aduna** care acceptă doi parametri de tip întreg (**int**) și returnează un rezultat de tip întreg.

**Tipurile de date acceptate**

PHP acceptă o gamă variată de tipuri de date în definirea funcțiilor:

* Tipuri scalare: **int**, **float**, **string**, **bool**.
* Tipuri compuse: **array**, **object**, **callable**.
* Tipuri personalizate: puteți defini propriile clase și interfețe și să le utilizați ca tipuri de date în funcții.

**Avantajele funcțiilor cu tipuri de date**

1. **Claritate**: Tipurile de date oferă claritate asupra tipului de date așteptat pentru parametrii funcției și a tipului de date returnat. Acest lucru face codul mai ușor de înțeles și de întreținut.
2. **Detectarea erorilor de tipuri**: Folosirea tipurilor de date ajută la detectarea mai rapidă a erorilor de tipuri în cod. PHP va genera erori sau excepții atunci când tipurile nu corespund, ceea ce ajută la depistarea problemelor mai devreme în procesul de dezvoltare.

**Limitări ale funcțiilor cu tipuri de date**

Cu toate acestea, este important să țineți cont de câteva limitări:

1. **Compatibilitate cu versiunile anterioare**: Funcțiile cu tipuri de date sunt disponibile începând cu PHP 7.0, deci dacă lucrați cu versiuni mai vechi ale PHP, această caracteristică nu este disponibilă.
2. **Tipuri personalizate**: Deși puteți defini tipuri de date personalizate pentru clase și interfețe, acest lucru poate adăuga complexitate și poate necesita o planificare atentă pentru a asigura compatibilitatea între codul dvs. și tipurile personalizate.

**Concluzie**

Funcțiile cu tipuri de date sunt o caracteristică utilă în PHP, care ajută la creșterea clarității și la reducerea erorilor de tipuri în codul dvs. Această caracteristică este importantă în dezvoltarea de aplicații mari și complexe, dar trebuie utilizată cu atenție și să țineți cont de compatibilitatea cu versiunile PHP pe care le utilizați.

## **Test de evaluare**

1. Ce sunt funcțiile în PHP?
   1. Variabile globale
   2. Instrucțiuni pentru bucle
   3. Blocuri de cod care pot fi apelate repetat pentru a realiza o anumită acțiune
   4. Comentarii în cod
2. Ce face sintaxa unei funcții în PHP să fie corectă?
   1. Există cel puțin un comentariu în interiorul funcției
   2. Numele funcției trebuie să fie scris cu litere mici
   3. Trebuie să includă întotdeauna un tip de date între paranteze
   4. Trebuie să includă întotdeauna un corp de funcție între acoladele {}
3. Cum apelați o funcție în PHP?
   1. Prin includerea numelui funcției într-un comentariu
   2. Prin adăugarea semnului de exclamare (!) în fața numelui funcției
   3. Prin utilizarea numelui funcției urmat de paranteze ()
   4. Prin crearea unei variabile cu numele funcției
4. Ce este un parametru de funcție în PHP?
   1. Un nume de funcție
   2. O valoare întoarsă de funcție
   3. O variabilă folosită pentru a transmite date către funcție
   4. O variabilă folosită pentru a păstra date în interiorul funcției
5. Ce reprezintă cuvântul cheie "return" într-o funcție PHP?
   1. Un tip de date
   2. Un nume de funcție
   3. O instrucțiune pentru bucle
   4. Valoarea pe care funcția o returnează către apelator
6. Care este scopul principal al funcțiilor recursive în PHP?
   1. Rezolvarea problemelor matematice
   2. Crearea bucle infinite
   3. Calcularea valorilor constante
   4. Reapelarea funcției în interiorul funcției
7. Ce este un tip de date scalar în PHP?
   1. Un tip de date care acceptă mai multe valori
   2. Un tip de date care acceptă un singur tip de valoare
   3. Un tip de date care reprezintă un șir de caractere
   4. Un tip de date care reprezintă o colecție de date
8. Ce este un namespace în PHP și cum este legat de funcții?
9. Un namespace este un nume de funcție globală
10. Un namespace este un tip de date scalar
11. Un namespace este un container pentru funcții și alte elemente
12. Un namespace nu are legătură cu funcțiile în PHP
13. Cum puteți defini tipurile de date pentru parametrii și valorile returnate în funcțiile PHP?
14. Utilizând cuvântul cheie "type"
15. Folosind prefixe speciale pentru tipuri de date
16. Folosind sintaxa "Tip $parametru: TipRezultat"
17. Nu puteți defini tipuri de date în funcțiile PHP
18. Care dintre următoarele afirmații este adevărată despre funcțiile arrow din PHP?
19. Funcțiile arrow sunt disponibile începând cu PHP 7.0
20. Funcțiile arrow nu pot conține mai multe instrucțiuni
21. Funcțiile arrow sunt utile pentru a crea funcții recursive
22. Funcțiile arrow nu pot avea nume

**Răspunsurile corecte sunt:**

c) Blocuri de cod care pot fi apelate repetat pentru a realiza o anumită acțiune

d) Trebuie să includă întotdeauna un corp de funcție între acoladele {}

c) Prin utilizarea numelui funcției urmat de paranteze ()

c) O variabilă folosită pentru a transmite date către funcție

d) Valoarea pe care funcția o returnează către apelator

a) Rezolvarea problemelor matematice

b) Un tip de date care acceptă un singur tip de valoare

c) Un namespace este un container pentru funcții și alte elemente

c) Folosind sintaxa "Tip $parametru: TipRezultat"

b) Funcțiile arrow nu pot conține mai multe instrucțiuni

## **Probleme practice**

1. Creați o funcție care să primească două numere întregi ca argumente și să returneze suma lor.
2. Creați o funcție care să verifice dacă un șir de caractere conține un anumit cuvânt.
3. Creați o funcție care să inverseze un șir de caractere.
4. Declarați un tablou asociativ cu numele, vârsta și orașul unei persoane și afișați informațiile.
5. Creați o funcție care să sorteze un tablou de numere în ordine descrescătoare.
6. Creați o funcție care să primească un număr și să verifice dacă este par sau impar.

**Problemă 7: Calcularea factorialului**

Scrieți o funcție în PHP care să calculeze factorialul unui număr dat (de exemplu, 5!). Factorialul unui număr este produsul tuturor numerelor de la 1 la numărul respectiv.

**Condiție:** Utilizați o funcție definită de utilizator și asigurați-vă că funcția poate gestiona numere mari pentru a evita depășirea capacitații numerelor.

**Problemă 8: Verificarea palindromelor**

Scrieți o funcție în PHP care să verifice dacă un șir de caractere dat este un palindrom (adică poate fi citit la fel în ambele sensuri, de la stânga la dreapta și de la dreapta la stânga).

**Condiție:** Funcția trebuie să ignore spațiile și să fie insensibilă la majuscule/minuscule.

**Problemă 9: Generarea unui șir acronim**

Scrieți o funcție în PHP care să accepte un șir de cuvinte și să genereze un acronim folosind prima literă din fiecare cuvânt.

**Condiție:** Funcția trebuie să returneze acronimul în litere mari.

**Problemă 10: Suma cifrelor unui număr**

Scrieți o funcție în PHP care să calculeze suma cifrelor unui număr dat. De exemplu, pentru numărul 12345, suma cifrelor este 1 + 2 + 3 + 4 + 5 = 15.

**Condiție:** Utilizați o funcție recursivă pentru a calcula suma cifrelor.

**Problemă 11: Validarea parolei**

Scrieți o funcție în PHP care să valideze o parolă dată conform următoarelor criterii:

* Cel puțin 8 caractere lungime.
* Cel puțin o literă mare, o literă mică și o cifră.
* Nu trebuie să conțină caractere speciale.

**Condiție:** Funcția trebuie să returneze **true** sau **false** în funcție de validitatea parolei.

# **Modulul 4: Lucrul cu directoare și fișiere**

**Unităţi de conţinut**

1. Funcții de lucru cu directoare.
2. Funcții de lucru cu fișiere.
3. Încărcarea fișierelor.

**Abilități**

A15. Obținerea directorului curent.

A16. Modificarea directorului curent de lucru.

A17. Operarea cu directoare (deschiderea, citirea, închiderea, crearea, redenumirea, ștergerea).

A18. Obținerea atributelor fișierului.

A19. Operarea cu fișiere (copierea, redenumirea, ștergerea, deschiderea, închiderea).

A20. Citirea datelor din fișiere.

A21. Afișarea conținutului fișierelor.

A22. Scrierea datelor în fișiere.

A23. Blocarea și obținerea accesului exclusiv la fișiere.

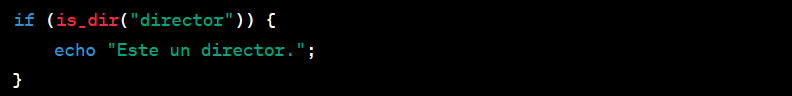
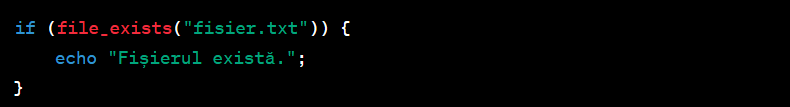
A24. Încărcarea fișierelor pe server.

**Cuvinte cheie:**

1. **mkdir**: Funcția pentru crearea de directoare în PHP.
2. **fopen**: Funcția pentru deschiderea fișierelor în PHP.
3. **fwrite**: Funcția pentru scrierea de date în fișiere în PHP.
4. **move\_uploaded\_file**: Funcția pentru mutarea fișierelor încărcate în PHP.
5. **file\_exists**: Funcția pentru verificarea existenței fișierelor sau directoarelor în PHP.

## **1. Funcții de lucru cu directoare și fișiere**

În PHP, există un set bogat de funcții pentru manipularea directoarelor și fișierelor, ceea ce vă permite să creați, să citiți, să scrieți și să gestionați aceste entități. Iată câteva funcții cheie pentru lucrul cu directoare și fișiere:

* **mkdir()**: Această funcție vă permite să creați un director nou.
* **rmdir()**: Funcția **rmdir()** șterge un director existent. 
* **is\_dir()**: Verifică dacă o cale specificată este un director. 
* **file\_exists()**: Verifică dacă un fișier sau director există. 

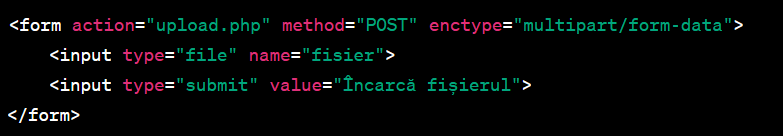
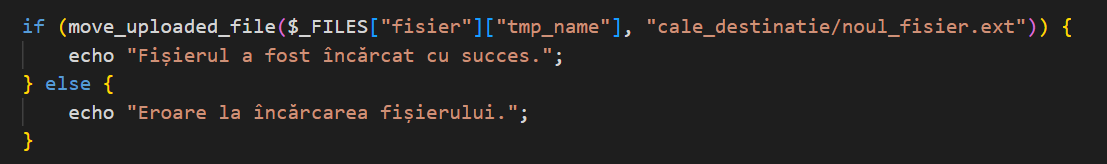
## **2. Funcții de lucru cu fișiere**

Pentru a manipula fișierele, PHP oferă funcții pentru deschiderea, citirea, scrierea și închiderea lor. Câteva funcții comune sunt:

* **fopen()**: Deschide un fișier sau URL în modul specificat. 
* **fclose()**: Închide un fișier deschis anterior. 
* **fwrite()**: Scrie date într-un fișier deschis. 
* **fread()**: Citește date dintr-un fișier deschis. 

## **3. Încărcarea fișierelor**

Încărcarea fișierelor este un aspect esențial în dezvoltarea aplicațiilor web, în special în ceea ce privește manipularea fișierelor încărcate de utilizatori. Pentru a încărca fișiere în PHP, puteți folosi:

* **<input type="file">**: Folosiți un element HTML de tip fișier în formularul dvs. pentru a permite utilizatorilor să selecteze fișierele. 
* **move\_uploaded\_file()**: Funcția **move\_uploaded\_file()** vă permite să mutați un fișier încărcat în directorul dorit. 

Aceste funcții vă permit să gestionați încărcarea și manipularea fișierelor încărcate de utilizatori în aplicațiile dvs. web PHP.

## **Test de evaluare**

1. Ce funcție PHP se utilizează pentru crearea unui director nou? a) **file\_exists()** b) **is\_dir()** c) **mkdir()** d) **rmdir()**
2. Ce face funcția **fclose()** în PHP? a) Deschide un fișier b) Închide un fișier deschis anterior c) Șterge un fișier d) Citește conținutul unui fișier
3. Pentru a deschide un fișier în modul de scriere, ce trebuie să furnizați ca al doilea argument pentru funcția **fopen()**? a) "r" b) "w" c) "a" d) "x"
4. Ce funcție este folosită pentru a muta un fișier încărcat în PHP într-un director specificat? a) **move\_uploaded\_file()** b) **copy()** c) **rename()** d) **write\_file()**
5. Ce face funcția **is\_dir()** în PHP? a) Deschide un fișier b) Verifică dacă un fișier există c) Verifică dacă o cale specificată este un director d) Citește conținutul unui director
6. Pentru a verifica dacă un fișier există, ce funcție PHP se utilizează? a) **is\_file()** b) **file\_exists()** c) **file\_check()** d) **verify\_file()**
7. Care este scopul principal al funcției **move\_uploaded\_file()** în PHP? a) De a deschide un fișier b) De a verifica dacă un fișier există c) De a muta un fișier încărcat într-un director specificat d) De a crea un fișier nou
8. Ce element HTML permite utilizatorilor să selecteze fișiere pentru încărcare într-un formular web? a) **<input type="text">** b) **<input type="submit">** c) **<input type="file">** d) **<input type="checkbox">**
9. Pentru a citi conținutul unui fișier, ce funcție PHP se utilizează? a) **read\_file()** b) **get\_file\_content()** c) **file\_get\_contents()** d) **read\_contents()**
10. Ce funcție PHP se utilizează pentru a verifica dacă un director este gol? a) **empty\_dir()** b) **is\_empty\_directory()** c) **rmdir()** d) **scandir()**

Răspunsurile corecte sunt:

1. c) **mkdir()**
2. b) Închide un fișier deschis anterior
3. b) "w"
4. a) **move\_uploaded\_file()**
5. c) Verifică dacă o cale specificată este un director
6. b) **file\_exists()**
7. c) De a muta un fișier încărcat într-un director specificat
8. c) **<input type="file">**
9. c) **file\_get\_contents()**
10. d) **scandir()**

## **Probleme practice**

**Problemă 1: Copierea unui fișier**

Scrieți un script PHP care să copieze un fișier existent dintr-un director în altul.

**Condiție suplimentară:** Asigurați-vă că scriptul poate gestiona cazurile în care fișierul de destinație există deja.

**Problemă 2: Listarea conținutului unui director**

Scrieți un script PHP care să listeze conținutul unui director (fișierele și subdirectoarele) și să afișeze aceste informații într-un format ușor de citit.

**Condiție suplimentară:** Implementați o funcție pentru a afișa ierarhia de directoare și subdirectoare.

**Problemă 3: Redenumirea unui fișier**

Scrieți un script PHP care să permită utilizatorului să redenumească un fișier existent. Afișați un formular care solicită noul nume al fișierului.

**Condiție suplimentară:** Verificați dacă numele noului fișier respectă regulile (de exemplu, să nu conțină caractere speciale sau spații).

**Problemă 4: Ștergerea unui fișier sau director**

Scrieți un script PHP care să permită utilizatorului să șteargă un fișier sau director selectat.

**Condiție suplimentară:** Adăugați o funcționalitate de confirmare pentru a evita ștergerile accidentale.

**Problemă 5: Încărcarea și descărcarea fișierelor**

Creați o pagină web PHP care să permită utilizatorilor să încarce fișiere pe server și să descarce fișiere existente. Afișați o listă cu fișierele încărcate și oferiți opțiuni pentru descărcare.

**Condiție suplimentară:** Limitați tipurile de fișiere încărcate și verificați dimensiunea fișierelor pentru securitate.

# **Modulul 5: Formulare**

**Unităţi de conţinut**

1. Transmiterea datelor din formular.
2. Metode de transmitere a datelor.
3. Validarea datelor.

**Abilități**

A25. Identificarea variabilelor de transmitere a datelor.

A26. Trimiterea datelor introduse în formulare scriptului de

prelucrare.

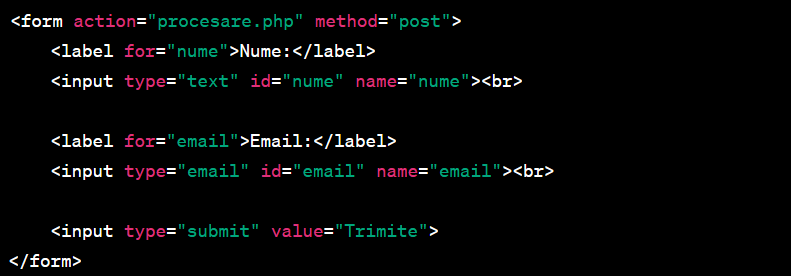
A27. Validarea datelor înainte de utilizarea acestora

**Cuvinte cheie:**

Formularele sunt o parte fundamentală a dezvoltării web, care permit utilizatorilor să trimită date către serverul web și să interacționeze cu aplicațiile online. În PHP, puteți crea și procesa formulare cu ușurință. Iată un ghid despre cum să lucrați cu formulare în PHP:

## **5.1. Crearea un Formular HTML**

Un formular HTML este creat folosind elementul **<form>**, care poate conține diferite tipuri de elemente de introducere, cum ar fi câmpuri text, casete de selecție, butoane radio, butoane de bifare și multe altele. Aici este un exemplu simplu de formular HTML:



**2. Atributul action al Formularului**

Atributul **action** specifică căruta către care sunt trimise datele formularului atunci când acesta este trimis. În exemplul de mai sus, datele vor fi trimise către **procesare.php**.

**3. Atributul method al Formularului**

Atributul **method** specifică metoda HTTP utilizată pentru a trimite datele către server. Cele două valori comune sunt **GET** și **POST**:

* **GET**: Datele formularului sunt incluse în URL și pot fi vizualizate în bara de adrese a browserului. Acesta este potrivit pentru cereri de tip read-only și interogări de căutare.
* **POST**: Datele formularului sunt trimise în corpul cererii HTTP și nu sunt vizibile în URL. Acesta este potrivit pentru trimiterea datelor sensibile sau pentru cereri care modifică starea serverului.

**4. Elementele de Introducere**

În formular, puteți folosi diferite tipuri de elemente de introducere, cum ar fi **<input>**, **<textarea>** sau **<select>**. Fiecare element are un atribut **name** care va fi utilizat pentru a identifica și recupera datele în PHP.

**5. Procesarea Datelor în PHP**

După ce utilizatorul completează formularul și îl trimite, datele sunt procesate într-un script PHP. Puteți accesa datele din formular folosind variabilele superglobale **$\_GET** sau **$\_POST**, în funcție de metoda specificată în formular (**GET** sau **POST**).



**6. Validarea Datelor**

Este important să validați datele primite din formular pentru a preveni introducerea de date incorecte sau malefice. Puteți efectua validarea folosind funcții PHP sau expresii regulate.

**7. Răspunsul către Utilizator**

După procesarea datelor, puteți genera un răspuns și să-l afișați utilizatorului. Acest lucru poate include afișarea unui mesaj de succes sau de eroare sau redirecționarea către o altă pagină.

**8. Securitatea Formularelor**

Asigurați-vă că luați în considerare securitatea formularelor pentru a preveni atacuri precum injecții SQL sau cross-site scripting (XSS). Filtrați și scăpați datele introduse corect, și utilizați funcții precum **htmlspecialchars()** pentru a preveni injectarea de cod HTML sau JavaScript malefic în formulare.

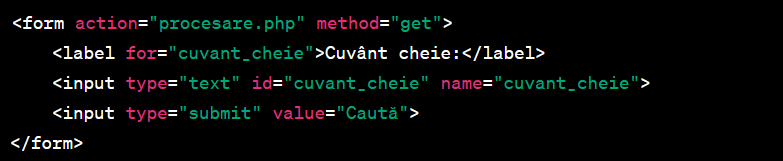
Transmiterea datelor dintr-un formular către server este un proces esențial în dezvoltarea web și permite utilizatorilor să trimită informații către aplicațiile web. În PHP, datele formularului pot fi transmise folosind două metode principale: **GET** și **POST**. Iată detalii despre fiecare dintre ele:

## **5.2. Transmiterea Datelor folosind Metoda GET**

Metoda **GET** este una dintre cele două metode HTTP principale folosite pentru a trimite date către server. Datele sunt atașate la URL și pot fi văzute în bara de adrese a browserului. Acest lucru face metoda **GET** potrivită pentru cereri de tip read-only și pentru cereri de căutare. Iată cum funcționează transmiterea datelor folosind **GET**:

1. **Formular HTML**:

Un formular HTML care utilizează metoda **GET** va arăta astfel:



1. **Trimiterea Datelor**:

Când utilizatorul completează formularul și îl trimite, datele sunt adăugate la URL și trimitere către server în felul următor:



Valoarea introdusă în câmpul de introducere (**cuvant\_cautat** în acest caz) devine parte a query string-ului din URL.

1. **Procesarea Datelor în PHP**:

În PHP, datele pot fi recuperate din **$\_GET**, o variabilă superglobală care conține toate datele **GET** trimise. De exemplu:

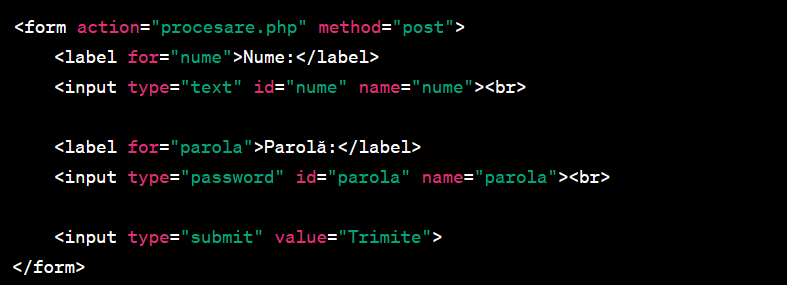


## **5.3 Transmiterea Datelor folosind Metoda POST**

Metoda **POST** este cealaltă metodă HTTP principală pentru transmiterea datelor către server. Datele nu sunt vizibile în URL și nu sunt limitate în dimensiune, ceea ce face ca metoda **POST** să fie potrivită pentru trimiterea datelor sensibile sau pentru cereri care modifică starea serverului. Iată cum funcționează transmiterea datelor folosind **POST**:

1. **Formular HTML**:

Un formular HTML care utilizează metoda **POST** va arăta astfel:

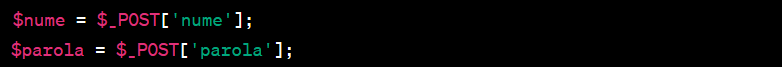


1. **Trimiterea Datelor**:

Când utilizatorul completează formularul și îl trimite, datele sunt trimise în corpul cererii HTTP, și nu sunt vizibile în URL.

1. **Procesarea Datelor în PHP**:

În PHP, datele pot fi recuperate din **$\_POST**, o variabilă superglobală care conține toate datele **POST** trimise. De exemplu:



Ambele metode, **GET** și **POST**, sunt utilizate în dezvoltarea web pentru a transmite date către server, și alegerea unei metode depinde de natura datelor și a cererii. Este important să acordați atenție securității datelor și să validați și să scăpați datele primite corect pentru a preveni vulnerabilități de securitate.

## **5.4 Validarea datelor.**

Validarea datelor este un aspect crucial al dezvoltării web, care implică verificarea și asigurarea că datele introduse într-un formular sau primite de la utilizatori sunt corecte, valide și sigure înainte de a fi procesate sau stocate în baza de date. Validarea datelor este esențială pentru a preveni erori și vulnerabilități de securitate în aplicațiile web. Iată totul despre validarea datelor:

**De Ce Este Necesară Validarea Datelor?**

1. **Prevenirea Erorilor**: Validarea datelor ajută la prevenirea erorilor în aplicații. Aceasta asigură că datele sunt în formatul corect și respectă regulile specificate.
2. **Securitate**: Validarea datelor este un element cheie al securității aplicațiilor web. Prevenind intrările incorecte sau malefice, puteți evita vulnerabilitățile precum injecțiile SQL sau cross-site scripting (XSS).
3. **Integritatea Datelor**: Asigurarea că datele stocate în baza de date sau folosite în aplicații sunt valide contribuie la integritatea datelor.

**Metode de Validare a Datelor**

1. **Validare Frontend vs. Validare Backend**:
   * **Validare Frontend**: Se efectuează în browser, de obicei folosind HTML5 și JavaScript. Acesta poate oferi feedback instant utilizatorilor, dar nu trebuie să fie singura linie de apărare, deoarece poate fi evitată sau înșelată.
   * **Validare Backend**: Se efectuează pe server folosind limbajul de programare, cum ar fi PHP. Este esențială pentru a asigura că datele introduse sunt valide și sigure, indiferent de cum au fost trimise.
2. **Validare Tipului de Date**:
   * Asigurați-vă că datele introduse sunt de tipul corect (de exemplu, numere, șiruri de caractere, adrese de email).
3. **Validare Lungime și Dimensiune**:
   * Verificați dacă datele respectă lungimea și dimensiunea permisă (de exemplu, parole de cel puțin 8 caractere).
4. **Validare Format**:
   * Verificați dacă datele respectă un format specific (de exemplu, un număr de telefon cu o anumită structură).
5. **Validare Integritate**:
   * Asigurați-vă că datele introduse respectă regulile și constrângerile definite pentru domeniul specific (de exemplu, o dată de naștere validă sau un cod poștal valid).
6. **Validare Împotriva Atacurilor**:
   * Verificați și scăpați datele pentru a preveni injecțiile de cod (SQL injection, XSS) și alte atacuri cibernetice.

**Exemple de Funcții și Instrumente de Validare în PHP**

1. **Funcția filter\_var()**: Folosită pentru validarea datelor, cum ar fi adresele de email și URL-uri. De asemenea, poate fi folosită pentru filtrarea datelor pentru a le face sigure.
2. **Funcția preg\_match()**: Folosită pentru a verifica dacă un șir respectă un anumit model sau expresie regulată.
3. **Biblioteci de Validare**: Există biblioteci specializate pentru validarea datelor, cum ar fi "Respect\Validation" sau "Symfony Validator", care pot face procesul de validare mai ușor și mai eficient.
4. **Validarea Bază de Date**: Bazele de date pot avea constrângeri de validare pentru a se asigura că datele introduse sunt corecte. De asemenea, se poate efectua validarea înainte de a insera sau actualiza date în baza de date.

**Fluxul de Lucru Recomandat**

1. **Validare Frontend Preliminară**: Folosiți validarea frontend pentru a oferi feedback instant utilizatorilor și pentru a îmbunătăți experiența utilizatorului.
2. **Validare Backend Detaliată**: Efectuați validarea completă pe server pentru a asigura că datele sunt valide și sigure.
3. **Scăpare și Filtrare a Datelor**: Înainte de a procesa sau stoca datele, asigurați-vă că acestea sunt corecte și sigure prin scăpare și filtrare.
4. **Tratarea Excepțiilor**: Gestionarea erorilor sau a datelor invalide este la fel de importantă ca și validarea în sine. Asigurați-vă că aplicația poate trata erori și excepții corespunzător.

În final, validarea datelor este un aspect fundamental al dezvoltării web responsabile și securizate. Prin aplicarea corectă a validării, puteți îmbunătăți siguranța și integritatea datelor în aplicațiile web.

## **Test de evaluare**

## **Probleme practice**

1. **Formular de Înregistrare**: Creați un formular de înregistrare cu câmpuri pentru nume, adresă de email și parolă. Procesați datele introduse și stocați-le într-un fișier text.
2. **Formular pentru Adăugarea de Notițe**: Dezvoltați un formular pentru adăugarea notițelor. Includeți câmpuri pentru titlu și conținut. Salvati notițele într-un fișier text.
3. **Formular de Contact**: Creați un formular de contact prin care utilizatorii pot trimite întrebări sau mesaje către administrator. Salvati mesajele într-un fișier text și afișați-le administratorului.
4. **Generare de Pagini HTML**: Dezvoltați un formular care permite utilizatorilor să genereze pagini HTML simple. Permiteți-le să introducă titlul, conținutul și stilul paginii și să generați o pagină HTML cu datele introduse.
5. **Încărcarea Fișierelor**: Implementați un formular pentru încărcarea fișierelor. Permiteți utilizatorilor să încarce imagini sau documente, să le procesați și să le stocați temporar pe server.

# **Modulul 6: POO**

**Unităţi de conţinut**

**Abilități**

Înțelegerea conceptelor avansate ale OOP în PHP.

Utilizarea claselor, obiectelor, moștenirii și polimorfismului.

Aplicarea conceptelor de incapsulare și abstractizare.

Lucrul cu metode și proprietăți statice.

Implementarea interfețelor și a claselor abstracte.

**Cuvinte cheie:**

**Introducere în Programarea Orientată pe Obiecte (OOP)**

Programarea orientată pe obiecte (OOP) este un stil de programare care folosește obiecte pentru a modela entitățile din lumea reală și interacțiunile dintre acestea. În PHP, OOP permite dezvoltatorilor să organizeze codul într-o manieră modulară, reutilizabilă și mai ușor de întreținut. Acesta devine un instrument esențial în dezvoltarea aplicațiilor web complexe.

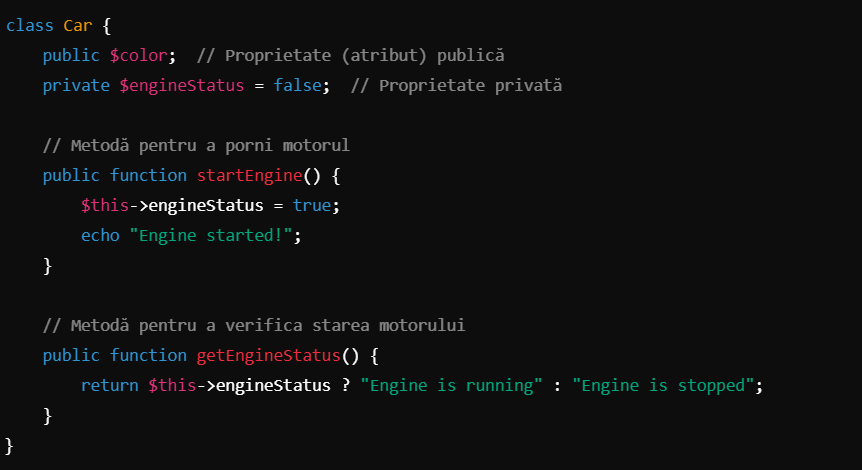
**Obiectele** și **clasele** sunt elementele fundamentale ale programării orientate pe obiecte (OOP). O clasă definește structura și comportamentul obiectelor, iar un **obiect** este o instanță a unei clase. În PHP, OOP oferă o modalitate eficientă și flexibilă de a organiza codul, facilitând reutilizarea, întreținerea și extinderea aplicațiilor.

**1. Clase în PHP**

O **clasă** este o structură care conține date (proprietăți) și comportamente (metode) asociate cu un anumit tip de obiect. Este, de fapt, un șablon sau o „matriță” pentru crearea obiectelor.

**Definirea unei clase**

O clasă este definită cu ajutorul cuvântului cheie class, urmat de numele clasei. În interiorul clasei, se pot defini **proprietăți** (variabile) și **metode** (funcții).

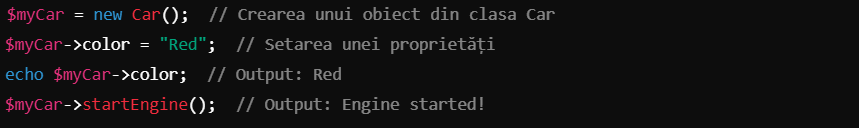


**2. Obiecte în PHP**

Un **obiect** este o instanță a unei clase. Pentru a crea un obiect, se folosește cuvântul cheie new urmat de numele clasei. Obiectul poate accesa metodele și proprietățile definite în clasa respectivă.

**Crearea unui obiect**

După ce o clasă a fost definită, putem crea obiecte din acea clasă.

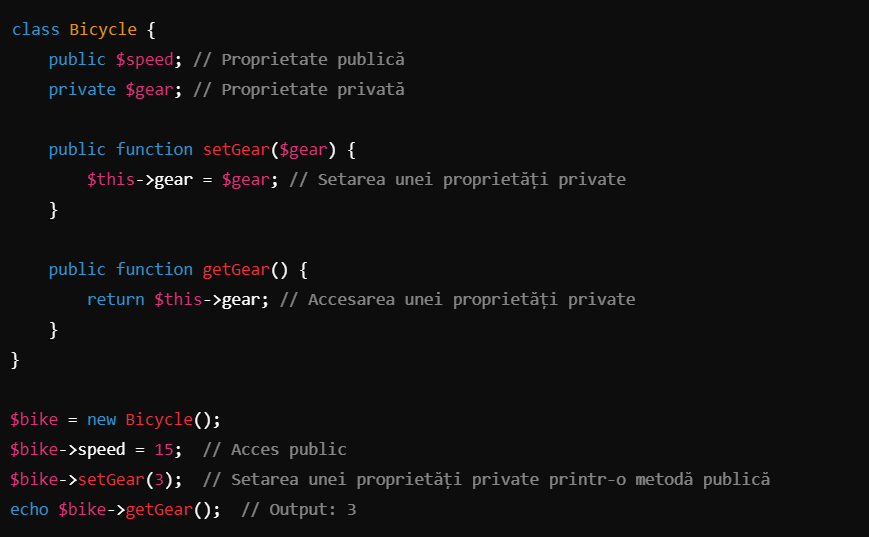


**3. Proprietăți și Metode**

**Proprietăți**

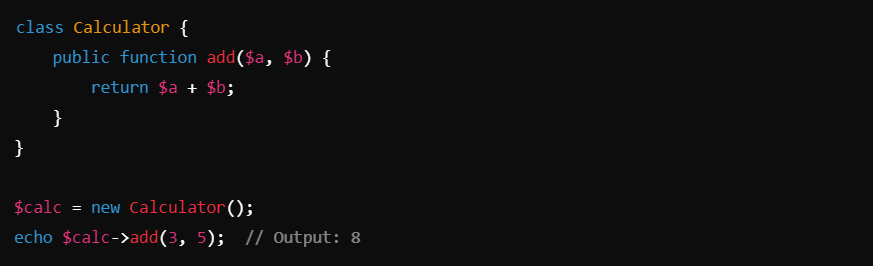
Proprietățile sunt variabile care aparțin unei clase și definesc caracteristicile pe care obiectele din acea clasă le vor avea. Ele pot avea diverse nivele de vizibilitate: **public**, **private** și **protected**.

* public – Proprietățile și metodele declarate public pot fi accesate de oriunde.
* private – Proprietățile și metodele declarate private pot fi accesate doar din interiorul clasei în care sunt definite.
* protected – Pot fi accesate doar din cadrul clasei sau al claselor derivate.



**Metode**

Metodele sunt funcții definite într-o clasă care definesc comportamentul obiectelor.



**4. Constructorul și Destructorul**

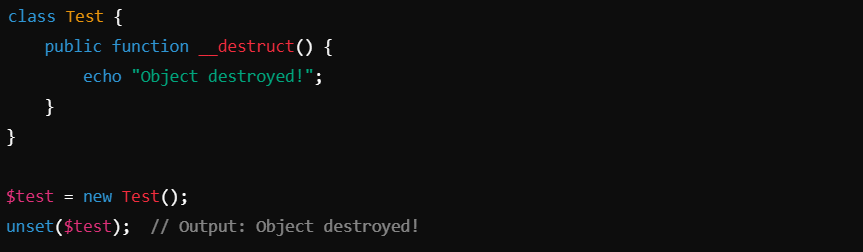
**Constructor**

Un **constructor** este o metodă specială care este apelată automat atunci când un obiect este creat. În PHP, constructorul este definit folosind metoda \_\_construct().



**Destructor**

Un **destructor** este o metodă specială care este apelată atunci când un obiect este distrus sau când scriptul în care este folosit se încheie. Destructorul este definit folosind metoda \_\_destruct().



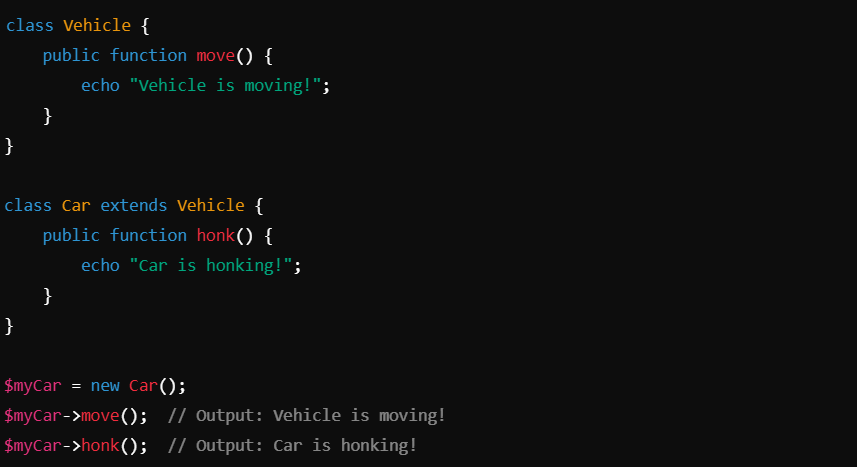
**1. Conceptul de Moștenire**

În PHP, moștenirea se realizează prin intermediul cuvântului cheie extends. O clasă copil poate extinde o singură clasă părinte (PHP nu permite moștenirea multiplă, însă acest lucru poate fi simulat prin utilizarea *traits*).

Prin moștenire, clasa copil poate:

* **Accesa și reutiliza** metodele și proprietățile clasei părinte.
* **Suprascrie** metodele clasei părinte pentru a adăuga sau modifica funcționalități.
* **Adăuga** metode și proprietăți noi care nu sunt prezente în clasa părinte.

**2. Definirea moștenirii în PHP**



**Accesarea metodelor și proprietăților clasei părinte**

Proprietățile și metodele din clasa părinte pot fi accesate direct din clasa copil dacă sunt declarate ca **public** sau **protected**.

* **Public**: Accesibile oriunde, inclusiv în instanțele de obiect.
* **Protected**: Accesibile doar din clasa în care sunt definite și din clasele copil.
* **Private**: Nu sunt accesibile din clasele copil. Pentru acces la datele private, se utilizează **metode getter** sau **setter**.



*Explicație*: Dog moștenește Animal. Proprietatea $name este **protected**, deci accesibilă în clasa copil.

**Suprascrierea Metodelor (Method Overriding)**

**Suprascrierea** este procesul prin care o metodă din clasa copil are aceeași semnătură (nume și parametri) ca o metodă din clasa părinte, dar oferă o implementare diferită. Pentru a apela metoda originală din clasa părinte în timpul suprascrierii, se folosește cuvântul cheie parent::.



*Explicație*: În clasa Penguin, metoda fly() este suprascrisă, oferind o implementare proprie, diferită de cea din clasa Bird.

**Apelarea metodei din clasa părinte**



*Explicație*: Clasa Eagle suprascrie metoda fly(), dar și apelează metoda din clasa părinte folosind parent::fly().

**Constructorul și Moștenirea**

Constructorii pot fi moșteniți și suprascriși în clasele copil. Dacă o clasă copil are un constructor propriu, acesta va suprascrie constructorul clasei părinte. Cu toate acestea, constructorul clasei părinte poate fi apelat explicit cu parent::\_\_construct().



*Explicație*: Constructorul din Employee apelează constructorul din Person pentru a seta numele, iar apoi inițializează titlul jobului specific angajatului.

**Moștenirea multiplă (Simulată cu Traits)**

PHP nu permite moștenirea multiplă directă (o clasă să extindă două clase simultan). Totuși, acest comportament poate fi simulat prin utilizarea **traits**.



*Explicație*: Clasa Duck folosește două traits, CanFly și CanSwim, pentru a obține comportamentul specific fiecăruia.

**Tipuri de moștenire**

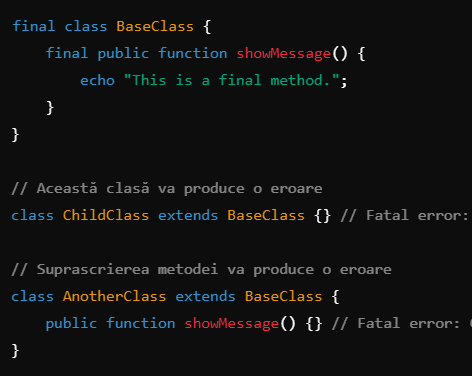
* **Moștenirea simplă**: O clasă copil moștenește o singură clasă părinte.
* **Moștenirea ierarhică**: Mai multe clase copil moștenesc aceeași clasă părinte.
* **Moștenirea pe mai multe nivele**: O clasă copil moștenește o clasă părinte, care la rândul său moștenește o altă clasă.

**Moștenirea pe mai multe nivele:**



**8. Restricționarea moștenirii cu final**

Cu ajutorul cuvântului cheie final, poți preveni moștenirea unei clase sau suprascrierea unei metode. O clasă sau o metodă marcată cu final nu poate fi extinsă sau suprascrisă.



*Explicație*: Metodele și clasele marcate cu final nu pot fi extinse sau modificate în clase derivate.

**Rezumat și Puncte Cheie:**

* **Moștenirea** permite reutilizarea codului, clasele copil moștenind proprietăți și metode din clasele părinte.
* **Suprascrierea** permite clasei copil să redefinească metode din clasa părinte pentru a personaliza comportamentul.
* **parent::** este folosit pentru a accesa metodele și constructorul clasei părinte din clasa copil.
* **Moștenirea multiplă** nu este permisă direct în PHP, dar poate fi simulată cu **traits**.
* Clasa sau metoda **final** împiedică moștenirea și suprascrierea lor.

**Incapsularea în PHP (Encapsulation)**

**Incapsularea** este unul dintre cele mai importante principii ale programării orientate pe obiecte (OOP). În esență, incapsularea constă în **ascunderea detaliilor interne** ale unei clase și **protejarea datelor** de accesul neautorizat din afara clasei. În acest fel, o clasă își gestionează propriile date (proprietăți) și expune doar acele funcționalități (metode) pe care dorește să le ofere altor părți ale programului.

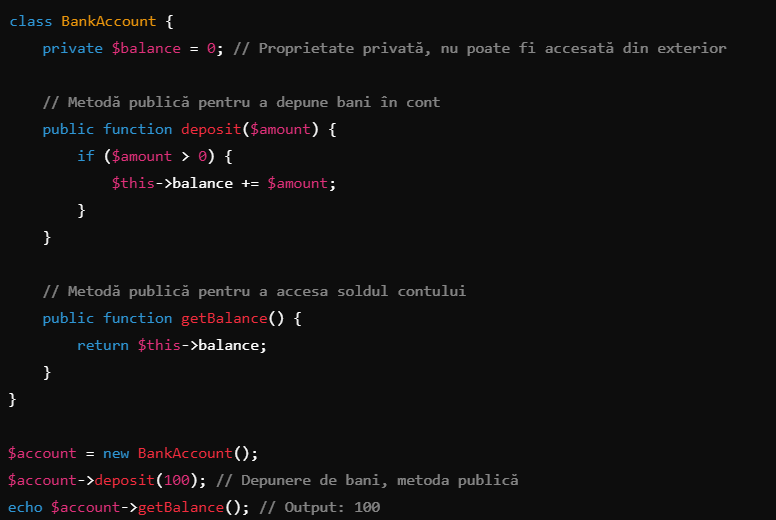
**Modificatorii de acces în PHP**

În PHP, incapsularea este realizată cu ajutorul **modificatorilor de acces**. Aceștia determină nivelul de acces la proprietățile și metodele unei clase. PHP oferă trei modificatori de acces principali:

* **public**: Proprietățile și metodele declarate publice pot fi accesate din orice parte a programului.
* **private**: Proprietățile și metodele declarate private pot fi accesate **doar din interiorul clasei** în care sunt definite.
* **protected**: Proprietățile și metodele declarate protected pot fi accesate doar din interiorul clasei în care sunt definite și din clasele derivate (moștenite).

**2. Utilizarea modificatorilor de acces**

**Exemplo de incapsulare:**



*Explicație*: În acest exemplu, proprietatea $balance este **privată**, ceea ce înseamnă că nu poate fi accesată sau modificată direct din exteriorul clasei. Accesul la această proprietate se face prin intermediul metodelor publice deposit() și getBalance().

**Getter și Setter**

În programarea orientată pe obiecte, accesul la proprietățile private se face prin intermediul **metodelor getter și setter**. Aceste metode permit citirea și modificarea proprietăților fără a expune detaliile interne ale clasei.

**Getter (obținerea valorii):**

Un **getter** este o metodă publică folosită pentru a obține valoarea unei proprietăți private sau protejate.

**Setter (setarea valorii):**

Un **setter** este o metodă publică folosită pentru a modifica valoarea unei proprietăți private sau protejate, oferind astfel un control suplimentar asupra modului în care datele sunt modificate.



*Explicație*: În acest exemplu, metoda setName() verifică dacă numele nu este gol înainte de a-l seta, asigurând validarea datelor. Accesul la numele utilizatorului se face prin metoda getName().

**Incapsularea în relație cu moștenirea**

În moștenire, proprietățile private nu pot fi accesate direct de către clasele copil. Totuși, proprietățile **protected** sunt accesibile în clasele derivate. Acest lucru permite păstrarea unui anumit nivel de control asupra datelor atunci când se extinde funcționalitatea unei clase.



**Polimorfism în PHP (Polymorphism)**

**Polimorfismul** este unul dintre cele patru principii fundamentale ale programării orientate pe obiecte (OOP), alături de moștenire, incapsulare și abstractizare. Termenul provine din grecescul "poly" (multe) și "morphos" (forme), și descrie capacitatea obiectelor de a fi tratate ca instanțe ale clasei lor de bază, dar de a se comporta în moduri diferite în funcție de implementarea concretă a metodelor.

În PHP, **polimorfismul** permite ca o metodă să fie definită în mai multe clase, dar cu implementări diferite, permițând utilizarea aceluiași nume de metodă pentru comportamente diferite. Acest lucru oferă flexibilitate și modularitate în dezvoltarea aplicațiilor, făcându-le mai ușor de extins și întreținut.

**1. Conceptul de Polimorfism**

În polimorfism, metodele cu același nume pot funcționa diferit în clasele derivate, oferind un comportament specific fiecărei clase. Polimorfismul poate fi realizat în două moduri principale în PHP:

* **Polimorfism la nivel de moștenire (overriding)**: Clasele derivate suprascriu metodele clasei părinte.
* **Polimorfism la nivel de interfețe**: Clasele implementează aceeași interfață, dar oferă implementări diferite pentru metodele din interfață.

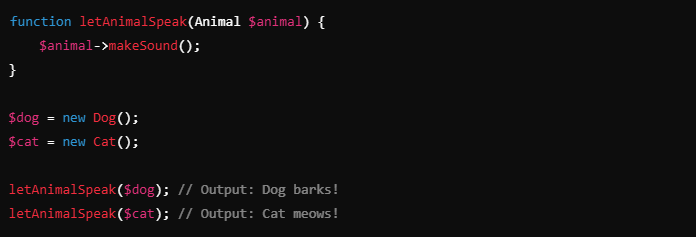
**2. Polimorfism prin Moștenire (Method Overriding)**

Polimorfismul prin moștenire apare atunci când o clasă derivată (copil) suprascrie o metodă din clasa părinte. Deși numele metodei este același, comportamentul este personalizat pentru fiecare clasă derivată.



*Explicație*: Metoda makeSound() este definită în clasa părinte Animal și este suprascrisă în clasele Dog și Cat pentru a oferi comportamente specifice fiecărui tip de animal.

**Utilizarea polimorfismului cu clase moștenite:**



*Explicație*: Funcția letAnimalSpeak() poate accepta orice obiect care extinde clasa Animal și va apela metoda makeSound(), afișând rezultatul specific fiecărui tip de animal.

**Polimorfism cu Interfețe**

Polimorfismul poate fi realizat și prin **interfețe**. În acest caz, mai multe clase implementează aceeași interfață, dar fiecare clasă oferă propria sa implementare pentru metodele din interfață. Acest lucru oferă o flexibilitate sporită, deoarece clasele care implementează interfața nu trebuie să aibă o relație ierarhică (de moștenire) între ele.

**Polimorfism și Tipare de Design**

Polimorfismul joacă un rol central în multe **tipare de design** (design patterns), cum ar fi **Factory Pattern**, **Strategy Pattern** și **Command Pattern**, deoarece aceste tipare se bazează pe faptul că diferite clase pot implementa aceleași metode în moduri diferite.

**Exemplu de Polimorfism într-un Pattern de Strategie:**

*Explicație*: În acest exemplu, PaymentProcessor poate accepta orice metodă de plată care implementează interfața PaymentMethod. Fie că plata se face cu cardul de credit sau prin PayPal, procesul de plată rămâne același din punct de vedere al PaymentProcessor, demonstrând polimorfismul prin interfețe.

**Beneficiile Polimorfismului**

Polimorfismul aduce numeroase avantaje în programarea orientată pe obiecte, printre care:

* **Extensibilitate**: Polimorfismul face ca adăugarea de noi clase sau comportamente să fie simplă, fără a modifica codul existent.
* **Reutilizarea codului**: Funcțiile și metodele pot manipula obiecte din clase diferite într-un mod generic, permițând reutilizarea codului.
* **Simplificarea codului**: Permite scrierea de cod mai curat și mai simplu prin utilizarea de metode comune pentru obiecte de diferite tipuri.
* **Îmbunătățirea organizării**: Polimorfismul contribuie la separarea clară a funcționalităților între obiecte, ceea ce duce la un design mai organizat al codului.

**Sarcini Practice:**

Creează o interfață Notification cu o metodă send(). Implementează două clase: EmailNotification și SMSNotification, care vor trimite mesaje prin e-mail și SMS. Creează o funcție care să poată trimite notificări folosind orice metodă de notificare.

**Abstractizarea în PHP (Abstraction)**

**Abstractizarea** este un concept fundamental în programarea orientată pe obiecte (OOP), care constă în definirea comportamentului comun al mai multor entități fără a furniza detalii complete despre implementarea acestuia. În PHP, abstractizarea este implementată prin **clase abstracte** și **metode abstracte**.

Pe scurt, **abstractizarea** implică crearea de structuri generale (abstracte) care nu pot fi instanțiate direct, ci doar extinse de alte clase concrete care oferă implementări specifice. Acest concept ajută la organizarea mai eficientă a codului, asigurând că clasele derivate implementează comportamentele necesare.

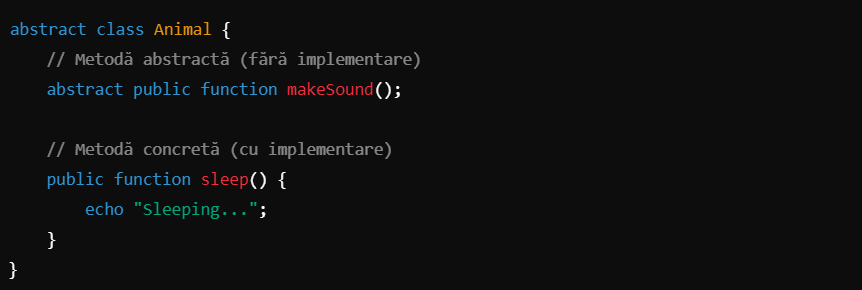
**1. Clase Abstracte în PHP**

O **clasă abstractă** este o clasă care nu poate fi instanțiată direct. În schimb, alte clase trebuie să o extindă și să implementeze metodele sale abstracte. O clasă abstractă poate conține atât **metode abstracte** (fără corp, doar semnătura metodei) cât și **metode concrete** (cu implementare completă).

**Definirea unei clase abstracte**

O clasă abstractă este definită cu cuvântul cheie abstract înainte de cuvântul cheie class.

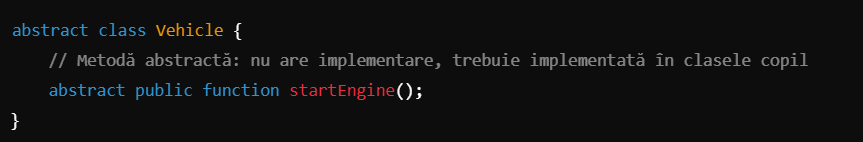
*Explicație*: Clasa Animal este abstractă și conține o metodă abstractă makeSound() (care va trebui implementată în clasele derivate) și o metodă concretă sleep().



**2. Metode Abstracte**

O **metodă abstractă** este o metodă declarată într-o clasă abstractă, dar care nu are implementare. Metodele abstracte forțează clasele derivate să ofere propria implementare.

**Definirea unei metode abstracte:**



*Explicație*: Metoda startEngine() nu are implementare în clasa Vehicle. Orice clasă care extinde Vehicle va trebui să implementeze această metodă.

**3. Moștenirea și implementarea claselor abstracte**

Clasele abstracte sunt utilizate pentru a oferi o structură de bază pe care clasele derivate trebuie să o urmeze. Clasele derivate (copil) sunt obligate să implementeze toate metodele abstracte din clasa părinte abstractă.

**Exemplu de clasă abstractă și implementarea ei:**

*Explicație*: Clasa abstractă Animal definește structura pentru metoda abstractă makeSound() și oferă o metodă concretă move(). Clasele Dog și Cat extind Animal și implementează makeSound() cu comportamente diferite. Acesta este un exemplu de **abstractizare**, deoarece detaliile specifice ale sunetului sunt ascunse în clasa abstractă și oferite doar de clasele derivate.

**Beneficiile abstractizării**

* **Forțează implementarea**: Metodele abstracte impun clasele copil să implementeze anumite funcționalități, asigurând consistența în ierarhia de clase.
* **Reutilizarea codului**: Clasele abstracte pot oferi metode concrete pe care toate clasele derivate le pot reutiliza, reducând dublarea codului.
* **Simplifică structura**: Abstractizarea oferă o modalitate clară de a organiza codul și de a separa conceptele generale de cele specifice.
* **Separarea preocupărilor**: Abstractizarea permite definirea unui cadru general (de exemplu, un concept de "vehicul" sau "animal"), lăsând detaliile specifice implementării pentru clasele derivate.

**Diferențe între Clase Abstracte și Interfețe**

Deși **clasele abstracte** și **interfețele** sunt similare în sensul că ambele pot defini metode pe care alte clase trebuie să le implementeze, există câteva diferențe esențiale:

* **Clasa abstractă** poate avea atât metode abstracte, cât și metode concrete (cu implementare). Interfețele nu pot avea metode concrete (înainte de PHP 8), ci doar metode abstracte (declarate fără implementare).
* **Clasa abstractă** poate conține proprietăți și poate avea un constructor. Interfețele nu pot defini proprietăți sau un constructor.
* **O clasă** poate extinde **o singură clasă abstractă**, dar poate implementa **mai multe interfețe**.

**Clase finale și clase abstracte**

Clasele abstracte pot fi combinate cu conceptul de **clasă finală**. O **clasă finală** nu poate fi extinsă. Este posibil să ai o clasă abstractă în lanțul de moștenire, dar să faci clasa finală pentru a preveni extinderea suplimentară.

**Rezumat și puncte cheie:**

* **Abstractizarea** permite crearea de clase care nu pot fi instanțiate direct, ci doar extinse.
* **Clasele abstracte** pot defini metode abstracte, care trebuie implementate în clasele derivate, și metode concrete, care pot fi reutilizate.
* Abstractizarea separă comportamentele generale de cele specifice, facilitând reutilizarea și extinderea codului.
* **Interfețele** și **clasele abstracte** sunt similare, dar interfețele oferă un nivel mai ridicat de abstractizare, fără a avea implementări concrete sau proprietăți.

**Traits în PHP**

**Traits** sunt un mecanism special în PHP care permite reutilizarea metodelor în mai multe clase, fără a fi nevoie de moștenire. Acestea au fost introduse în PHP 5.4 pentru a rezolva limitarea moștenirii simple din PHP, care permite unei clase să moștenească doar de la o singură clasă părinte. **Traits** permit partajarea metodelor comune între clase care nu au o relație ierarhică directă, evitând astfel duplicarea codului.

**1. Conceptul de Traits**

Un **trait** este similar cu o clasă, dar nu poate fi instanțiat direct. Un trait este menit să fie inclus într-o clasă și să ofere acelei clase funcționalitățile definite în trait. O clasă poate **folosi unul sau mai multe traits**, combinându-le cu metodele sale proprii.

**2. Definirea și utilizarea Traits**

Un trait se definește cu cuvântul cheie trait, urmat de numele trait-ului și apoi corpul acestuia, care conține metodele și proprietățile ce urmează să fie reutilizate în alte clase.

**Proprietăți în Traits**

La fel ca metodele, și **proprietățile** pot fi definite într-un trait. Clasele care folosesc acel trait vor moșteni acele proprietăți. Totuși, trebuie să fii atent la coliziuni, deoarece proprietățile cu același nume în clasă și în trait vor genera erori.

**Exemplu de proprietăți în Traits:**



**Avantajele utilizării Traits**

* **Reutilizarea codului**: Traits permit reutilizarea metodelor între clase diferite, fără a introduce o relație de moștenire între ele.
* **Evitarea duplicării codului**: În loc să copiem aceleași metode în mai multe clase, putem crea un trait și îl putem folosi oriunde este necesar.
* **Flexibilitate**: Traits oferă flexibilitate prin combinarea mai multor seturi de funcționalități într-o singură clasă, fără a se lega de o ierarhie strictă de moștenire.
* **Soluționarea problemelor legate de moștenirea multiplă**: PHP nu permite moștenirea multiplă (o clasă nu poate moșteni de la două clase simultan), dar traits oferă o modalitate de a imita acest comportament, permițând unei clase să "moștenească" funcționalități de la mai multe surse.