Le funzioni

Le Funzioni in PHP

Le **funzioni** in PHP sono blocchi di codice che possono essere riutilizzati per svolgere operazioni specifiche. Le funzioni possono accettare **parametri di input** e possono restituire un **valore di output**. L'utilizzo delle funzioni permette di rendere il codice più modulare, riutilizzabile e leggibile.

1. Dichiarare una Funzione

In PHP, una funzione viene definita utilizzando la parola chiave `function`, seguita dal nome della funzione e dalle parentesi che possono contenere i parametri.

Esempio di dichiarazione base

```php
<?php
function saluta() {
    echo "Ciao, mondo!";
}

saluta();  // Output: Ciao, mondo!
?>
```

Spiegazione:

- `function saluta() { ... }`: definisce una funzione chiamata `saluta`.
- `saluta();`: richiama la funzione per eseguire il codice all'interno di essa.

2. Parametri delle Funzioni

Le funzioni possono accettare dei **parametri** che rappresentano valori di input che la funzione utilizzerà per eseguire le operazioni.

Esempio con parametri

```php
<?php
function salutaUtente($nome) {
    echo "Ciao, $nome!";
}

salutaUtente("Alice");  // Output: Ciao, Alice!
?>
```

Spiegazione:

- La funzione `salutaUtente($nome)` accetta un parametro `$nome`, che viene utilizzato all'interno del messaggio di saluto.
- Quando la funzione viene chiamata, il valore "Alice" viene passato come argomento.

3. Valori di Default per i Parametri

Puoi assegnare un **valore predefinito** ai parametri, nel caso in cui la funzione venga chiamata senza passare argomenti.

Esempio con valore di default

```php
<?php
function salutaUtente($nome = "Ospite") {
    echo "Ciao, $nome!";
}

salutaUtente();  // Output: Ciao, Ospite!
?>
```

Spiegazione:

- Se la funzione `salutaUtente()` viene chiamata senza argomenti, il parametro `$nome` assumerà il valore di default "Ospite".

4. Restituire un Valore con `return`

Una funzione può restituire un valore utilizzando la parola chiave **`return`**. Questo valore può essere salvato in una variabile o utilizzato direttamente.

Esempio con `return`

```php
<?php
function somma($a, $b) {
    return $a + $b;
}

$resultato = somma(5, 10);
echo $resultato;  // Output: 15
?>
```

Spiegazione:

- La funzione `somma($a, $b)` somma due numeri e restituisce il risultato con `return`.
- Il valore restituito viene assegnato alla variabile `$resultato` e poi stampato.

5. Funzioni con Parametri Variabili (`...`)

In PHP, è possibile definire una funzione che accetta un numero variabile di parametri utilizzando l'operatore `...` (spread operator).

Esempio con parametri variabili

```php
<?php
function sommaTotale(...$numeri) {
    return array_sum($numeri);
}

echo sommaTotale(1, 2, 3, 4);  // Output: 10
?>
```

Spiegazione:

- La funzione `sommaTotale(...$numeri)` accetta un numero indefinito di argomenti e li tratta come un array.
- Viene usata la funzione `array_sum()` per sommare tutti i numeri passati.

6. Funzioni Anonime e Lambda

In PHP, una **funzione anonima** (o **lambda**) è una funzione che non ha un nome e può essere assegnata a una variabile o passata come argomento ad altre funzioni. Queste funzioni sono particolarmente utili quando hai bisogno di una funzione temporanea o di una funzione da passare a una chiamata come callback.

Ecco un semplice esempio di una lambda in PHP:

```php
// Definire una funzione anonima e assegnarla a una variabile
$somma = function($a, $b) {
    return $a + $b;
};

// Chiamare la funzione tramite la variabile
echo $somma(2, 3);  // Output: 5
```

In questo caso, la funzione anonima prende due parametri `$a` e `$b`, somma i loro valori e restituisce il risultato. La funzione è assegnata alla variabile `$somma`, e successivamente può essere chiamata come se fosse una normale funzione.

Passare una lambda come callback

Un altro caso comune di utilizzo delle funzioni anonime in PHP è passare una lambda come **callback** a funzioni come `array_map()` o `array_filter()`. Ecco un esempio:

```php
// Usare una funzione anonima come callback per array_map
$numeri = [1, 2, 3, 4, 5];

$quadrati = array_map(function($n) {
    return $n * $n;
}, $numeri);

print_r($quadrati);  // Output: Array ( [0] => 1 [1] => 4 [2] => 9 [3] => 16 [4] => 25 )
```

In questo esempio, la funzione anonima viene passata a `array_map()`, che applica la funzione a ogni elemento dell'array `$numeri`, restituendo un nuovo array con i quadrati dei numeri.

arrow function

In PHP 7.4 è stata introdotta una sintassi abbreviata per le funzioni anonime chiamata **arrow function** (o **lambda con** `fn:`). Questa nuova sintassi rende il codice più conciso e automaticamente cattura le variabili dall'ambito esterno senza dover utilizzare la parola chiave `use` . Le **arrow functions** utilizzano la sintassi `fn` invece di `function` .

Ecco un esempio di utilizzo di una **lambda** con `fn` :

```
// Definire una arrow function che somma due numeri
$somma = fn($a, $b) => $a + $b;

// Chiamare la arrow function
echo $somma(2, 3);  // Output: 5
```

Come puoi vedere, la sintassi è molto più compatta rispetto alle funzioni anonime tradizionali. La funzione `fn($a, $b) => $a + $b` è un'**arrow function** che restituisce

Passare una arrow function come callback

Le arrow functions sono particolarmente utili quando si lavora con funzioni che accettano callback, come `array_map()` o `array_filter()`:

```
// Usare una arrow function come callback per array_map
$numeri = [1, 2, 3, 4, 5];

$quadrati = array_map(fn($n) => $n * $n, $numeri);

print_r($quadrati);  // Output: Array ( [0] => 1 [1] => 4 [2] => 9 [3] => 16 [4] => 25 )
```

In questo esempio, la **arrow function** `fn($n) => $n * $n` viene passata a `array_map()`, che applica la funzione a ogni elemento dell'array `$numeri` per ottenere un nuovo array con i quadrati dei numeri.

Le **arrow functions** sono ideali per funzioni semplici e monoespressione, rendendo il codice più pulito e leggibile

7. Funzioni Ricorsive

Le **funzioni ricorsive** sono funzioni che si richiamano all'interno della loro definizione. Sono utili per risolvere problemi che possono essere divisi in sotto-problemi simili.

Esempio di funzione ricorsiva (fattoriale)

```php
<?php
function fattoriale($n) {
    if ($n <= 1) {
        return 1;
    }
    return $n * fattoriale($n - 1);
}

echo fattoriale(5);  // Output: 120
?>
```

Spiegazione:

- La funzione `fattoriale($n)` chiama se stessa finché `$n` non raggiunge 1. Questo permette di calcolare il fattoriale di un numero.

8. Funzioni Predefinite

PHP include una vasta gamma di **funzioni predefinite** per eseguire operazioni comuni, come la manipolazione delle stringhe, la gestione degli array, operazioni matematiche e altro.

Esempio di funzioni predefinite

```php
<?php
echo strtoupper("ciao!");  // Output: CIAO!
echo count([1, 2, 3, 4]);  // Output: 4
?>
```

Spiegazione:

- `strtoupper()` converte una stringa in maiuscolo.
- `count()` restituisce il numero di elementi in un array.

9. Funzioni Come Argomenti (Callback)

In PHP, è possibile passare funzioni come argomenti ad altre funzioni. Questo si chiama **callback**.

Esempio di callback

```php
<?php
function eseguiCallback($callback) {
    echo $callback();
}

eseguiCallback(function() {
    return "Callback eseguito!";
});  // Output: Callback eseguito!
?>
```

Spiegazione:

- La funzione `eseguiCallback()` accetta una funzione come parametro e la esegue.
- Una funzione anonima viene passata come callback.

Conclusione

Le funzioni in PHP sono uno degli strumenti fondamentali per scrivere codice organizzato e riutilizzabile. Attraverso l'uso di parametri, valori di ritorno, funzioni anonime e callback, puoi modularizzare il tuo codice e migliorare la sua manutenibilità.

Le funzioni tipizzate

Funzioni con Indicazione del Tipo in PHP 7+ e 8

A partire da PHP 7, è possibile dichiarare il **tipo dei parametri** e il **tipo di ritorno** di una funzione. Questo permette di specificare che tipo di dati una funzione può accettare come argomenti e quale tipo di dati deve restituire. PHP 8 ha ulteriormente migliorato questa funzionalità introducendo i **tipi unione** e altre funzionalità avanzate.

L'indicazione del tipo garantisce maggiore **sicurezza** e **leggibilità** del codice, poiché aiuta a prevenire errori relativi ai tipi di dati e rende chiaro il comportamento della funzione.

Sintassi base delle funzioni con tipizzazione

Esempio di funzione con tipi di argomenti e tipo di ritorno

```php
<?php
function somma(int $a, int $b): int {
    return $a + $b;
}

echo somma(5, 10);  // Output: 15
?>
```

Spiegazione:

- `int $a` **e** `int $b`: i parametri `$a` e `$b` devono essere di tipo **intero** (`int`).
- `: int`: indica che la funzione deve restituire un valore di tipo **intero** (`int`).

Se un valore passato alla funzione non è del tipo corretto, PHP tenterà di convertirlo, ma se non può essere convertito automaticamente, verrà generato un errore di tipo.

Tipi di Dati Supportati

PHP supporta diversi tipi di dati per la tipizzazione delle funzioni:

- **Tipi scalari**: `int`, `float`, `string`, `bool`
- **Tipi complessi**: `array`, `object`, `callable`, `iterable`
- **Tipi speciali**: `void`, `mixed`, `null`, `false`

Esempio con vari tipi di argomenti

```php
<?php
function descriviOggetto(string $titolo, float $prezzo): string {
    return "L'oggetto $titolo costa $prezzo euro.";
}

echo descriviOggetto("Lampada", 45.99);
// Output: L'oggetto Lampada costa 45.99 euro.
?>
```

Tipi di Ritorno

A partire da PHP 7, è possibile indicare anche il tipo di ritorno della funzione, usando `: <tipo>`. Se la funzione non restituisce il tipo corretto, verrà generato un errore.

Esempio

```php
<?php
function getPrezzo(): float {
    return 99.99;  // Deve restituire un float
}

echo getPrezzo();  // Output: 99.99
?>
```

Tipi Unione (PHP 8)

PHP 8 ha introdotto i **tipi unione**, che permettono di specificare più di un tipo per un parametro o per il valore di ritorno di una funzione. Questo è utile quando una funzione può accettare o restituire più di un tipo di dato.

Esempio di tipo unione

```php
<?php
function sommaNumeri(int|float $a, int|float $b): int|float {
    return $a + $b;
}

echo sommaNumeri(5, 10.5);  // Output: 15.5
?>
```

Spiegazione:

- `int|float`: indica che i parametri `$a` e `$b` possono essere di tipo **int** o **float**. Lo stesso vale per il tipo di ritorno.

Tipi di ritorno `void` e `mixed`

- `void`: indica che la funzione non deve restituire alcun valore.

Esempio di funzione `void`

```php
<?php
function stampaMessaggio(string $messaggio): void {
    echo $messaggio;
}

stampaMessaggio("Ciao, PHP!");  // Output: Ciao, PHP!
?>
```

- `mixed`: indica che la funzione può restituire **qualsiasi tipo** di dato (introdotto in PHP 8).

Esempio con `mixed`

```php
<?php
function somma($a, $b): mixed {
    return $a + $b;
}

echo somma(5, 3);  // Output: 8
echo somma(5.5, 4.5);  // Output: 10
?>
```

Utilizzo dei Tipi di Oggetto

È possibile specificare che un parametro deve essere un'istanza di una determinata classe o di un'interfaccia.

Esempio con tipo oggetto

```php
<?php
class Oggetto {
    public $nome;

    public function __construct($nome) {
        $this->nome = $nome;
    }
}

function descriviOggetto(Oggetto $oggetto): string {
    return "Questo è un oggetto: " . $oggetto->nome;
}

$oggetto = new Oggetto("Statua");
echo descriviOggetto($oggetto);  // Output: Questo è un oggetto: Statua
?>
```

Conclusione

La tipizzazione delle funzioni in PHP aiuta a migliorare la qualità del codice, rendendolo più leggibile, sicuro e meno soggetto a errori. PHP 7 e 8 offrono strumenti potenti come la tipizzazione dei parametri, il tipo di ritorno, i **tipi unione** e l'uso di `void` e `mixed` per situazioni avanzate.