Guida alla programmazione PHP

Pasquale Cerullo

Indice

Prefazione

Questa guida introduce la programmazione web lato server con PHP.

Il testo è rivolto agli studenti del corso di Informatica e affianca alle spiegazioni teoriche esempi di codice ed esercizi.

1 Introduzione a PHP

1.1 Cos'è PHP e come funziona

1.1.1 Definizione e architettura client-server

PHP (*PHP: Hypertext Preprocessor*) è un linguaggio di scripting **server-side** progettato per lo sviluppo di applicazioni web dinamiche.

- **Non richiede una compilazione manuale prima dell'esecuzione**: il motore PHP traduce internamente il sorgente in istruzioni intermedie (*opcode*) e le esegue sul server.
- **Client (browser)**: richiede una risorsa al server tramite una richiesta HTTP.
- **Server web (Apache/Nginx)**: elabora il file `.php` mediante l'interprete PHP.
- **Output**: il server restituisce al client una risposta, spesso HTML, ma anche JSON, immagini, file o altri tipi di contenuto.

Il codice PHP viene utilizzato per generare dinamicamente i documenti HTML che il client riceve e visualizza nel browser.

> **Da ricordare**
>
> Il browser del client riceve unicamente l'output generato dall'elaborazione; il codice sorgente PHP non viene mai mostrato direttamente all'utente.

1.1.2 Ciclo di vita di una richiesta HTTP

Il protocollo **HTTP** (*HyperText Transfer Protocol*) regola la comunicazione tra client e server web. Il funzionamento si basa sul modello **richiesta/risposta**: il client invia una richiesta e il server restituisce una risposta. Il server non contatta mai il client di propria iniziativa.

```
Browser (client)                          Server web (Apache) + PHP
     |                                              |
     |  ---- 1. Richiesta HTTP (GET/POST) ---->     |
     |                                              |  2. Esecuzione dello script PHP
     |  <---- 3. Risposta HTTP (HTML) --------      |
```

```
    |                                                    |
    4. Il browser visualizza la pagina
```

Le fasi del ciclo di vita

1. **Richiesta dell'utente**: l'utente digita un indirizzo (URL) nel browser, clicca su un link oppure invia un form.
2. **Risoluzione dell'indirizzo**: il browser individua il server. In un sito reale il nome di dominio viene tradotto in un indirizzo IP dal servizio DNS; in locale si usa `localhost` (o `127.0.0.1`), che indica il computer stesso.
3. **Connessione**: il browser si collega al server sulla porta 80 (HTTP) oppure 443 (HTTPS, la versione cifrata del protocollo).
4. **Invio della richiesta HTTP**: il browser invia al server il metodo (GET o POST), la risorsa richiesta, alcune informazioni aggiuntive (header) e, nel caso del metodo POST, i dati del form (body).
5. **Elaborazione lato server**: il server web cerca la risorsa richiesta.
   - Se il file è statico (`.html`, `.css`, immagini), lo restituisce così com'è.
   - Se il file è `.php`, lo passa all'interprete PHP, che esegue il codice — eventualmente leggendo dati da un database o da altri file — e produce come risultato un documento HTML.
6. **Invio della risposta HTTP**: il server restituisce al browser uno **status code** (l'esito della richiesta), gli header e il contenuto (body), generalmente HTML.
7. **Visualizzazione**: il browser interpreta l'HTML e mostra la pagina. Se nella pagina sono presenti altre risorse (fogli di stile, immagini, script JavaScript), il browser effettua una nuova richiesta per ciascuna di esse.

Esempio di richiesta e risposta

Richiesta inviata dal browser (le righe successive alla prima sono gli header):

```
GET /sitoweb/index.php?nome=Mario HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
Accept: text/html
```

Risposta restituita dal server:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html>
    <body>
        <p>Ciao Mario</p>
    </body>
</html>
```

La prima riga della richiesta indica il metodo (`GET`), la risorsa richiesta (`/sitoweb/index.php?nome=Mario`) e la versione del protocollo. La prima riga della risposta contiene lo status code. Una riga vuota separa gli header dal body.

Gli status code principali

Lo status code è un numero di tre cifre; la prima cifra indica la categoria dell'esito:

- **2xx** (successo): `200 OK`, la richiesta è stata eseguita correttamente;
- **3xx** (reindirizzamento): `301` e `302`, la risorsa si trova a un altro indirizzo;
- **4xx** (errore del client): `403 Forbidden`, accesso negato; `404 Not Found`, risorsa inesistente;
- **5xx** (errore del server): `500 Internal Server Error`, errore durante l'elaborazione, ad esempio nello script PHP.

PHP e pagine dinamiche

Poiché lo script PHP viene eseguito a **ogni richiesta**, la risposta può cambiare da una richiesta all'altra:

```php
<?php
// orario.php
echo "<p>Sono le " . date("H:i:s") . "</p>";
?>
```

Il browser non riceve mai il codice PHP, ma solo il risultato, ad esempio `<p>Sono le 10:30:15</p>`. Ricaricando la pagina viene effettuata una nuova richiesta e l'orario mostrato sarà diverso.

HTTP è stateless

HTTP è un protocollo **stateless** (senza memoria): ogni richiesta è indipendente dalle precedenti e il server non conserva alcuna informazione sul client. Questo spiega perché le variabili PHP vengono distrutte al termine dello script: per mantenere dati tra una richiesta e l'altra occorrono meccanismi specifici come cookie, sessioni e database.

Osservare le richieste

Per osservare le richieste e le risposte reali basta aprire gli strumenti per sviluppatori del browser (tasto `F12`) e selezionare la scheda **Rete** (*Network*): per ogni risorsa sono visibili metodo, status code, header e contenuto.

1.2 Stack tecnologico

- **Editor di codice**: Visual Studio Code, Notepad++ o equivalenti.
- **XAMPP**: ambiente di sviluppo che comprende Apache, MariaDB, PHP e altri strumenti.

Guida all'installazione e all'utilizzo di XAMPP

1.3 Sintassi base e output

1.3.1 Tag di apertura e istruzioni di output

I file da interpretare devono avere l'estensione `.php`, come ad esempio `index.php`. Ogni blocco di codice PHP deve essere delimitato dai tag `<?php ... ?>`. L'istruzione `echo` è utilizzata per inviare testo o markup HTML in output.

```php
<?php

// Esempio 1: output di testo semplice e markup HTML
echo "<h1>Benvenuti al corso di PHP</h1>";
echo "<p>Primo script eseguito correttamente.</p>";

// Esempio 2: echo con argomenti multipli separati da virgola
echo "Docente: ", "Prof. Cerullo Pasquale", "<br>";

// Esempio 3: echo con variabili
```

```php
$nome = "Pasquale";
echo "Ciao: " . $nome;
?>
```

> **File contenenti solo PHP**
>
> Quando un file contiene esclusivamente codice PHP, è buona pratica omettere il tag finale `?>`: si evitano così spazi o righe vuote inviati accidentalmente al browser.

1.3.2 Per iniziare

- Avviare **Apache** dal pannello di controllo di XAMPP.
- Salvare il file `.php` nella cartella `C:\xampp\htdocs`.
- Aprire nel browser `http://localhost/<nome_file>.php` oppure `http://127.0.0.1/<nome_file>.php`.

> **Progetti in sottocartelle**
>
> Se il progetto è suddiviso in sottocartelle, l'indirizzo segue il percorso del file. Per esempio, al file `C:\xampp\htdocs\sitoweb\index.php` corrisponde l'indirizzo `http://127.0.0.1/sitoweb/index.php`.

1.4 Variabili, tipi di dati e costanti

1.4.1 Gestione delle variabili e tipi primari

Le variabili in PHP sono precedute dal simbolo `$`. Il linguaggio supporta la tipizzazione dinamica: una variabile può contenere valori di tipi differenti durante l'esecuzione. Le variabili dichiarate fuori dalle funzioni appartengono all'ambito globale dello script; quelle dichiarate dentro una funzione sono normalmente locali. Tutte vengono eliminate al termine della richiesta.

```php
<?php

// Variabili di tipi differenti
$nomeStudente = "Marco";  // string
$eta          = 20;       // int
$mediaVoti    = 28.5;     // float
$isIscritto   = true;     // bool: può contenere anche false
```

```php
// Definizione di costanti
define("ANNO_ACCADEMICO", 2026);
const NOME_CORSO = "Programmazione Web Server-Side";

// Interpolazione di variabili, valida nelle stringhe con doppi apici
echo "Lo studente $nomeStudente (età: $eta) ha una media del $mediaVoti.<br>";

// Concatenazione tramite l'operatore punto (.)
echo "Corso: " . NOME_CORSO . " - Anno: " . ANNO_ACCADEMICO;
?>
```

1.4.2 Le stringhe

Una stringa è un insieme di caratteri. Le stringhe possono essere delimitate in due modi:

- con apici singoli;
- con virgolette doppie.

```php
<?php
$frase = 'Anna disse: "Ciao!" ma nessuno rispose';
echo $frase;

$frase2 = "Anna disse: 'Ciao!' ma nessuno rispose";
echo $frase2;

echo 'Torniamo un\'altra volta'; // Torniamo un'altra volta
echo "Torniamo un'altra volta";  // Torniamo un'altra volta

// La riga seguente causerebbe un errore, perché l'apostrofo
// sarebbe interpretato come apice di chiusura:
// echo 'Torniamo un'altra volta';
?>
```

> ⚠ Nomenclatura e distinzione tra maiuscole e minuscole
>
> I nomi delle variabili sono **case-sensitive**: `$nome` e `$NOME` indicano due variabili differenti.

1.4.3 Array

Un array (vettore) è una variabile complessa che contiene una serie di valori, ciascuno caratterizzato da una chiave o indice. Il seguente array contiene cinque valori:

```php
$colori = array('bianco', 'nero', 'giallo', 'verde', 'rosso');
echo $colori[1]; // stampa 'nero'
echo $colori[4]; // stampa 'rosso'
```

Ogni colore è caratterizzato da un indice numerico, che PHP assegna automaticamente a partire da 0. È possibile accedere agli elementi dell'array tramite tale indice.

Esistono più modi per definire un array:

```php
$colori = array('bianco', 'nero', 'giallo', 'verde', 'rosso');
$colori = ['bianco', 'nero', 'giallo', 'verde', 'rosso'];

$colori = array(1 => 'bianco', 'nero', 'giallo', 'verde', 'rosso');
// L'indice iniziale è 1.

$colori = array('primo' => 'bianco', 'secondo' => 'nero');
// Le chiavi sono stringhe associate ai valori.
```

In PHP la chiave di un array può essere un numero intero oppure una stringa. Gli array PHP possono quindi comportarsi sia come liste sia come mappe associative.

1.4.4 Array bidimensionali

Un array bidimensionale è un array che contiene altri array. Osserviamo un esempio:

```php
$persone = array(
    array('nome' => 'Mario', 'cognome' => 'Rossi'),
    array('nome' => 'Paolo', 'cognome' => 'Bianchi'),
    array('nome' => 'Luca', 'cognome' => 'Verdi')
);

echo $persone[0]['cognome']; // stampa 'Rossi'
echo $persone[2]['nome'];    // stampa 'Luca'

$persone = [
    ['nome' => 'Mario', 'cognome' => 'Rossi'],
    ['nome' => 'Paolo', 'cognome' => 'Bianchi'],
    ['nome' => 'Luca', 'cognome' => 'Verdi']
]; // Sintassi equivalente all'esempio precedente.
```

1.4.5 Gli operatori

```
// Assegnazione
$nome = 'Giorgio';
$nome_app = $nome;

// Operatori aritmetici
$a = 3 + 7; // addizione
$b = 5 - 2; // sottrazione
$c = 9 * 6; // moltiplicazione
$d = 8 / 2; // divisione
$e = 7 % 4; // modulo

// Concatenazione delle stringhe
$nome = 'Pippo';
$stringa1 = 'Ciao ' . $nome;

// Operatori di assegnazione combinati
$x += 4; // equivale a $x = $x + 4
$x -= 3; // equivale a $x = $x - 3
$x .= $a; // equivale a $x = $x . $a
$x /= 5; // equivale a $x = $x / 5
$x *= 4; // equivale a $x = $x * 4
$x %= 2; // equivale a $x = $x % 2

// Incremento e decremento
$a++; // usa $a, poi lo incrementa di 1
$a--; // usa $a, poi lo decrementa di 1
++$a; // incrementa $a, poi lo usa
--$a; // decrementa $a, poi lo usa
```

Gli operatori di confronto producono sempre un valore booleano: `true` oppure `false`.

```
==  // valore uguale, anche se il tipo è diverso
!=  // valore diverso
>   // maggiore
>=  // maggiore o uguale
<   // minore
<=  // minore o uguale
=== // valore uguale e stesso tipo
!== // valore diverso oppure tipo diverso
```

Negli esempi moderni sono generalmente preferibili `===` e `!==`, perché confrontano sia il valore sia il tipo ed evitano conversioni implicite inattese.

Gli operatori logici consentono di combinare più espressioni:

- **OR** è falso solamente se entrambi gli operandi sono falsi; si indica con `or` oppure `||`.
- **AND** è vero solamente se entrambi gli operandi sono veri; si indica con `and` oppure `&&`.
- **NOT** è la negazione logica; si indica con `!`.
- **XOR**, detto anche *or esclusivo*, è vero se uno solo dei due operandi è vero; si indica con `xor`.

> ⚠ Precedenza degli operatori logici
>
> `and` e `or` non hanno la stessa precedenza di `&&` e `||`. Per rendere evidente l'ordine di valutazione, negli esempi e nelle condizioni è preferibile usare `&&` e `||` e aggiungere parentesi quando si combinano più espressioni.

1.5 Esercizi

1. Dato un numero, per esempio `$numero = 35`, stamparne il quadrato.
2. Date base e altezza di un rettangolo, calcolare e stampare area e perimetro.
3. Dati due numeri, stampare il resto della divisione e la media aritmetica.
4. Convertire un numero di secondi in ore, minuti e secondi usando gli operatori `/` e `%`.
5. Creare alcune variabili con tipi differenti e osservarne tipo e valore con `var_dump()`.

> ⚠ Istruzioni di consegna
>
> Consegnare ogni esercizio in un file PHP diverso e numerato. Inserire tutti i file in una cartella denominata `cognome_nome` e consegnarla in formato compresso.

2 Strutture di controllo

2.1 `echo` e `print`

`echo` e `print` sono costrutti del linguaggio usati per produrre output. I valori non testuali vengono convertiti in stringhe quando possibile.

- `echo` accetta uno o più argomenti e non restituisce alcun valore;
- `print` accetta un solo argomento e restituisce sempre `1`.

```php
<?php
echo "Ciao mondo!";
print "Ciao mondo!";

echo 123;

$messaggio = "Ciao mondo!";
echo $messaggio;

// echo può ricevere più argomenti separati da virgole.
echo "Nome: ", "Mario", "<br>";
?>
```

Nelle stringhe delimitate da virgolette doppie PHP sostituisce il nome di una variabile con il suo valore. Le parentesi graffe delimitano il nome quando subito dopo sono presenti altri caratteri.

```php
<?php
$numero = 45;
$veicolo = "autobus";
echo "Questo $veicolo ha una capacità di $numero passeggeri";

$memoria = 256;
echo "Il mio computer ha {$memoria} MByte di RAM";
?>
```

> **Interpolazione delle variabili**
>
> L'interpolazione funziona nelle stringhe delimitate da virgolette doppie, non in quelle delimitate da apici singoli.

2.2 `var_dump()` e `print_r()`

Le funzioni `var_dump()` e `print_r()` sono utili durante il debug:

- `var_dump()` mostra tipo e valore;
- `print_r()` mostra il contenuto in forma leggibile, soprattutto per gli array.

```php
<?php
$numero = 39;
var_dump($numero); // int(39)

$colori = ["blu", "rosso"];
print_r($colori);
/*
Array
(
    [0] => blu
    [1] => rosso
)
*/
?>
```

> **Strumenti di sviluppo**
>
> Queste funzioni sono adatte al debug durante lo sviluppo. In un'applicazione pubblicata non dovrebbero mostrare agli utenti informazioni interne o dati sensibili.

2.3 La selezione con `if`

L'istruzione `if` valuta un'espressione booleana. Se la condizione è `true`, esegue il primo blocco; altrimenti può eseguire il blocco `else`.

```php
<?php
if (<condizione booleana>) {
    <istruzioni eseguite se la condizione è true>
} else {
    <istruzioni eseguite se la condizione è false>
}
?>
```

Un esempio completo:

```php
<?php
$eta = 17;

if ($eta >= 18) {
    echo "Maggiorenne";
} else {
    echo "Minorenne";
}
?>
```

Per verificare più casi si utilizza `elseif`:

```php
<?php
$voto = 8;

if ($voto < 6) {
    echo "Insufficiente";
} elseif ($voto < 8) {
    echo "Buono";
} else {
    echo "Ottimo";
}
?>
```

> **Blocchi e annidamento**
>
> Le istruzioni `if`, `elseif` ed `else` possono essere annidate. Le parentesi graffe rendono espliciti i blocchi e vanno usate anche quando contengono una sola istruzione.

2.4 La selezione multipla con `switch`

`switch` confronta il valore di un'espressione con diversi casi. `break` impedisce che l'esecuzione prosegua nei casi successivi.

```php
<?php
$giorno = 2;

switch ($giorno) {
    case 1:
        echo "Lunedì";
        break;
    case 2:
        echo "Martedì";
        break;
    default:
        echo "Valore non valido";
}
?>
```

> **Confronto usato da switch**
>
> `switch` effettua confronti non stretti, simili a `==`. Quando è importante distinguere anche i tipi, può essere preferibile usare `match`, disponibile da PHP 8, oppure una sequenza di confronti con `===`.

2.5 Il ciclo `while`

`while` ripete un blocco finché la condizione rimane vera. Se la condizione è falsa fin dall'inizio, il blocco non viene mai eseguito.

```php
<?php
$numero = 10;

while ($numero > 5) {
    echo $numero . " ";
    $numero--; // Evita un ciclo infinito.
}
?>
```

2.6 Il ciclo `do...while`

`do...while` controlla la condizione dopo il blocco: le istruzioni vengono quindi eseguite almeno una volta.

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

do {
    echo $numero . " ";
    $numero++;
} while ($numero <= 5);
?>
```

2.7 Il ciclo `for`

`for` raccoglie nella sua intestazione l'inizializzazione, la condizione e l'aggiornamento della variabile di controllo.

```php
<?php
for ($i = 1; $i <= 10; $i++) {
    $risultato = 5 * $i;
    echo $risultato . " ";
}
?>
```

I tre elementi vengono valutati in questo ordine:

1. **inizializzazione**, una sola volta all'inizio;
2. **condizione**, prima di ogni iterazione;
3. **aggiornamento**, alla fine di ogni iterazione.

2.8 Il ciclo `foreach`

`foreach` è progettato per elaborare gli array. La forma completa rende disponibili la chiave e il valore dell'elemento corrente:

```php
<?php
$voti = ["Mario" => 8, "Anna" => 9, "Luca" => 7];

foreach ($voti as $nome => $voto) {
    echo "$nome: $voto<br>";
}
?>
```

Se interessa solamente il valore si usa la forma abbreviata:

```php
<?php
$numeri = [10, 20, 30];

foreach ($numeri as $numero) {
    echo $numero . " ";
}
?>
```

2.9 Esercizi

1. Data una variabile numerica, stabilire se contiene un numero positivo, negativo oppure zero.
2. Dato un voto da 1 a 10, stampare `insufficiente`, `sufficiente`, `buono` oppure `ottimo` usando `if`, `elseif` ed `else`.
3. Usare `switch` per stampare il nome del giorno corrispondente a un numero da 1 a 7.
4. Stampare i numeri da 1 a 20 prima con `while`, poi con `do...while` e infine con `for`.
5. Stampare la tabellina di un numero da 1 a 10 usando un ciclo `for`.
6. Dato un array di numeri, calcolare il minimo, il massimo e la media usando `foreach`, senza ricorrere direttamente alle funzioni `min()`, `max()` e `array_sum()`.
7. Dato un array associativo contenente nomi e voti, stampare solamente gli studenti con voto sufficiente.

> **Istruzioni di consegna**
>
> Consegnare ogni esercizio in un file PHP diverso e numerato. Inserire tutti i file in una cartella denominata `cognome_nome` e consegnarla in formato compresso.

3 Le funzioni

3.1 Cos'è una funzione

Una funzione è un insieme di istruzioni che esegue un'operazione specifica. Evita di riscrivere lo stesso codice più volte: è sufficiente **richiamarla**, fornendole gli eventuali **parametri** di cui ha bisogno. Ogni funzione è caratterizzata da:

- un nome;
- i parametri richiesti (quali e quanti);
- il fatto di restituire o meno un valore.

```
nome_funzione(); // nessun parametro, nessun valore restituito
nome_funzione($a, $b); // due parametri
$valore = nome_funzione($a);  // il valore restituito viene salvato in una variabile
```

> **Attenzione**
>
> Le parentesi tonde sono **obbligatorie** anche quando la funzione non richiede parametri.

Le funzioni possono essere **built-in** (incorporate nel linguaggio) oppure **definite dall'utente**.

3.2 Le funzioni built-in

PHP dispone di moltissime funzioni incorporate. Non è necessario impararle a memoria, ma è importante saper consultare il manuale ufficiale php.net: digitando `https://www.php.net/nome_funzione` si accede direttamente alla pagina della funzione.

> **Nota**
>
> Nel manuale ogni funzione è descritta nella forma `tipo_restituito nome(parametri)`. I parametri indicati tra parentesi quadre `[ ]` sono opzionali.

Funzioni sulle variabili

- `isset($var)`: TRUE se la variabile è definita e diversa da NULL;
- `empty($var)`: TRUE se la variabile è vuota (stringa vuota, 0, NULL) oppure non definita;
- `is_null($var)`: TRUE se il valore della variabile è NULL;
- `is_int($var)`, `is_float($var)`, `is_string($var)`, `is_array($var)`: TRUE se la variabile è, rispettivamente, un intero, un decimale, una stringa, un array;
- `is_numeric($var)`: TRUE se la variabile è un numero oppure una stringa che contiene un numero;
- `unset($var)`: distrugge la variabile (è un costrutto del linguaggio, non una vera funzione). Dopo l'`unset`, `isset` restituisce FALSE ed `empty` restituisce TRUE;
- `print_r($var)`: stampa il contenuto di una variabile o di un array (utile per il debug, vedi sezione precedente).

```php
$a = 5;
$b = isset($a);         // TRUE
$b = is_int($a);        // TRUE
$b = is_float($a);      // FALSE

$a = '5';
$b = is_int($a);        // FALSE: ora $a contiene una stringa
$b = is_string($a);     // TRUE
$b = is_numeric($a);    // TRUE: la stringa contiene un numero

unset($a);
$b = isset($a);         // FALSE: variabile distrutta

// utilizzo tipico: condizione di un if
if (empty($a)) {
    echo "Variabile vuota o non definita";
} else {
    echo "La variabile contiene un valore";
    }
```

Funzioni sulle stringhe

- `strlen($str)`: restituisce la lunghezza della stringa in byte. Con testo UTF-8 contenente caratteri accentati, per contare i caratteri si usa `mb_strlen()` se l'estensione `mbstring` è disponibile;
- `trim($str)`, `ltrim($str)`, `rtrim($str)`: eliminano gli spazi rispettivamente all'inizio e alla fine, solo all'inizio, solo alla fine della stringa;
- `ucfirst($str)`: porta in maiuscolo il primo carattere della stringa;
- `ucwords($str)`: porta in maiuscolo il primo carattere di ogni parola;
- `strtolower($str)`, `strtoupper($str)`: convertono tutta la stringa in minuscolo o in maiuscolo;
- `str_replace($cerca, $sostituisci, $str)`: sostituisce nella stringa tutte le occorrenze di `$cerca` con `$sostituisci` (`str_ireplace` ignora la differenza tra maiuscole e minuscole);
- `strpos($str, $cerca)`: restituisce la posizione (a partire da 0) della prima occorrenza di `$cerca`, oppure FALSE se non è presente (`stripos` ignora maiuscole e minuscole);
- `strstr($str, $cerca)`: restituisce la parte di stringa che parte dalla prima occorrenza di `$cerca`, oppure FALSE (`stristr` ignora maiuscole e minuscole);
- `substr($str, $inizio [, $lunghezza])`: restituisce una porzione della stringa.

Il funzionamento di `substr` dipende dai valori dei parametri:

- i caratteri si contano **a partire da 0**;
- se `$inizio` è negativo si parte dalla fine (`-1` è l'ultimo carattere);
- se `$lunghezza` è omessa si arriva fino alla fine della stringa;
- se `$lunghezza` è negativa indica quanti caratteri escludere dalla fine.

```php
$s = "  ciao mondo  ";
echo strlen($s);                  // 14
$s = trim($s);                    // "ciao mondo"
echo ucfirst($s);                 // Ciao mondo
echo ucwords($s);                 // Ciao Mondo
echo strtoupper($s);              // CIAO MONDO
echo str_replace("o", "0", $s);   // cia0 m0nd0
echo strpos($s, "mondo");         // 5
echo strstr("Lorenzo", "re");     // renzo
```

```php
echo substr("Lorenzo", 4);        // nzo
echo substr("Lorenzo", -3, 2);    // nz
echo substr("Lorenzo", 1, -2);    // oren
```

> **Attenzione**
>
> `strpos` restituisce `0` se la stringa cercata si trova all'inizio, ma `0` e `false` sono valori uguali per l'operatore `==`. Per verificare se la ricerca è riuscita si usa l'operatore `!==`:

```php
if (strpos("Lorenzo", "Lo") !== false) {
    echo "Trovato";
}
```

Funzioni sugli array

- `count($array)` (o `sizeof`): restituisce il numero di elementi dell'array;
- `array_reverse($array [, $preserve_keys])`: restituisce l'array con gli elementi in ordine inverso. Con `TRUE` come secondo parametro le chiavi vengono mantenute;
- `sort($array)`, `rsort($array)`: ordinano l'array in modo crescente o decrescente;
- `in_array($valore, $array)`: TRUE se il valore è presente nell'array;
- `array_key_exists($chiave, $array)`: TRUE se la chiave è presente nell'array;
- `array_search($valore, $array)`: restituisce la chiave del valore trovato, oppure FALSE;
- `array_merge($array1, $array2 [, ...])`: unisce due o più array. Le chiavi numeriche vengono rinumerate; per le chiavi associative uguali, l'ultimo valore sovrascrive i precedenti;
- `array_push($array, $valore [, ...])`: aggiunge uno o più valori in fondo all'array;
- `array_pop($array)`: rimuove e restituisce l'ultimo elemento;
- `array_unshift($array, $valore [, ...])`: aggiunge uno o più valori all'inizio dell'array;
- `array_shift($array)`: rimuove e restituisce il primo elemento (gli indici numerici vengono rinumerati);
- `explode($separatore, $str)`: trasforma una stringa in un array, dividendola in base al separatore;

- `implode($separatore, $array)`: trasforma un array in una stringa, inserendo il separatore tra gli elementi.

> **Attenzione**
>
> Le funzioni `sort`, `rsort`, `array_push`, `array_pop`, `array_shift` e `array_unshift` **modificano direttamente l'array** passato come parametro: l'ordine o il contenuto originale va perso. Dopo `sort` e `rsort` le chiavi originali vengono eliminate e sostituite da indici numerici a partire da 0.

```php
$numeri = [4, 9, 1];
echo count($numeri);              // 3
sort($numeri);                    // 1, 4, 9
array_push($numeri, 7);           // 1, 4, 9, 7
$ultimo = array_pop($numeri);     // $ultimo = 7      -> 1, 4, 9
$primo = array_shift($numeri);    // $primo = 1       -> 4, 9
array_unshift($numeri, 0);        // 0, 4, 9

var_dump(in_array(4, $numeri));  // bool(true)
echo array_search(9, $numeri);   // 2

$c = array_merge([1, 2], [3, 4]);         // 1, 2, 3, 4
$parole = explode(" ", "ciao Mario");     // ["ciao", "Mario"]
$testo = implode("-", $parole);           // "ciao-Mario"
```

Come per `strpos()`, il risultato di `array_search()` deve essere confrontato con `false` usando `!==` quando si vuole sapere se il valore è stato trovato.

Funzioni sulle date

Le funzioni sulle date si basano sul **timestamp**: un numero intero che rappresenta i secondi trascorsi dal 1° gennaio 1970.

- `time()`: restituisce il timestamp del momento in cui viene eseguita;
- `date($formato [, $timestamp])`: restituisce una stringa con la data formattata secondo il formato indicato. Se il timestamp è omesso, usa quello attuale;
- `mktime($ora, $minuti, $secondi, $mese, $giorno, $anno)`: calcola il timestamp di una data. Se un valore supera il proprio limite, PHP lo riporta nell'intervallo corretto, ad esempio il mese 14 diventa febbraio dell'anno successivo. Questo permette di eseguire calcoli sulle date;

- `checkdate($mese, $giorno, $anno)`: TRUE se i valori formano una data valida.

I principali codici per il formato di `date`:

Codice	Significato	Codice	Significato
`Y`	anno (4 cifre)	`d`	giorno del mese (01–31)
`y`	anno (2 cifre)	`j`	giorno del mese (1–31)
`m`	mese (01–12)	`w`	giorno della settimana (0 = dom, 6 = sab)
`n`	mese (1–12)	`l`	giorno della settimana testuale
`F`	mese testuale (`January`)	`D`	giorno della settimana su 3 lettere
`M`	mese su 3 lettere (`Jan`)	`H`	ora (00–23)
`i`	minuti (00–59)	`G`	ora (0–23)
`s`	secondi (00–59)		

```php
echo time();                          // timestamp attuale
echo date("d/m/Y");                   // data attuale, es. 24/09/2026
echo date("H:i:s");                   // ora attuale, es. 10:30:15
echo date("d/m/Y", mktime(0, 0, 0, 14, 1, 2026));   // 01/02/2027
var_dump(checkdate(2, 30, 2026));     // bool(false)
```

> **Attenzione**
>
> I parametri di `mktime` hanno un ordine particolare (ora, minuti, secondi, mese, giorno, anno) che porta facilmente a errori.

I timestamp non contengono informazioni sul fuso orario. Per applicazioni reali è preferibile utilizzare `DateTimeImmutable` e impostare esplicitamente il fuso:

```php
$oraRoma = new DateTimeImmutable("now", new DateTimeZone("Europe/Rome"));
echo $oraRoma->format("d/m/Y H:i:s");
```

> **Date moderne**
>
> `DateTimeImmutable` restituisce nuovi oggetti quando si modifica una data, evitando cambiamenti accidentali dell'oggetto originale. È generalmente preferibile per elaborazioni non banali.

3.3 Le funzioni definite dall'utente

Quando una stessa sequenza di istruzioni deve essere ripetuta più volte, è consigliabile definire una funzione propria. I vantaggi sono un codice più leggibile, più compatto e più facile da mantenere. La sintassi è la seguente:

```
function nomeFunzione($parametro1, $parametro2 = valoreDefault) {
    <blocco di istruzioni>
    return <espressione>;   // opzionale
}
```

Le parentesi graffe sono obbligatorie. Le dichiarazioni di tipo dei parametri e del valore restituito sono facoltative, ma rendono il contratto della funzione più chiaro e permettono a PHP di rilevare utilizzi errati.

```
function saluta($nome) {
    return "Ciao " . $nome;
}

echo saluta("Mario");   // Ciao Mario
```

La stessa funzione può essere dichiarata specificando i tipi:

```
function saluta(string $nome): string {
    return "Ciao " . $nome;
}
```

Parametri con valore di default

Se a un parametro viene assegnato un valore di default nella definizione, il parametro diventa **facoltativo**: se nella chiamata manca, la funzione usa il valore di default. I parametri facoltativi vanno indicati dopo quelli obbligatori.

```php
function anagrafe($nome, $indirizzo, $cf = "non disponibile") {
    echo "Nome: " . $nome . "<br>";
    echo "Indirizzo: " . $indirizzo . "<br>";
    echo "Codice fiscale: " . $cf . "<br>";
}

anagrafe("Mario Rossi", "via Roma 2", "RSSMRA69S12A944X");
anagrafe("Paolo Verdi", "via Parigi 9");   // codice fiscale: non disponibile
```

Passaggio dei parametri: per valore e per riferimento

- **Per valore** (default): la funzione lavora su una **copia** della variabile, quindi la variabile originale non viene modificata;
- **Per riferimento**: la funzione lavora sulla variabile originale, che può quindi essere modificata. Si ottiene anteponendo il simbolo `&` al parametro nella **definizione** della funzione.

Nella chiamata il simbolo `&` non va indicato.

```php
function quadrato($a) {
    $a = $a * $a;
}

function quadratoRif(&$a) {
    $a = $a * $a;
}

$b = 2;
quadrato($b);
echo $b;          // 2: passaggio per valore, $b non cambia
quadratoRif($b);
echo $b;          // 4: passaggio per riferimento, $b viene modificata
```

Il costrutto return

L'istruzione `return` (opzionale) restituisce **un solo valore** al chiamante e **termina l'esecuzione** della funzione.

```php
function divisione($num1, $num2) {
    if ($num2 == 0) {
            return FALSE;   // la funzione termina qui
    }
    return $num1 / $num2;
}

$m = divisione(4, 2);   // 2
$m = divisione(5, 0);   // FALSE
```

> **Attenzione**
>
> Poiché `0` e `false` sono uguali per l'operatore `==`, per controllare l'esito della funzione precedente si usa `$m === false`.

Per restituire più valori si può restituire un **array**, che il chiamante può scomporre con `list()` o con la sintassi abbreviata `[]`. Nella forma posizionale gli elementi vengono associati a partire dall'indice `0`.

```php
function calcola($a, $b) {
    return array($a * 10, $b * 100);
}

list($x, $y) = calcola(2, 3);   // $x = 20, $y = 300
[$x, $y] = calcola(2, 3);       // sintassi abbreviata equivalente
```

Visibilità (scope) delle variabili

Le variabili usate all'interno di una funzione sono **locali**: esistono solo nella funzione e sono diverse da quelle definite all'esterno, anche se hanno lo stesso nome.

```php
function raddoppia($num) {
    $temp = $num * 2;
}

raddoppia(5);
echo $temp;   // ERRORE: $temp esiste solo dentro la funzione
```

Per utilizzare all'esterno un valore calcolato in una funzione, bisogna **restituirlo con return**:

```php
function raddoppia($num) {
    return $num * 2;
}

$temp = raddoppia(5);
echo "Risultato: " . $temp;   // Risultato: 10
```

> **Nota**
>
> Fanno eccezione le variabili **superglobali** (come `$_GET` e `$_POST`), che sono accessibili in qualsiasi punto dello script, anche all'interno delle funzioni.

Riuso delle funzioni: include e require

Le funzioni utili in più pagine si scrivono in un file separato, che viene poi importato con le istruzioni `include` e `require`.

File `funzioni.php`:

```php
<?php
function grassetto(string $stringa): string {
    return "<strong>" . $stringa . "</strong>";
}
```

File `main.php`:

```php
<?php
require_once "funzioni.php";
echo grassetto("testo in grassetto");
```

La differenza tra le due istruzioni riguarda il comportamento in caso di errore (file non trovato):

- `include`: genera un **warning** e lo script continua;
- `require`: genera un **errore fatale** e lo script si interrompe.

Le varianti `include_once` e `require_once` importano il file una sola volta, evitando l'errore causato dalla ridefinizione delle stesse funzioni.

3.4 Esercizi

1. Scrivere una funzione `isPari(int $numero): bool` che stabilisca se un numero è pari.
2. Scrivere una funzione che riceva nome e cognome e restituisca una stringa formattata con le iniziali maiuscole.
3. Scrivere una funzione che riceva un array di numeri e ne restituisca minimo, massimo e media.
4. Scrivere una funzione con un parametro facoltativo per generare un messaggio di saluto formale oppure informale.
5. Dimostrare con un esempio la differenza tra passaggio per valore e passaggio per riferimento.
6. Separare alcune funzioni in `funzioni.php` e importarle in una seconda pagina usando `require_once`.
7. Creare un oggetto `DateTimeImmutable` relativo al fuso `Europe/Rome` e stampare data e ora nei formati italiano e ISO 8601.

> **Istruzioni di consegna**
>
> Consegnare ogni esercizio in un file PHP diverso e numerato. Inserire tutti i file in una cartella denominata `cognome_nome` e consegnarla in formato compresso.

4 Form HTML e interazione con PHP

4.1 Il ruolo dei form

I **form** permettono a una pagina web di raccogliere dati dall'utente — testo, scelte e file — e inviarli a una risorsa sul server. L'elaborazione può avvenire in un'altra pagina PHP oppure nella stessa pagina che contiene il form.

4.2 Il tag FORM

```
<form name="..." action="..." method="..." target="...">
    <!-- campi del form -->
</form>
```

Attributi principali:

- **action**: URL della pagina PHP che riceve i dati;
- **method**: `GET` o `POST`;
- **target**: indica dove visualizzare la risposta; normalmente può essere omesso;
- **enctype**: formato dei dati, obbligatorio con i file (`multipart/form-data`).

> ⚠ Attenzione
>
> Fare attenzione ai percorsi **relativi** e **assoluti** del valore assegnato all'attributo `action`.

FIELDSET e LEGEND

Servono a raggruppare i campi in macro-aree, con un titolo:

```
<fieldset>
    <legend>Dati anagrafici</legend>
    <!-- campi appartenenti al gruppo -->
</fieldset>
```

Ogni `fieldset` dovrebbe avere al massimo un elemento `legend`, collocato come primo elemento del gruppo.

4.3 GET e POST

Caratteristica	GET	POST
Posizione dei dati	Query string dell'URL	Corpo della richiesta
Array PHP	`$_GET`	`$_POST`
Condivisione	URL salvabile e condivisibile	Dati non inclusi nell'URL
Uso tipico	Ricerche, filtri, paginazione	Creazione o modifica di dati, login
Dimensione	Limitata anche da browser e server	Limitata dalla configurazione del server

! GET, POST e sicurezza

POST non cifra i dati: evita solamente di mostrarli nell'URL. Per proteggere le informazioni durante il trasferimento è necessario usare HTTPS. Le password e gli altri dati sensibili non devono essere inseriti in una richiesta GET.

4.4 Il tag LABEL

Associa un'etichetta descrittiva a un campo. L'attributo `for` deve corrispondere all'`id` del campo:

```
<label for="nome">Nome:</label>
<input type="text" id="nome" name="nome">
```

4.5 Il tag INPUT

Con l'attributo `type` si ottengono quasi tutti i campi di un form. `input` è un elemento vuoto e non ha tag di chiusura. Gli attributi più comuni sono `name`, che identifica il dato inviato a PHP, `type`, `value` e `id`.

Testo e password

```
<input type="text" name="nome">
<input type="password" name="pwd">
```

Checkbox e radio

- **checkbox**: selezione indipendente, anche multipla;
- **radio**: una sola scelta possibile; tutti i pulsanti dello stesso gruppo devono avere lo stesso `name`.

```
<input type="checkbox" name="sport" value="calcio" checked> Calcio
<input type="radio" name="corso" value="inf" checked> Informatica
<input type="radio" name="corso" value="ele"> Elettronica
```

Campo nascosto e file

```
<input type="hidden" name="id" value="42">
<input type="file" name="documento">
```

> **Nota**
>
> Il campo `file` richiede `method="POST"` ed `enctype="multipart/form-data"`. I dati del file arrivano in `$_FILES`, non in `$_POST`, e devono essere verificati prima di essere salvati.

Pulsanti

```
<input type="submit" value="Invia">
<input type="reset" value="Ripristina">
```

`submit` invia i dati, `reset` svuota il form.

4.6 TEXTAREA e SELECT

```
<textarea name="messaggio" rows="5" cols="40"></textarea>

<select name="corso">
    <option value="inf" selected>Informatica</option>
    <option value="ele">Elettronica</option>
</select>
```

4.7 Ricevere i dati in PHP

I dati del form arrivano allo script PHP tramite gli array superglobali `$_GET` e `$_POST`, con una chiave uguale all'attributo `name`. I valori semplici ricevuti sono stringhe, anche quando provengono da `input type="number"`.

```
<?php
$nome = $_POST["nome"] ?? "";

if ($nome !== "") {
    echo "Ciao " . htmlspecialchars(
        $nome,
        ENT_QUOTES | ENT_SUBSTITUTE,
        "UTF-8"
    );
}
?>
```

L'operatore `??` fornisce un valore predefinito quando la chiave non esiste. `htmlspecialchars()` codifica il testo per inserirlo in modo sicuro nell'HTML; non sostituisce però la validazione del dato.

Un checkbox non selezionato non invia alcun dato: per verificarlo si usa `isset()`:

```php
<?php
if (isset($_POST["sport"])) {
    echo "Hai selezionato Calcio";
}
?>
```

> **Attenzione**
>
> La chiave `$_POST["..."]` deve corrispondere esattamente (anche nelle maiuscole) all'attributo `name` del campo HTML.

4.7.1 Validare i dati

La **validazione** controlla che un valore rispetti i requisiti previsti; la **sanificazione** può invece modificarlo. Per esempio, `filter_var()` può verificare un indirizzo email:

```php
$email = $_POST["email"] ?? "";

if (filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo "Indirizzo email non valido";
} else {
    echo "Indirizzo email valido";
}
```

> **Non fidarsi dell'input**
>
> Attributi HTML come `required`, `min`, `max` e `type="number"` migliorano l'esperienza dell'utente, ma possono essere aggirati. Il server deve sempre controllare nuovamente tutti i dati ricevuti.

4.8 Esempi completi

4.8.1 Massimo e minimo tra tre numeri

```php
<form action="esempio1.php" method="post">
        <fieldset>
            <legend>Max e min tra 3 numeri</legend>
            <label>Primo numero: </label>
```

```
            <input type="text" name="a1" value="0"><br>
            <label>Secondo numero: </label>
            <input type="text" name="b1" value="0"><br>
            <label>Terzo numero: </label>
            <input type="text" name="c1" value="0"><br>
        </fieldset>
        <input type="submit" value="Invia i dati">
        <input type="reset" value="Cancella i dati">
    </form>
```

```
<?php
$a = $_POST["a1"] ?? null;
$b = $_POST["b1"] ?? null;
$c = $_POST["c1"] ?? null;

if (is_numeric($a) && is_numeric($b) && is_numeric($c)) {
    $a = (float) $a;
    $b = (float) $b;
    $c = (float) $c;

    $max = $a;
    $min = $a;

    if ($b > $max) { $max = $b; }
    if ($b < $min) { $min = $b; }
    if ($c > $max) { $max = $c; }
    if ($c < $min) { $min = $c; }

    echo "minimo = " . $min . "<br>";
    echo "massimo = " . $max . "<br>";
} else {
    echo "Inserire tre numeri validi";
}
?>
```

4.8.2 Calcolatrice con checkbox

L'utente può selezionare più operazioni contemporaneamente:

```
<form action="esempio2.php" method="post">
    <fieldset>
```

```
    <legend>Operazione</legend>
        <input type="checkbox" name="somma" checked> Somma<br>
        <input type="checkbox" name="differenza"> Differenza<br>
        <input type="checkbox" name="prodotto"> Prodotto<br>
        <input type="checkbox" name="quoziente"> Quoziente<br>
    </fieldset>
    <label>Primo operando: </label>
    <input type="text" name="a" value="0"><br>
    <label>Secondo operando: </label>
    <input type="text" name="b" value="0"><br>
    <input type="submit" value="Invia">
    </form>
```

```
<?php
$a = $_POST["a"] ?? null;
$b = $_POST["b"] ?? null;

if (is_numeric($a) && is_numeric($b)) {
    $a = (float) $a;
    $b = (float) $b;

    if (isset($_POST["somma"])) echo ($a + $b) . "<br>";
    if (isset($_POST["differenza"])) echo ($a - $b) . "<br>";
    if (isset($_POST["prodotto"])) echo ($a * $b) . "<br>";
    if (isset($_POST["quoziente"])) {
        echo $b != 0
            ? ($a / $b) . "<br>"
            : "Errore: divisione per zero<br>";
    }
} else {
    echo "Errore: inserire due numeri validi";
}
?>
```

Nota

`is_numeric()` verifica che un valore sia numerico: controllare sempre i dati ricevuti da un form.

4.8.3 Calcolatrice con radio button

Qui è possibile scegliere una sola operazione, quindi si usa uno `switch`:

```
<form action="esempio3.php" method="post">
        <fieldset>
            <legend>Operazione</legend>
            <input type="radio" name="op" value="somma" checked> Somma
            <input type="radio" name="op" value="differenza"> Differenza
            <input type="radio" name="op" value="prodotto"> Prodotto
            <input type="radio" name="op" value="quoziente"> Quoziente
        </fieldset>
    <label>Primo operando: </label>
    <input type="text" name="a" value="0"><br>
    <label>Secondo operando: </label>
    <input type="text" name="b" value="0"><br>
    <input type="submit" value="Invia">
    </form>
```

```
<?php
$a = $_POST["a"] ?? null;
$b = $_POST["b"] ?? null;
$operazione = $_POST["op"] ?? "";

if (is_numeric($a) && is_numeric($b)) {
    $a = (float) $a;
    $b = (float) $b;

    switch ($operazione) {
        case "somma":
            echo $a + $b;
            break;
        case "differenza":
            echo $a - $b;
            break;
        case "prodotto":
            echo $a * $b;
            break;
        case "quoziente":
            echo ($b != 0) ? $a / $b : "ERRORE: divisore nullo!";
            break;
        default:
```

```
            echo "Operazione non valida";
    }
} else {
    echo "Errore: inserire due numeri validi";
}
?>
```

4.9 Esercizi

1. Creare un form con nome e cognome che mostri un messaggio di benvenuto, codificando correttamente l'output HTML.
2. Creare un form con checkbox per scegliere più colori e mostrare quelli selezionati.
3. Creare un form con radio button per scegliere una taglia (`S`, `M`, `L`) e confermarla.
4. Creare una calcolatrice completa utilizzando un menu `select` per scegliere l'operazione.
5. Creare un form che riceva un indirizzo email e lo validi con `FILTER_VALIDATE_EMAIL`.
6. Creare un form di registrazione con campi obbligatori e mostrare un messaggio specifico per ogni dato mancante o non valido.

> **Istruzioni di consegna**
>
> Consegnare ogni esercizio in un file PHP diverso e numerato. Inserire tutti i file in una cartella denominata `cognome_nome` e consegnarla in formato compresso.

5 Cookie e sessioni

5.1 Perché servono cookie e sessioni

Il protocollo HTTP è **stateless**, cioè non conserva memoria delle richieste precedenti. Ogni volta che un utente visita una pagina, la richiesta viene elaborata dal server come indipendente dalle precedenti.

Per questo motivo, quando è necessario conservare alcune informazioni durante la navigazione, si utilizzano principalmente i **cookie** e le **sessioni**.

Alcuni esempi di utilizzo sono:

- mantenere un utente autenticato dopo aver effettuato correttamente il login;
- ricordare la lingua scelta dall'utente;
- conservare i prodotti inseriti nel carrello di un sito di e-commerce.

5.2 I cookie

Un **cookie** è un piccolo insieme di informazioni che viene memorizzato sul computer dell'utente dal browser.

I cookie sono particolarmente utili per conservare informazioni semplici e non sensibili, ad esempio:

- un nome utente;
- una preferenza dell'utente;
- la lingua scelta;
- un valore vero/falso.

I dati contenuti nei cookie vengono salvati come stringhe e possono essere letti, modificati oppure eliminati dall'utente attraverso le impostazioni del proprio browser.

> **Attenzione**
>
> I cookie vengono memorizzati sul computer dell'utente e possono essere modificati o eliminati dall'utente stesso. Per questo motivo non devono essere utilizzati per conservare password o altri dati sensibili.

Creazione di un Cookie

In PHP per creare un cookie si utilizza la funzione `setcookie()`.

La funzione deve essere utilizzata **prima di qualsiasi output**, cioè prima di inviare al browser codice HTML o utilizzare funzioni come `echo()` e `print()`.

Nelle versioni moderne di PHP è consigliabile usare la forma con l'array delle opzioni. Le opzioni principali sono:

- il nome del cookie;
- il valore del cookie;
- la data di scadenza;
- il percorso per il quale il cookie è valido;
- `secure`, che limita l'invio alle connessioni HTTPS;
- `httponly`, che impedisce l'accesso al cookie da JavaScript;
- `samesite`, che limita l'invio del cookie nelle richieste provenienti da altri siti.

Un esempio di creazione di un cookie è il seguente:

```php
<?php

$nomeCookie = "preferenza";
$valoreCookie = "tema_scuro";
$connessioneSicura = isset($_SERVER["HTTPS"])
    && $_SERVER["HTTPS"] !== "off";

setcookie($nomeCookie, $valoreCookie, [
    "expires" => time() + 86400, // un giorno
    "path" => "/",
    "secure" => $connessioneSicura,
    "httponly" => true,
    "samesite" => "Lax"
]);
```

```
    ?>
```

In questo esempio:

- `$nomeCookie` contiene il nome del cookie;
- `$valoreCookie` contiene il valore da memorizzare;
- `time() + 86400` stabilisce una durata di un giorno, cioè 24 ore;
- `path` impostato a `/` rende il cookie valido in tutto il sito;
- `secure`, `httponly` e `samesite` migliorano la protezione del cookie.

È possibile anche creare un cookie utilizzando solamente i due parametri obbligatori:

```
<?php

    setcookie("test_cookie", "Sono un cookie!");

    ?>
```

In questo caso il cookie è un cookie di sessione e scade normalmente alla chiusura del browser. Il nuovo valore sarà disponibile in `$_COOKIE` a partire dalla richiesta successiva, non immediatamente nella stessa esecuzione di `setcookie()`.

Lettura di un Cookie

Una volta creato un cookie, il suo valore può essere letto attraverso l'array superglobale `$_COOKIE`.

Per verificare se un determinato cookie esiste si può utilizzare la funzione `isset()`.

```
<?php

$nomeCookie = "preferenza";

if (isset($_COOKIE[$nomeCookie])) {
    $valore = htmlspecialchars(
        $_COOKIE[$nomeCookie],
        ENT_QUOTES | ENT_SUBSTITUTE,
        "UTF-8"
    );
```

```
    echo "Il cookie vale: " . $valore;
} else {
    echo "Il cookie non è stato impostato";
}

    ?>
```

Modifica di un Cookie

Per modificare un cookie è sufficiente utilizzare nuovamente la funzione `setcookie()` con lo stesso nome e assegnare un nuovo valore.

```
<?php

$nomeCookie = "preferenza";

setcookie($nomeCookie, "tema_chiaro", [
    "expires" => time() + 86400,
    "path" => "/",
    "httponly" => true,
    "samesite" => "Lax"
]);

    ?>
```

Eliminazione di un Cookie

Per eliminare un cookie è possibile impostare una data di scadenza nel passato.

```
<?php

$nomeCookie = "preferenza";

setcookie($nomeCookie, "", [
    "expires" => time() - 3600,
    "path" => "/",
    "httponly" => true,
    "samesite" => "Lax"
]);

    ?>
```

Per eliminare correttamente un cookie bisogna usare lo stesso nome, percorso e dominio usati durante la creazione.

5.3 Le sessioni

Le **sessioni** permettono di conservare informazioni durante la navigazione dell'utente, ma a differenza dei cookie i dati vengono memorizzati sul **server**.

Quando viene avviata una sessione, il server associa alla navigazione un identificatore chiamato **session ID**.

Il session ID viene normalmente memorizzato sul computer dell'utente tramite un cookie chiamato `PHPSESSID`.

I dati della sessione vengono invece memorizzati sul server e possono essere utilizzati nelle diverse pagine del sito attraverso l'array superglobale `$_SESSION`.

> **Nota**
>
> Nei cookie vengono memorizzati direttamente i dati sul computer dell'utente. Nelle sessioni i dati vengono invece mantenuti sul server; il browser conserva normalmente solamente il session ID che permette di identificare la sessione.

Avvio di una sessione

Per utilizzare le sessioni all'interno di una pagina PHP è necessario utilizzare la funzione `session_start()`.

La funzione non richiede parametri e deve essere chiamata prima di qualsiasi output.

```php
<?php

    session_start();

    ?>
```

> **Attenzione**
>
> La funzione `session_start()` deve essere eseguita prima di inviare al browser qualsiasi codice HTML o altro output tramite funzioni come `echo()` e `print()`.

Salvataggio dei dati nella sessione

Una volta avviata la sessione, è possibile salvare le informazioni all'interno dell'array superglobale `$_SESSION`.

Ad esempio:

```php
<?php

    // Apro la sessione
    session_start();

// Recupero i dati inviati dal form.
$username = trim($_POST["user"] ?? "");

if ($username !== "") {
    // Dopo la verifica delle credenziali, rinnovo l'ID
    // per contrastare gli attacchi di session fixation.
    session_regenerate_id(true);

    $_SESSION['username'] = $username;
    $_SESSION['autenticato'] = true;
}

    ?>
```

In questo modo nella sessione vengono memorizzati il nome utente e lo stato di autenticazione. La password non deve essere conservata nella sessione.

Utilizzo dei dati della sessione

I dati salvati nella sessione possono essere recuperati anche da una pagina diversa.

È sufficiente richiamare nuovamente `session_start()` e accedere agli elementi dell'array `$_SESSION`.

```php
<?php

    // Apro la sessione
    session_start();

$username = $_SESSION["username"] ?? null;
```

```php
if ($username !== null) {
    echo "Ciao " . htmlspecialchars(
        $username,
        ENT_QUOTES | ENT_SUBSTITUTE,
        "UTF-8"
    );
} else {
    echo "Utente non autenticato";
}

    ?>
```

In questo modo le informazioni precedentemente salvate possono essere utilizzate nelle successive pagine del sito.

5.4 Eliminazione dei dati di sessione

PHP mette a disposizione diverse funzioni per eliminare le informazioni memorizzate in una sessione.

Eliminazione di una singola variabile

Per eliminare una specifica variabile di sessione si utilizza la funzione `unset()`.

```php
<?php

    unset($_SESSION['username']);

    ?>
```

In questo modo viene eliminata solamente la variabile `username`.

Eliminazione di tutte le variabili

Per eliminare tutte le variabili contenute nell'array `$_SESSION` si può utilizzare:

```php
<?php

    session_unset();

    ?>
```

È inoltre possibile inizializzare nuovamente l'array della sessione:

```php
<?php

    $_SESSION = array();

    ?>
```

Distruzione della sessione

`session_destroy()` elimina i dati associati alla sessione sul server, ma non svuota automaticamente `$_SESSION` e non elimina il cookie con il session ID. Per un logout completo si eseguono tutti i passaggi:

```php
<?php
session_start();

$_SESSION = [];

if (ini_get("session.use_cookies")) {
    $parametri = session_get_cookie_params();

    setcookie(session_name(), "", [
        "expires" => time() - 42000,
        "path" => $parametri["path"],
        "domain" => $parametri["domain"],
        "secure" => $parametri["secure"],
        "httponly" => $parametri["httponly"],
        "samesite" => $parametri["samesite"] ?? "Lax"
    ]);
}

session_destroy();
?>
```

La differenza tra le operazioni è importante:

- `session_unset()` azzera immediatamente il contenuto dell'array `$_SESSION`;
- `session_destroy()` elimina i dati associati alla sessione sul server, ma da solo non modifica le variabili già caricate nello script e non elimina il cookie di sessione.

5.5 Esercizi

1. Creare un cookie contenente il proprio nome e visualizzarlo in una pagina PHP successiva.
2. Creare un cookie contenente la lingua preferita dell'utente e visualizzare un messaggio diverso in base alla lingua scelta.
3. Creare una pagina che permetta di modificare ed eliminare un cookie mantenendo coerenti le opzioni `path`, `httponly` e `samesite`.
4. Creare una sessione nella quale salvare il nome e la classe dello studente, quindi visualizzare tali informazioni in una seconda pagina.
5. Creare un semplice sistema di autenticazione simulato che memorizzi nella sessione il nome dell'utente e un valore booleano, senza salvare la password.
6. Creare una pagina di logout che svuoti `$_SESSION`, elimini il cookie di sessione e chiami `session_destroy()`.

> ⚠ Istruzioni di consegna
>
> Consegnare ogni esercizio in un file PHP diverso e numerato. Inserire tutti i file in una cartella denominata `cognome_nome` e consegnarla in formato compresso.