



PHP ESSENTIALS #3

By WI400 Team

: basi del linguaggio



■ Agenda

- ✓ Basi del linguaggio
- ✓ variabili
- ✓ Operatori
- ✓ Strutture di controllo

Tags PHP

- I documenti **PHP** sono pagine **HTML** con comandi **PHP** incorporati
- Quando il **browser** richiede il documento **PHP**, Il web server esegue i comandi **PHP** ed invia il loro risultato insieme all'**HTML** contenuto nella pagina richiesta
- Come fa il web server a capire qual'è il codice **PHP** da eseguire:

```
<?php echo 'Long Tags' ; ?>
```

```
<? echo 'Short Tags' ; ?>
```

```
<% echo 'ASP Tags' ; %>
```

```
<script language="php"> echo 'Really Long Tags' ; </script>
```

Nota: ogni riga di php termina con il ;

Hello World!

Considerando il seguente documento PHP:

```
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <?php
      echo "<h1> Hello World! </h1>";
    ?>
  </body>
</html>
```

- Il web server esegue le istruzioni **PHP** e le sostituisce con il suo **output**
- L'istruzione **echo** produrrà come output la stringa “<h1> **Hello World!** </h1>”

Hello World!

- Il risultato finale che il **web server** invierà al browser del richiedente sarà il seguente:

```
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1> Hello World! </h1>
  </body>
</html>
```



Hello World!

Regole di sintassi

- Le regole da tenere sempre in mente sono le seguenti:
 - Il **PHP** è delimitato dai tag di apertura e di chiusura
`<?php...?>`
- Le righe **PHP** terminano generalmente con il punto e virgola (;)
- E' possibile (*doveroso* 😊) inserire **commenti** al codice **PHP**:
- facendo precedere il commento con il doppio slash: //
- utilizzando la combinazione /* e */ rispettivamente come apertura e chiusura del commento:

```
<?php
/*
 * Questo è un commento stile C
 */

// Questo è un commento
// stile C++

# Questo è un commento
# stile Perl
```

Configurazione base PHP

- Il file `php.ini`
- Direttive principali di configurazione
 - ✓ Language options
 - ✓ Error handling
 - ✓ Error logging
 - ✓ Paths e directory
 - ✓ Resource limits
 - ✓ File uploads
- `phpinfo()`

Configurazione base: php.ini

- Il file **php.ini** è il file di configurazione con tutte le direttive di esecuzione del **PHP**
- Il file **php.ini** è un file di testo
- Il file è strutturato in **sezioni**
- Ciascuna sezione contiene una descrizione e tutte le direttive relative alla sezione stessa
- Ciascuna riga contiene una direttiva nella forma:
direttiva=valore
- Le righe che iniziano con il “;” rappresentano commenti e vengono quindi ignorate

Configurazione base: php.ini

- Alcune delle direttive principali del file **php.ini** sono le seguenti:
 - ✓ `short_open_tag`, `asp_tags`
 - ✓ `error_reporting`, `display_errors`, `display_startup_errors`
 - ✓ `log_errors`, `error_log`, `log_errors_max_len`
 - ✓ `include_path`
 - ✓ `max_execution_time`, `memory_limit`
 - ✓ `file_uploads`, `upload_tmp_dir`, `upload_max_filesize`

Direttive principali: Language options

- **short_open_tag** abilita/disabilita la possibilità di utilizzare gli short tag per l'apertura e la chiusura del codice PHP (`<? e ?>`).
- **asp_tags** abilita/disabilita la possibilità di utilizzare i tag in formato asp per l'apertura e la chiusura del codice PHP (`<% e %>`).

Direttive principali: Error Handling

- **error_reporting** definisce il livello degli errori da visualizzare (`E_ALL` | `E_STRICT` per la visualizzare tutti gli errori).
- **display_errors** è la direttiva che definisce se visualizzare o meno gli errori (1 o 0). In fase di sviluppo dell'applicazione dovrebbe essere impostato a 1 mentre in fase di produzione a 0.
- **display_startup_errors** è la direttiva che definisce se visualizzare o meno gli errori che si verificano durante l'avvio del PHP (1 o 0). Dovrebbe avere la stessa configurazione della direttiva `display_errors`.

Direttive principali: Error Logging

- **log_errors** è la direttiva che permette la scrittura dei messaggi di errore su file di log (**On** o **Off**). Dovrebbe essere sempre impostato ad **On**.
- **error_log** è la direttiva che definisce il file di log sul quale andare a scrivere i messaggi di errore.
- **log_errors_max_len** è la direttiva che definisce la lunghezza massima del file di log in bytes.

Direttive principali: Paths e directory

- **include_path** è la direttiva che specifica le directory dove il PHP deve cercare i file quando vengono richiamate funzioni come `include`, `include_once`, `require`, `require_once` e `file_get_contents` (tratteremo più avanti....).
- Il carattere che separa una directory dall'altra varia in base al sistema operativo sul quale è installato il **PHP**:
 - ✓ Per sistemi **UNIX** si utilizza il `:`
 - `“./php/includes”`
 - ✓ Per sistemi Windows si utilizza il `;`
 - `“.;c:\php\includes”`

Direttive principali: Resource limits

- **max_execution_time** è la direttiva che definisce il numero massimo di secondi che possono essere impiegati da uno script PHP per la sua esecuzione.
- **memory_limit** è la direttiva che definisce la quantità massima di memoria (espressa in MB) che uno script PHP può utilizzare durante la sua esecuzione.

Direttive principali: File uploads

- **file_uploads** abilita o disabilita la possibilità di eseguire upload di file da form **HTML**
- **upload_tmp_dir** contiene la directory per il salvataggio temporaneo dei file derivanti da form **HTML**
- **upload_max_filesize** definisce le dimensioni massime dei file che possono essere caricati tramite form **HTML**

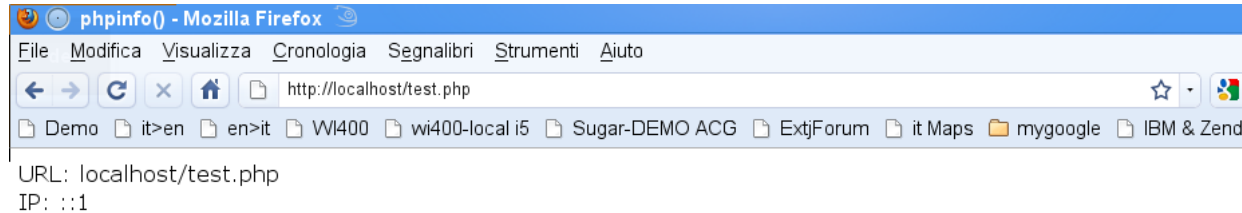
phpinfo()

- E' una funzione nativa del PHP
- Visualizza tutte le informazioni riguardanti il PHP installato (opzioni di compilazione, estensioni, versione del PHP, informazioni del sistema operativo del server, ecc.)

```
<html>
  <head>
    <title>phpinfo()</title>
  </head>
  <body>
    <?php phpinfo(); ?>
  </body>
</html>
```


phpinfo(): risultato

- Questa è il risultato della funzione `phpinfo()`:



System	Linux linux-xyw5 2.6.31.12-0.2-desktop #1 SMP PREEMPT 2010-03-16 21:25:39 +0100 i686
Build Date	Mar 31 2010 13:13:13
Configure Command	'./configure' '--prefix=/usr/local/zend' '--with-config-file-path=/usr/local/zend/etc' '--disable-debug' '--enable-inline-optimization' '--disable-all' '--enable-libxml' '--enable-session' '--enable-spl' '--enable-xml' '--enable-hash' '--enable-reflection' '--with-pear' '--with-apxs2=/usr/local/zend/apache2/bin/apxs' '--with-layout=GNU' '--enable-filter' '--with-pcre-regex' '--with-zlib=/usr/local/zlib-1.2.3' '--enable-simplexml' '--enable-dom' '--with-libxml-dir=/usr/local/libxml2-2.6.32' '--with-openssl=/usr/local/openssl-0.9.8n' '--enable-pdo' '--with-pdo-sqlite' '--with-iconv' '--with-readline=/usr/local/readline-5.2' '--with-sqlite3' '--disable-pear'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/zend/etc
Loaded Configuration File	/usr/local/zend/etc/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626

Basi del linguaggio

- Introduzione
- Struttura lessicale
 - ✓ Case sensitivity
 - ✓ Statements e separatore di istruzioni
 - ✓ Spazi bianchi e interruzioni di linea
 - ✓ Commenti
- Tipi di dati
- Variabili
- Operatori
- Strutture di controllo

Basi del linguaggio: introduzione

- Come tutti gli altri linguaggi di programmazione anche il **PHP** ha delle regole di base da seguire
- Il **PHP** è influenzato molto da altri linguaggi come **Perl** e **C** e le sue regole di base assomigliano molto a questi due linguaggi

Basi del linguaggio: struttura lessicale

- La **struttura lessicale** è l'insieme di regole che definiscono come deve essere scritto un programma
- E' **sintassi** vera e propria del linguaggio
- Riguarda informazioni come nomi delle **variabili**, **commenti** al codice e come le **istruzioni** sono separate una dall'altra

Struttura lessicale: case sensitivity

- Le variabili **PHP** sono **case-sensitive**
\$variabile e **\$VARIABLE** sono due variabili diverse

`$variabile` e `$VARIABLE`

- I nomi delle **classi** e delle **funzioni** definite dall'utente, così come **costrutti** e parole chiavi del linguaggio sono **case-insensitive**:
echo ed **ECHO** produrranno lo stesso risultato

Struttura lessicale: statements e separatore di istruzioni

- Uno **statement** è una raccolta di codice **PHP**
- Può essere semplice come assegnare un valore ad una **variabile**

```
$linguaggio = 'PHP';
```

- Può essere complesso come una **struttura di controllo**

```
if ($linguaggio == PHP) {  
    echo 'wi400';  
}
```

- Il **PHP** utilizza il carattere ‘;’ per separare istruzioni semplici
- Uno statement complesso non ha bisogno del ‘;’ dopo le parentesi graffe che racchiudono il codice dello statement
Il ‘;’ è opzionale (anche se consigliata) per l’istruzione immediatamente precedente il **tag** di chiusura del **PHP**

Struttura lessicale: spazi bianchi e interruzioni di linea

- Generalmente gli **spazi** bianchi e le interruzioni di linea non cambiano il comportamento del codice
- E' possibile scrivere aggiungendo **spazi**

```
nome_funzione ( $param1, $param2, $param3 );
```

- E' possibile scrivere eliminando gli **spazi**

```
nome_funzione($param1,$param2,$param3);
```

- E' possibile scrivere su più **linee**:

```
nome_funzione(  
    $param1,  
    $param2,  
    $param3  
);
```

Tipi di dati

- Il PHP mette a disposizione 8 tipi di dati di cui 4 scalari, 2 composti e 2 tipi speciali
 - I dati scalari sono:
 - ✓ **integers**
 - ✓ **numeri floating point**
 - ✓ **strings**
 - ✓ **booleans**
 - I dati composti sono:
 - ✓ **arrays**
 - ✓ **Objects**
 - I dati speciali sono:
 - ✓ **resource**
 - ✓ **null**



■ Variabili

- ✓ Uso delle variabili
- ✓ Tipi di dati
- ✓ Visibilità delle variabili
- ✓ Array superglobal
- ✓ Variabili di variabili
- ✓ Costanti

Variabili: uso delle variabili

- Le **variabili** sono contenitori di informazioni (**dati**)
- Possono contenere tutti i tipi di dati messi a disposizione dal **PHP**
- Le variabili **PHP** sono identificate dal simbolo del **dollaro**:
\$variabile
- I nomi delle **variabili** possono contenere caratteri alfabetici, numeri e underscore ma non possono iniziare con numeri
- E' possibile assegnare una **stringa** ad una **variabile** attraverso l'operatore di assegnazione '='
- E' possibile visualizzare il contenuto di una **variabile** attraverso il costrutto 'echo': **echo \$variabile;**

dov'è l'errore ?

```
$myvar = "ciao";  
echo $myvar;  
$1myvar = "pippo";
```

Struttura lessicale: l'output

- l'apice 'singolo' e l'apice “doppio”
- il PHP tenta di “risolvere” le variabili all'interno dei doppi apici (“), mentre quelle tra apici singoli (') vengono lasciate inalterate

```
print "Il valore di a è $a";  
print ('Con $a si indica una variabile');
```

Variabili: tipi di dati

- PHP è un linguaggio loosely-typed. Questo significa che non è necessario definire il tipo di dato che una variabile può contenere
- PHP prova a fare quello che “*tu vorresti che facesse*”

```
<?php  
$x = 5; // Questa variabile contiene un intero  
$y = '3'; // Questa variabile contiene una stringa  
$z = $x + $y; // Eseguo la somma di un intero e di una stringa  
echo "$x + $y = $z"; // Visualizzo il risultato della somma
```



5 + 3 = 8

Variabili: visibilità

- l'ambito di visibilità (**scope**) di una **variabile** è il contesto (parte dello script) in cui è possibile accedere al suo contenuto
- per lo “**scope**” ci sono due categorie di variabili:
 - **globali**, valide nella parte più esterna degli script
 - **locali**, valide all'interno di funzioni o classi
- le variabili sono visibili nella parte **esterna** dello script, ma non all'interno di funzioni o classi (a meno di non dichiararle con **global**)

Variabili: array superglobali

- PHP ha alcune speciali variabili chiamate “*superglobal*” *array*

vengono utilizzate, ad esempio, per il reperimento dei dati di **POST** o **GET** da una form **HTML**, o per i dati di **sessione** o **cookies**.

- - **\$_GET**
- **\$_POST**
- **\$_COOKIES**

Variabili: speciali

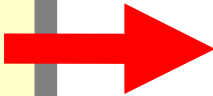
- Degli esempi:

Variabili del Server	Descrizione
<code>\$_SERVER["PHP_SELF"]</code>	Il percorso dello script (da document root)
<code>\$_SERVER["QUERY_STRING"]</code>	Query string della URI richiesta
<code>\$_SERVER["REQUEST_URI"]</code>	URI richiesta del client
<code>\$_SERVER["SERVER_NAME"]</code>	Nome del server
<code>\$_SERVER["SERVER_PORT"]</code>	Port number del server
Variabili di Ambiente	Descrizione
<code>\$_ENV["PWD"]</code>	Directory corrente
<code>\$_ENV["PATH"]</code>	Percorsi per la ricerca di eseguibili e librerie
<code>\$_ENV["USERNAME"]</code>	Nome dell'utente
<code>\$_ENV["HOME"]</code>	Home directory dell'utente
Variabili di GET/POST	Descrizione
<code>\$HTTP_POST_FILES[<file>]</code>	Matrice di attributi del <file> di upload
Variabili Cookies	Descrizione
<code>\$HTTP_COOKIE_VARS[<cookie>]</code>	Array associativo delle coppie <cookie> = valore
Variabili di Sessione	Descrizione
<code>\$_SESSION[<var>]</code>	Array associativo delle coppie <var> = valore

Variabili: controllo

- esiste la possibilità di controllare se una variabile è assegnata o meno (se è stata inizializzata) con la funzione **isset()**.

```
<?php
$a = 1;
if ((isset ($a))) {
    echo "a e' assegnata";
}
echo "<br>";
if ( (! isset ( $b ))) {
    print "b non e' assegnata";
}
```



a e' assegnata
b non e' assegnata

Variabili: variabili di variabili

- in **PHP** è possibile utilizzare la **variabile di variabile**.
è bene però ricordare che questa cosa va evitata se possibile, in quanto rende il codice di difficile interpretazione.

```
$varname = 'myvar';  
$myvar = 'Hello, World!';  
echo $$varname;
```

\$ + myvar

Hello, World!

Variabili: costanti

- in **PHP** è possibile definire variabili “**COSTANTI**”
- non sono **variabili** a tutti gli effetti, ma sono contenitori di dati, non iniziano con il **\$** ed è prassi comune definirle sempre in **MAIUSCOLO**.
- la principale differenza è che il valore di una costante **non** cambia mai
- **non** si può sovrascrivere il valore di una costante neanche ridefinendola.

```
define('NAME', 'ciao');  
define('NAME', 'sovrascrivo');  
echo NAME;
```



ciao



▪ Operatori

- ✓ Precedenza Operatori
- ✓ Aritmetici
- ✓ Assegnazione
- ✓ Comparazione
- ✓ Logici
- ✓ Stringhe
- ✓ Altri

Operatori: Precedenza degli operatori

- la precedenza di un **operatore** determina quale operazione viene eseguita per prima
- ecco perchè il risultato dell'espressione **$1 + 2 * 3$** è **7** e non **9** (*la moltiplicazione ha la precedenza sull'addizione*)
- l'utilizzo delle parentesi “**guida**” il calcolo delle espressioni:
 $(1 + 2) * 3$ dà come risultato **9**.

Operatori: Aritmetici

- negazione $-\$a$
- somma $\$a + \b
- sottrazione $\$a - \b
- moltiplicazione $\$a * \b
- divisione $\$a / \b
- modulo $\$a \% \b

Operatori: Assegnazione

- di base: $\$a = \b
- combinata: $\$a += \b
- l'assegnazione combinata funziona con tutti gli operatori aritmetici

```
$a = 3;  
$b = 5;  
echo $a += $b;
```

→ 8

Operatori: Comparazione

- equal: $\$a == \b
- identical: $\$a === \b
- not equal: $\$a != \b o $\$a <> \b
- not identical: $\$a !== \b
- less than: $\$a < \b
- greater than: $\$a > \b
- less than or equal to: $\$a <= \b
- greater than or equal to: $\$a >= \b

Operatori: Logici

- and: $\$a \ \&\& \ \b
- or: $\$a \ || \ \b
- not: $!\$a$
- xor: $\$a \ xor \ \b

Operatore	Esempio	Vero se...
not	$!\$a$	$\$a$ è falsa
and	$\$a \ \&\& \ \$b \ (\$a \ and \ \$b)$	$\$a$ e $\$b$ sono entrambe vere
or	$\$a \ \ \$b \ (\$a \ or \ \$b)$	almeno una tra $\$a$ e $\$b$ è vera
xor	$\$a \ xor \ \b	solo una tra $\$a$ e $\$b$ è vera

Operatori: Logici

- Ricorda:

0 = false

1 = true

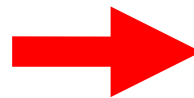
```
<?php
$a=0;
if(!$a) {
    echo "hello world";
}
```

Hello world

Operatori: Stringhe

- concatenazione: `$a . $b`
- Concatenazione e assegnazione: `$a .= $b`

```
$a = "hello ";  
$b = "world";  
echo $a . $b;  
echo "<br>";  
$a .= $b;  
echo $a;
```



```
Hello world  
Hello world
```

Operatori: Altri

- pre-incremento: **++\$a**
- post-incremento: **\$a++**
- pre-decremento: **--\$a**
- post-decremento: **\$a--**
- soppressione degli errori: **@myfunction()**
(*attenzione all'uso...*)



■ Strutture di controllo

- ✓ if
- ✓ switch
- ✓ cicli while
- ✓ cicli do-while
- ✓ cicli for
- ✓ cicli foreach

Strutture di controllo

- il codice procedurale viene eseguito, in genere, dall'alto al basso
- le specifiche condizionali (**conditional statement**) consentono di prendere delle decisioni e alterarne il flusso.
- in questo modo è possibile creare delle parti di codice che verranno eseguite solo quando viene verificata una particolare condizione.
- tutte le condizioni verificheranno un risultato **true** o **false**.

Strutture di controllo: if

- espressione booleana

```
$a = FALSE;  
$b = TRUE;  
if ($a and $b) print ("a e b sono vere");
```

- espressione di confronto

```
$a = " 04";  
$b = "0003";  
if ($a>$b) print("a è maggiore di b");  
if (strcmp($a,$b)) print("a è più grande di b");
```

- espressione condizionale (*ternary operator*)

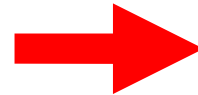
```
// Un numero e' pari se, diviso per due, non da' resto  
echo "\$n è ". ( $n % 2 == 0 ? "pari" : "dispari" );
```

Strutture di controllo: if

- if-elseif-else

```
<?php
$hungry = true;
$thirsty = true;

if ($hungry && $thirsty) {
    echo 'Eat and Drink';
} elseif ($hungry) {
    echo 'Eat';
} elseif ($thirsty) {
    echo 'Drink';
} else {
    echo 'Do Nothing';
}
```



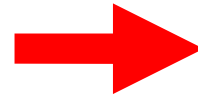
Eat and Drink

Strutture di controllo: if

- if-elseif-else

```
<?php
$hungry = true;
$thirsty = false;

if ($hungry && $thirsty) {
    echo 'Eat and Drink';
} elseif ($hungry) {
    echo 'Eat';
} elseif ($thirsty) {
    echo 'Drink';
} else {
    echo 'Do Nothing';
}
```



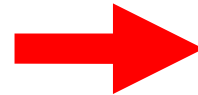
Eat

Strutture di controllo: if

- if-elseif-else

```
<?php
$hungry = false;
$thirsty = true;

if ($hungry && $thirsty) {
    echo 'Eat and Drink';
} elseif ($hungry) {
    echo 'Eat';
} elseif ($thirsty) {
    echo 'Drink';
} else {
    echo 'Do Nothing';
}
```



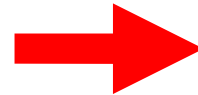
Drink

Strutture di controllo: if

- if-elseif-else

```
<?php
$hungry = false;
$thirsty = false;

if ($hungry && $thirsty) {
    echo 'Eat and Drink';
} elseif ($hungry) {
    echo 'Eat';
} elseif ($thirsty) {
    echo 'Drink';
} else {
    echo 'Do Nothing';
}
```



Do Nothing

Esercizio 1

- Crea uno script che memorizzi il tuo **nome** in una **variabile** e lo visualizza (in **output**)
 - ✓ *Ricordare i tag **PHP***
 - ✓ *Ricordare di utilizzare l'estensione **.php** per il file*
- Estendere lo script per memorizzare l'età così come avete fatto per il nome. Lo script deve visualizzare due **messaggi** diversi a seconda se l'età è maggiore o minore di **17**.

Esercizio 1: Solution

- Una delle molte soluzioni al problema precedente potrebbe essere la seguente:

```
<?php
$name = "Mario Rossi";
$age = 50;
if ($age > 17)
{
    echo "Ciao $name hai già preso la patente di guida?";
}
else
{
    echo "Ciao $name ricordati di iscriverti alla scuola guida
quando sarai maggiorenne";
}
```

Strutture di controllo: Switch

- quando l'esecuzione di un **blocco** di istruzioni piuttosto che un altro dipende da una **espressione**, è conveniente usare lo **switch**
- all'uso dello **switch** va abbinato l'utilizzo del **break**, che consente l'immediata uscita dal blocco di istruzioni
- può essere indicato un blocco di **default** quando tutti gli altri non si applicano.

Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```

Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```

Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```


Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```

Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```

Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```

Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```

Strutture di controllo: Switch

```
<?php
switch ( $color) {
    case 'red' :
        echo 'The color is red.';
        break;
    case 'blue' :
        echo 'The color is blue.';
        break;
    default :
        echo 'The color is not red or blue.';
        break;
}
```

Esercizio 2:

- Crea uno **script** che memorizzi il tuo nome in una **variabile** e lo visualizzi in output
 - ✓ ricordare i tag **PHP**
 - ✓ ricordare di utilizzare l'estensione **.php** per il file
- Estendere lo script per memorizzare l'età così come avete fatto per il nome. Lo script deve visualizzare due messaggi diversi a seconda se l'età è maggiore o minore di **17**.
- Utilizzare il costrutto **switch** per il secondo punto dell'esercizio

Esercizio 2: Solution

- La soluzione al problema della slide precedente potrebbe essere il seguente:

```
<?php
$name = "Marco Rossi";
$age = 50;
switch($age > 17)
{
case true:
    echo "Ciao $name hai già preso la patente di guida?";
    break;
case false:
    echo "Ciao $name ricordati di iscriverti alla scuola guida
        quando sarai maggiorenne";
    break;
}
```

Strutture di controllo: Cicli While

- I cicli **while** sono il modo più semplice di eseguire più volte la stessa parte di codice.

```
while (espressione) {  
    blocco-istruzioni;  
}
```

- Se l'**espressione** di controllo è vera viene eseguito lo **statement** e viene rieseguito il controllo (fino a quando l'**espressione** è verificata)
- La condizione viene controllata prima che il codice venga eseguito

```
<?php  
$i = 0;  
while ( $i < 10 ) {  
    echo $i;  
    $i ++;  
}
```


Strutture di controllo: Cicli Do-While

- I cicli **do-while** sono identici ai cicli **while** ad eccezione del fatto che l'espressione di controllo viene valutata soltanto **dopo** che il codice è stato eseguito
do
 statement
while (*espressione*)
- Questo garantisce che il codice viene eseguito almeno una volta

```
<?php  
  
$i = 0;  
  
do {  
    echo $i;  
    $i ++;  
} while ( $i < 10 );
```

Strutture di controllo: Cicli For

- I cicli **for** sono molto simili ai cicli **while**
- Si differenziano dai cicli **while** in quanto aggiungono l'inizializzazione e l'espressione di manipolazione del contatore

for (*inizializzazione*; *espressione*; *incremento*)
statement

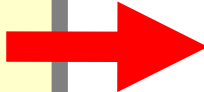
```
<?php  
  
for($i = 0; $i < 10; $i ++) {  
    echo $i;  
}
```

Strutture di controllo: Cicli For

- Il **for()** loops è indicato nei casi in cui si vuole incrementare un contatore all'interno del loop, e quando, solitamente, si conosce il numero di *iterazioni* da effettuare

```
for (inizio;condiz;contatore) {  
    blocco-istruzioni;  
}
```

```
// I quadrati dei primi n  
// numeri interi  
$n = 5;  
for ($i=1; $i<=$n; $i++) {  
    echo "Il quadrato di $i  
        e' " . ($i*$i) . "<br>";  
}
```



```
Il quadrato di 1 e' 1.  
Il quadrato di 2 e' 4.  
Il quadrato di 3 e' 9.  
Il quadrato di 4 e' 16.  
Il quadrato di 5 e' 25.
```

Strutture di controllo: Cicli Foreach

- I cicli **foreach** sono utili soprattutto per scorrere gli elementi di un array
- *Gli array saranno affrontati successivamente*
- Esistono due tipi di sintassi per il costrutto **foreach**
 - ✓ per recuperare soltanto i **valori** di un array
 - ✓ per recuperare **chiavi** e **valori** di un array

```
<?php
foreach ( $_POST as $valore ) {
    echo "<p>Il valore è $valore</p>";
}

foreach ( $_POST as $chiave => $valore ) {
    echo "<p>La chiave è $chiave e il valore è $valore</p>";
}
```

Strutture di controllo: Cicli Foreach

- esempio:

```
$arr = array("one", "two", "three");  
foreach ($arr as $value) {  
    echo "Valore: $value<br>";  
}  
echo "<br>";  
foreach ($arr as $key => $value) {  
    echo "Chiave: $key; Valore: $value<br>\n";  
}
```



```
Valore: one  
Valore: two  
Valore: three  
  
Chiave: 0; Valore: one  
Chiave: 1; Valore: two  
Chiave: 2; Valore: three
```

Esercizio 3:

- Crea uno script che memorizzi tre variabili: \$affamato, \$mele, \$pesche
 - ✓ \$affamato = 10;
 - ✓ \$mele = 4;
 - ✓ \$pesche = 5;
- Fino a quando \$affamato non è 0 “mangiare” una mela o una pesca. Mangiando una mela diminuisce \$affamato di 1 mentre mangiando una pesca diminuisce \$affamato di 2
- Si dovrebbe mangiare il frutto del quale si ha più disponibilità.
- *Non preoccupatevi di rimanere a corto di cibo* 😊

Esercizio 3: Solution

```
<?php

$affamato = 10;
$mele = 4;
$pesche = 5;
while ( $affamato > 0 ) {
    if ( $mele > $pesche ) {
        $mele --;
        echo "Ho mangiato una mela<br>";
    } else {
        $pesche --;
        $affamato --;
        echo "Ho mangiato una pesca<br>";
    }
    $affamato --;
    echo "Ho ancora $affamato fame<br>";
}
```



QUESTION TIME ?



Nome _____

Cognome _____

Data _____



ARRIVEDERCI



TITOLO