PHP Arrays

Arrays

An array is a data structure that stores multiple values in a single variable.
Use: Used to store lists of data such as users, products, or marks.

Example:

```
<?php

$colors = ["Red", "Green", "Blue"];

?>
```

Indexed Arrays

Indexed arrays use numeric indexes starting from 0.
Use: Used when data is stored in a list format.

Example:

```
<?php

$cars = ["BMW", "Audi", "Swift"];

echo $cars[1];  // Audi

?>
```

Associative Arrays

Associative arrays use named keys.
Use: Used when data needs meaningful keys.

Example:

```
<?php

$student = ["name" => "John", "age" => 20];

echo $student["name"];

?>
```

Create Arrays

Arrays can be created using array() or [].
Use: Initializes multiple values at once.

Example:

```
<?php

$fruits = array("Apple", "Banana");

?>
```

Access Array Items

Array items are accessed using index or key.
Use: Retrieve stored values.

Example:

```
<?php

echo $fruits[0];

?>
```

Update Array Items

Used to change an existing value in an array.
Use: Modify stored data.

Example:

```
<?php

$fruits[0] = "Mango";

?>
```

Add Array Items

New items can be added using array index or array_push().
Use: Insert new data.

Example:

```
<?php

array_push($fruits, "Orange");

?>
```

Remove Array Items

Items can be removed using unset() or array_pop().
Use: Delete unwanted data.

Example:

```
<?php

unset($fruits[1]);

?>
```

Sorting Arrays

Sorting arranges array items in a specific order.
Use: Display data in order.

Example:

```
<?php

sort($fruits);

?>
```

Multidimensional Arrays

Arrays that contain other arrays.
Use: Store complex data structures.

Example:

```
<?php

$students = [
  ["name" => "John", "age" => 20],

  ["name" => "Anna", "age" => 22]

];

echo $students[1]["name"];

?>
```

PHP Regular Expressions (RegEx)

A **Regular Expression (RegEx)** is a sequence of characters that defines a search pattern. It is mainly used to **search**, **match**, and **replace** text in strings.

Regular Expression Modifiers

Modifiers define **how** the pattern matching is performed.

Modifier	**Description**
i	Case-insensitive search
m	Multiline search
u	Enables UTF-8 matching

Example

```
<?php

$text = "PHP is great";

if (preg_match("/php/i", $text)) {

  echo "Match found";

}

?>
```

Regular Expression Patterns (Brackets)

Brackets are used to match a **range of characters**.

Expression	**Description**
[abc]	Matches a, b, or c

Expression	Description
[^abc]	Matches any character except a, b, c
[a-z]	Matches lowercase letters
[A-Z]	Matches uppercase letters
[A-z]	Matches both upper and lowercase
[0-9]	Matches digits
[0-5]	Matches digits between 0 and 5
[123]	Matches digits 1, 2, or 3

Example

```
<?php

$text = "Hello123";

if (preg_match("/[0-9]/", $text)) {

 echo "Number found";

}

?>
```

Regular Expression Metacharacters

Metacharacters have **special meanings** in RegEx.

Metacharacter	Description
|	Matches one of multiple patterns
.	Matches any single character
^	Matches beginning of string
$	Matches end of string
\d	Matches any digit
\D	Matches non-digit
\s	Matches whitespace
\S	Matches non-whitespace
\w	Matches letters and digits
\W	Matches non-letter and non-digit
\b	Matches word boundary
\uxxxx	Matches Unicode character

Examples

Check digits:

```
<?php

$text = "Order123";

preg_match("/\d+/", $text, $match);

print_r($match);

?>
```

Check start of string:

```
<?php

$text = "Hello World";

preg_match("/^Hello/", $text);

?>
```

Check end of string:

```
<?php

$text = "Welcome PHP";

preg_match("/PHP$/", $text);

?>
```

Uses of PHP Regular Expressions

- Form validation (email, password, phone number)
- Searching text in strings
- Replacing specific words
- Data filtering and security
- Pattern matching