HYPERTEXT PREPROCESSOR(PHP)

UNIT 2

Numeric Arrays:

Explanation: Numeric arrays are indexed arrays where each element is accessed by a numeric index, starting from zero. These arrays are useful for storing lists of items or sequential data.

Real-life Example: Imagine you're managing a library. Each book in the library can be represented by its position on a shelf. The first book placed on the shelf corresponds to index 0, the second book to index 1, and so on.

PHP Syntax:

$books = array("Harry Potter", "Lord of the Rings", "The Hunger Games");

echo $books[0]; // Outputs: Harry Potter

Associative Arrays:

Explanation: Associative arrays are arrays where each element is associated with a specific key or name. These arrays are helpful for storing data with key-value pairs, allowing easy access to values using descriptive keys.

Real-life Example: Consider a student database where each student's information is stored. Instead of accessing a student's information by their position in the database, you access it using their unique student ID or name.

PHP Syntax:

```
$studentInfo = array(
    "John" => array("age" => 20, "grade" => "A"),
    "Jane" => array("age" => 22, "grade" => "B")
);
echo $studentInfo["John"]["age"]; // Outputs: 20
```

Multidimensional Arrays:

Explanation: Multidimensional arrays are arrays containing one or more arrays as elements. These arrays are used to represent complex data structures with multiple levels of hierarchy.

Real-life Example: Think of a movie database where each movie has various attributes such as title, director, and release year. Additionally, each movie can have multiple actors. Here, you can use a multidimensional array to represent movies and their cast.

PHP Syntax

```
$movies = array(
   array("title" => "Inception", "director" => "Christopher Nolan", "actors" => array("Leonardo DiCaprio", "Joseph Gordon-Levitt")),
   array("title" => "The Matrix", "director" => "Lana Wachowski", "actors" => array("Keanu Reeves", "Laurence Fishburne"))
);
echo $movies[0]["actors"][0]; // Outputs: Leonardo DiCaprio
```

Defining a Function:

Explanation: In PHP, a function is a block of reusable code that performs a specific task. It is defined using the function keyword followed by the function name and a pair of parentheses containing optional parameters. The function body is enclosed within curly braces {}.

Real-life Analogy: Think of a recipe in a cookbook. The recipe contains a set of instructions (function body) to prepare a dish. The recipe name (function name) identifies the dish, and optional ingredients (parameters) can customize the dish according to taste.

PHP Syntax:

```
function greet($name) {
   echo "Hello, $name!";
}
```

Calling a Function:

Explanation: After defining a function, you can execute it by calling its name followed by parentheses. If the function has parameters, you pass values within the parentheses**.**

Real-life Analogy: Consider a phone. To make a call, you dial the phone number (function name) and press the call button. If you want to talk to someone specific (function with parameters), you input their number before making the call.

PHP Syntax:

```
greet("John");
// Outputs: Hello, John!
```

Parameter Passing:

Explanation: Parameters are variables used to pass data into a function. They allow functions to accept input and perform actions based on that input.

Real-life Analogy: When ordering a pizza, you specify the toppings you want. These toppings act as parameters that customize your pizza order. Similarly, function parameters customize the behavior of a function.

PHP Syntax:

```
function add($num1, $num2) {
   return $num1 + $num2;
}
```

Returning a Value from Function:

Explanation: Functions can return values using the return statement. The returned value can be assigned to a variable or used directly in code.

Real-life Analogy: Imagine a vending machine. When you insert money and select an item, the machine dispenses the item to you. In this scenario, the item dispensed is like the value returned by a function.

PHP Syntax:

```
function add($num1, $num2) {
   return $num1 + $num2;
}
$result = add(3, 4); // $result is now 7
```

Creating and Accessing Strings:

Explanation: Strings are sequences of characters enclosed within single quotes ('') or double quotes (""). You can create strings directly or by assigning them to variables. Accessing characters within a string is done using square brackets [] with the index of the character you want to access.

Real-life Analogy: Think of a string as a sentence written on a piece of paper. Each character in the sentence has a specific position, just like how characters in a string are indexed.

PHP Syntax:

```
$str1 = "Hello, world!";
$str2 = 'PHP is awesome!';
```

echo $str1[0]; // Outputs: H

String Related Library Functions:

Explanation: PHP provides a wide range of functions to manipulate strings, such as getting the length, converting case, trimming whitespace, etc.

Real-life Analogy: Imagine you have a toolbox full of tools for various tasks. Each tool serves a specific purpose, like PHP string functions do for manipulating strings.

PHP Syntax:

$str = " Hello, World! ";

echo strlen($str); // Outputs: 17

echo strtoupper($str); // Outputs: HELLO, WORLD!

echo trim($str); // Outputs: Hello, World!

Searching:

Explanation: Searching within strings involves finding specific substrings or characters within a given string. PHP offers functions like strpos() to find the position of the first occurrence of a substring within another string.

Real-life Analogy: Searching in a library catalog for a specific book title is akin to searching for a substring within a string.

PHP Syntax:

$str = "Hello, world!";

echo strpos($str, "world"); // Outputs: 7

Replacing:

Explanation: Replacing involves substituting one substring with another within a string. PHP provides functions like str_replace() to perform string replacement.

Real-life Analogy: Imagine editing a document and replacing certain words or phrases with others. This is similar to replacing substrings within a string in PHP.

PHP Syntax:

$str = "Hello, world!";

echo str_replace("world", "PHP", $str); // Outputs: Hello, PHP!

Formatting:

Explanation: String formatting involves presenting strings in a specific format, such as date formatting or number formatting. PHP provides functions like sprintf() for string formatting.

Real-life Analogy: Formatting a document to follow specific guidelines, such as setting margins, font size, etc., is akin to formatting strings in PHP.

PHP Syntax:

```
$name = "John";

$age = 25;

$formattedString = sprintf("My name is %s and I am %d years old.", $name, $age);

echo $formattedString; // Outputs: My name is John and I am 25 years old.
```

Pattern Matching:

Explanation: Pattern matching involves searching for specific patterns or regular expressions within a string. PHP provides functions like preg_match() for pattern matching.

Real-life Analogy: Think of searching for a particular pattern in a textile or fabric. You look for specific designs or sequences, similar to pattern matching in strings.

PHP Syntax:

```
$str = "Hello, world!";

if (preg_match("/world/", $str)) {
   echo "Pattern found!";
} else {
   echo "Pattern not found!";
}
```