Tishk International University
Faculty of Applied Science
Information Technology Department

Control Structures & User-defined Functions

Web Programming

Lecture 5

Fall 2025

Course Code: IT349

Grade 3

Islam Abdulazeez
islam.abdulaziz@tiu.edu.iq

October 29, 2025

Outlines

- Control Structures
- Selection Statements
- Iteration (Looping)
- Arrays in PHP
- Array Functions in PHP
- User-defined Functions

Learning Outcomes

- At the end of today's session, you will be able to:

  - ✓ **Explain control structures and functions in PHP.**
  - ✓ **Use selection and looping statements.**
  - ✓ **Manipulate arrays with PHP functions.**
  - ✓ **Create user-defined functions.**

Control Structures

- Control Structures are programming constructs that determine the flow of execution of instructions in a program.
- They allow the program to make **decisions**, **repeat actions**, and control which statements are executed based on certain conditions.
- Generally, a program is executed sequentially, **line by line**, and a control structure allows you to alter that flow, usually depending on certain conditions.

Control Structures

1. **Sequential:** Executes statements in order, one after another.

```php
<?php
$name = "Renas";
$age = 21;
echo "Name: " . $name . "<br>";
echo "Age: " . $age . "<br>";
echo "Welcome to PHP programming!";
?>
```

Control Structures

2. **Selection (Decision):** Allows a program to choose different actions based on whether a condition is true or false.

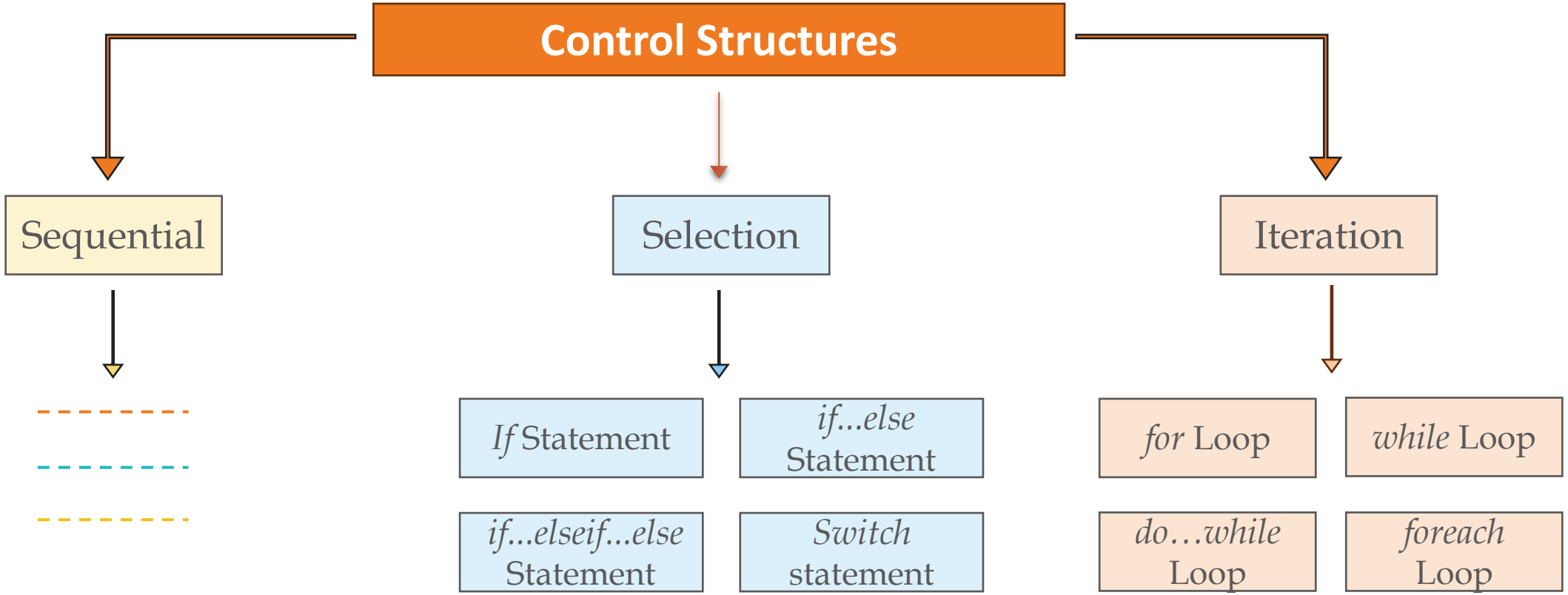
- ✓ ***if*** **Statement**
- ✓ ***if...else*** **Statement**
- ✓ ***if...elseif...else*** **Statement**
- ✓ ***switch*** **Statement**

Control Structures

3. **Iteration (Looping):** Repeats a block of code.

- ✓ **for Loop**
- ✓ ***while Loop***
- ✓ **do...while Loop**
- ✓ **foreach Loop**

Control Structures



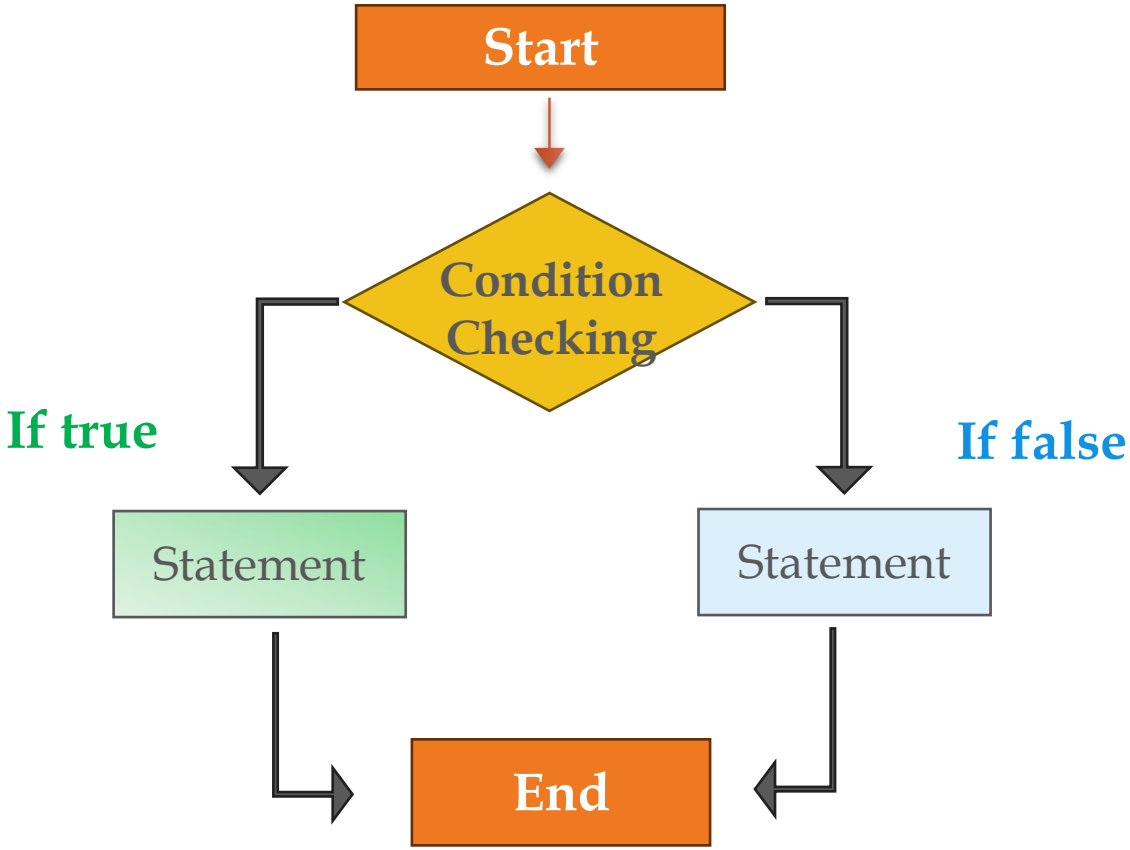
if Statement

An if statement is a control statement that allows a program to execute a block of code only if a specified condition is true.

- The statement can be a single statement or a compound statement.
- A compound statement consists of multiple statements enclosed by curly brackets.

```php
<?php
    if (condition){
     //code will be executed if specified condition is true
    }
?>
```

if Statement Flowchart



Example

```php
<?php
$y = 10;
if ($y > 0) {
    echo "Y is positive";
}
?>
```

// Y is positive

if…else Statement

An if…else statement is a control statement that chooses between two paths of execution:

```php
<?php
if (condition) {
    // code if condition is true
} else {
    // code if condition is false
}
?>
```

Example

```php
<?php
    $x = -5;
    if($x > 0) {
        echo "x is positive";
    } else {
        echo "x is negative";
    }
?>
```

// x is negative

if…else if…else Statement

An if…else if…else statement allows a program to choose between multiple conditions:

```php
<?php
  if (condition1) {
    // code if condition1 is true
  } elseif (condition2) {
    // code if condition2 is true
  } else {
    // code if none of the above conditions are true
  }
?>
```

Example

```php
<?php
  $marks = 75;

  if ($marks >= 90) {
    echo "Grade A";
  } elseif ($marks >= 75) {
    echo "Grade B";
  } elseif ($marks >= 50) {
    echo "Grade C";
  } else {
    echo "Fail";
  }
?>
```

// Grade B

switch Statement

A switch statement is a control statement used to select one of many blocks of code to execute based on the value of a variable.

```php
<?php
  switch (variable) {
    case value1:
        // code to execute if variable == value1
        break;
    case value2:
        // code to execute if variable == value2
        break;
    default:
        // code to execute if variable doesn't match any case
}
?>
```

Example

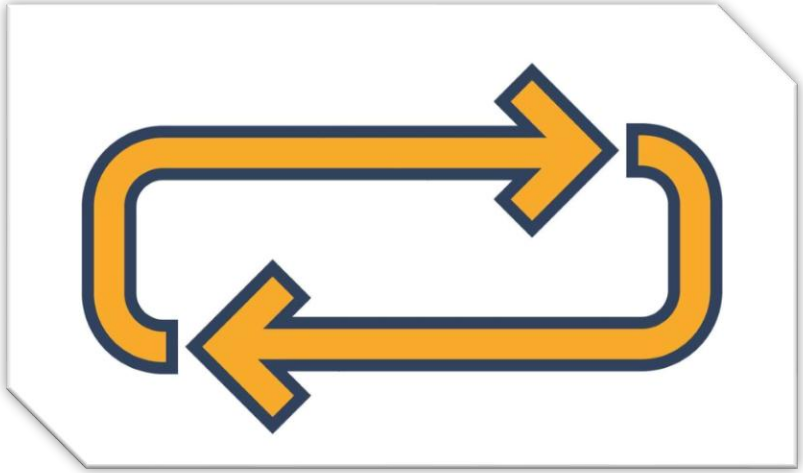
```php
<?php
    $day = "Tuesday";

    switch($day) {
        case "Monday":
            echo "Today is Monday";
            break;
        case "Tuesday":
            echo "Today is Tuesday";
            break;
        default:
            echo "Today is not Monday or Tuesday";
    }
?>
```

// Today is Tuesday

Loops

- In programming, loops allow you to execute a block of code multiple times.
- PHP provides several types of loops, each suited for different use cases.
- Loops are especially useful when working with arrays, databases, and other situations where repetitive tasks are common.



Types of Loops in PHP

PHP supports the following loop structures:

- **for Loop** - Repeats a block of code a specified number of times.
- **while Loop** - Repeats a block of code as long as a condition is true.
- **do...while Loop** - Similar to while, but guarantees the code runs at least once.
- **foreach Loop** - Specially designed for iterating over arrays.

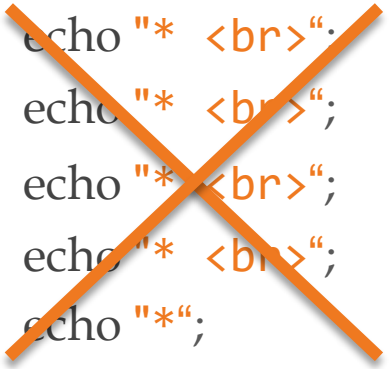
for Loop

The for loop is ideal when you know beforehand how many times you need to iterate. It consists of three parts: **initialization**, **condition**, and **increment/decrement**.

- **Why Use a For Loop?**

  It helps reduce code complexity and improves readability.

```
for (.....) {
        echo "* <br>";
}
```



Example

```php
<?php
    for ($i = 0; $i < 5; $i++) {
    echo "Iteration: $i <br>";
    }
?>
```

```
/*
    Iteration: 0
    Iteration: 1
    Iteration: 2
    Iteration: 3
    Iteration: 4
*/
```

while Loop

The while loop is useful when the number of iterations is unknown and depends on a condition. It checks the condition before each iteration.

```php
<?php
    $count = 1;
    while($count <= 5){
        echo "Count: $count <br>";
        $count++;
    }
?>
```

```
/*
    Count: 1
    Count: 2
    Count: 3
    Count: 4
    Count: 5
*/
```

do…while loop

The do...while loop is similar to the while loop, but it executes the code block

at least once, regardless of the condition. The condition is checked after the loop executes.

```php
<?php
    $count = 1;
    do{
        echo "Count: $count <br>";
        $count++;
    } while($count >= 5);
?>
```

// Count: 1

Arrays

Arrays are one of the most fundamental data structures in PHP, allowing you to store multiple values in a single variable.

In PHP, arrays are flexible, allowing a **mix of data types** and associative key value pairs.

Types of Arrays in PHP

- **Indexed Arrays:** Arrays with numeric indices.
- **Associative Arrays:** Arrays with named keys.
- **Multidimensional Arrays**: Arrays containing one or more arrays as elements.

Indexed Arrays

- An indexed array uses numeric indices to store values. PHP automatically assigns indices starting from 0

```php
<?php
    $myArray1 = array("value1", "value2", "value3");
    $myArray2 = ["value1", "value2", "value3"];
?>
```

Indexed Arrays

- PHP automatically assigns the next available numeric index (starting from 0).

```php
<?php
    $fruits = array();

    $fruits[] = "Apple";
    $fruits[] = "Banana";
    $fruits[] = "Orange";
    $fruits[] = "Pineapple";
?>
```

Indexed Arrays

- Or the index can be assigned manually

```php
<?php
    $fruits = array();

    $fruits[2] = "Apple";
    $fruits[4] = "Banana";
    $fruits[5] = "Orange";
    $fruits[7] = "Pineapple";
?>
```

Indexed Arrays

- Printing array values

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    echo "I like " . $cars[1] . " more than " . $cars[2] . ".";

?>
```

// I like BMW more than Toyota.

```php
<?php
    $fruits = array("Apple", "Banana", "Orange");

    echo $fruits[0] . "<br>";
    echo $fruits[1] . "<br>";
    echo $fruits[2];
?>
```

```
/*
    Apple
    Banana
    Orange
*/
```

foreach Loop

The foreach loop is specially designed to iterate over arrays, making it the most efficient choice for array manipulation.

```php
<?php
    $fruits = array("Apple", "Banana", "Orange");

    foreach ($fruits as $fruit) {
        echo "$fruit <br>";
    }
?>
```

```
/* Apple
   Banana
   Orange  */
```

Viewing Array Structure and Values

- You can see the structure and values of any array by using one of two statements var_dump() or print_r(). The print_r() statement, however, gives somewhat less information.

- Use print_r() when you just want to indexes (or keys) and their corresponding values.
- Use var_dump() when you need full detail (type, size, structure).

Viewing Array Structure and Values

```php
<?php
    $fruits = array("Apple", "Banana", "Orange");
    print_r($fruits);
?>
```

// Array ([0] => Apple [1] => Banana [2] => Orange)

```php
<?php
    $fruits = array("Apple", "Banana", "Orange");
    var_dump($fruits);
?>
```

// array(3) { [0]=> string(5) "Apple" [1]=> string(6) "Banana" [2]=> string(6) "Orange" }

Loop Through an Indexed Array

- To loop through and print all the values of an indexed array, you could use a for loop, like this:

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    for($i = 0; $i < 3; $i++) {
    echo $cars[$i];
    echo "<br>";
}
?>
```

```
/* Volvo
   BMW
   Toyota  */
```

Associative Arrays

- Associative arrays use **named keys** that you assign to each value, making it easier to access data by name rather than by numeric index.

```
$testArray = array("key1" => "value1", "key2" => "value2");
```

```php
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    // Second way
    $age["Peter"] = "35";
    $age["Ben"] = "37";
    $age["Joe"] = "43";
?>
```

Associative Arrays

- Example

```php
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    echo "Ben is " . $age['Ben'] . " years old.";
?>
```

// Ben is 37 years old.

Loop Through an Associative Array

- To loop through and print all the values of an associative array, you could use a foreach loop, like this:

```php
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    foreach($age as $value) {
        echo "Age: " . $value;
        echo "<br>";
    }
?>
```

```
/*
Age: 35
Age: 37
Age: 43
*/
```

Loop Through an Associative Array

- Printing the keys and values.

```php
<?php
    $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
    foreach($age as $key => $value){
        echo "Key=" . $key . ", Value=" . $value;
        echo "<br>";

    }
?>
```

```
/*
Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43
*/
```

Multidimensional Arrays

- A multidimensional array is an array that contains one or more arrays inside it.
- PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

Multidimensional Arrays

- Example

```php
<?php
    $students = array(
    array("Ali", 20, "Computer Science"),
    array("Sana", 22, "Information Technology"),
    array("Omar", 21, "Software Engineering")
);

echo $students[0][0];
echo $students[1][2];
?>
```

// Ali
// Information Technology

Multidimensional Arrays

```php
<?php
$students = array(
    array("Ali", 20, "Computer Science"),
    array("Sana", 22, "Information Technology"),
    array("Omar", 21, "Software Engineering")
);

for ($i = 0; $i < 3; $i++) {
    for ($j = 0; $j < 3; $j++) {
        echo $students[$i][$j] . " ";
    }
    echo "<br>";
}
?>
```

```
/*
Ali 20 Computer Science
Sana 22 Information Technology
Omar 21 Software Engineering
*/
```

Array Functions in PHP

- The count() function is used to return the length (the number of elements) of an array:

```php
<?php
    $numbers = [1,2,3,4,5,6,7];
    echo count($numbers);
?>
```

// 7

Array Functions in PHP

- in_array() - Searches an array for a specific value.

```php
<?php
    $fruits = ["apple", "banana", "cherry", "orange"];
    if (in_array("orange", $fruits)) {
        echo "Orange is in the array";
    } else {
        echo "Orange is not in the array";
    }
?>
```

// Orange is in the array

Array Functions in PHP

- shuffle() - The shuffle() function randomizes the order of the elements in the array.

```php
<?php
    $fruits = ["apple", "banana", "cherry", "orange"];
    shuffle($fruits);
?>
```

Array Functions in PHP

- The shuffle() function assigns new indices for the elements in the array. Existing indices will be removed (See Example below).

```php
<?php
    $fruits = array("Apple", "Banana", "Orange", "Mango");
    print_r($fruits);
    echo "<br>";
    shuffle($fruits);
    print_r($fruits);
?>
```

// Array ([0] => Apple [1] => Banana [2] => Orange [3] => Mango)

// Array ([0] => Orange [1] => Banana [2] => Mango [3] => Apple)

Array Functions in PHP

- implode() - The implode() function returns a string from the elements of an array.

```php
<?php
    $fruits = array("Apple", "Banana", "Orange", "Pineapple");
    $text = implode("_", $fruits);
    echo $text;
?>
```

// Apple_Banana_Orange_Pineapple

Array Functions in PHP

- explode() - The explode() function breaks a string into an array.

```php
<?php
    $names = "Ali Aram Kani Kurdistan Milan Saween";
    $myArray = explode(" ", $names);
    echo $myArray[0];
    echo "<br>";
    echo $myArray[3];
?>
```

// Ali
// Kurdistan

Array Functions in PHP

- array_merge(): Allows you to append one array into another.

  Think of it as concatenation for arrays.

```php
<?php
    $names1 = array("Kardo", "Zara" ,"Azad");
    $names2 = array("Hardi", "Sava", "Bestun");
    $names = array_merge($names1, $names2);
    print_r($names);
?>
```

// Array ([0] => Kardo [1] => Zara [2] => Azad [3] => Hardi [4] => Sava [5] => Bestun)

Array Functions in PHP

- Example

```php
<?php
    $names1 = array("Kardo", "Zara" ,"Azad");
    $names2 = array("Hardi", "Sava", "Bestun");
    $names = array_merge($names1, $names2);
    echo $names[3];
?>
```

// Hardi

Deleting Array Elements

- unset() – The unset() function is used to destroy a variable or an element of an array.

```php
<?php
    $names = array("Kardo", "Zara" ,"Azad", "Renas", "Sava");
    unset($names[2]);
    print_r($names);
?>
```

// Array ([0] => Kardo [1] => Zara [3] => Renas [4] => Sava)

Deleting Array Elements

- If you see the above example carefully you will find that the unset() function didn't reindex the array after deleting the value from the indexed array. To fix this you can use the array_splice() function.

array_splice()

Deleting Array Elements

- It takes three parameters: an array, offset (where to start), and length (number of elements to be removed). Let's see how it actually works:
- Example

```php
<?php
    $names = array("Kardo", "Zara" ,"Azad", "Renas", "Sava");
    array_splice($names, 3, 1);
    print_r($names);
?>
```

// Array ([0] => Kardo [1] => Zara [2] => Azad [3] => Sava)

PHP Sorting Arrays

- PHP comes with a number of built-in functions designed specifically for sorting array elements in different ways like alphabetically or numerically in ascending or descending order. Here we'll explore some of these functions most commonly used for sorting arrays.

- sort() and **r**sort() — For sorting **indexed** arrays.
- **a**sort() and **ar**sort() — For sorting **associative** arrays by **value**.
- **k**sort() and **kr**sort() — For sorting **associative** arrays by **key**.

r = **reverse**
a = **associative**
k = **key**

sort() and rsort()

- Examples

```php
<?php
    $names = array("Kardo", "Zara" ,"Azad", "Renas", "Sava");
    sort($names);
    print_r($names);
?>
```

// Array ([0] => Azad [1] => Kardo [2] => Renas [3] => Sava [4] => Zara)

```php
<?php
    $numbers = array(5, 2, 1, 6, 3);
    sort($numbers);
    print_r($numbers);
?>
```

// Array ([0] => 1 [1] => 2 [2] => 3 [3] => 5 [4] => 6)

sort() and rsort()

- Examples

```php
<?php
    $names = array("Kardo", "Zara" ,"Azad", "Renas", "Sava");
    rsort($names);
    print_r($names);
?>
```

// Array ([0] => Zara [1] => Sava [2] => Renas [3] => Kardo [4] => Azad)

```php
<?php
    $numbers = array(5, 2, 1, 6, 3);
    rsort($numbers);
    print_r($numbers);
?>
```

// Array ([0] => 6 [1] => 5 [2] => 3 [3] => 2 [4] => 1)

asort() and ksort()

- Examples

```php
<?php
    $ages = array("Zana" => 23, "Aram" => 22, "Kani" => 20);
    asort($ages);
    print_r($ages);
?>
```

// Array ([Kani] => 20 [Aram] => 22 [Zana] => 23)

```php
<?php
    $ages = array("Zana" => 23, "Aram" => 22, "Kani" => 20);
    ksort($ages);
    print_r($ages);
?>
```

// Array ([Aram] => 22 [Kani] => 20 [Zana] => 23)

User-defined Functions

- A user-defined function in PHP is a function that you create yourself to perform a specific task

```php
<?php

    function functionName(){
        echo "This is a function";
    }
    functionName();
?>
```

// This is a function

```php
<?php

    function square($number) {
    $result = $number * $number;
    echo "The square of $number is $result";
}
    square(5);

?>
```

// The square of 5 is 25

User-defined Functions

- More Examples

```php
<?php

    function checkAge($age) {
    if ($age >= 18) {
        echo "You are an adult.";
    } else {
        echo "You are a minor.";
    }
}
    checkAge(20);

?>
```

// You are an adult.

```php
<?php

    function printNumbers($n) {
     for ($i = 1; $i <= $n; $i++) {
         echo $i . " ";
     }
}
printNumbers(5);

?>
```

// 1 2 3 4 5

Lab Assesments and Next Session's Topic

Lab Assessments

- Lab Exercises

Next Session's Topic

- Date, Time & Form Data Handling

References

- Tatroe, K., & MacIntyre, P. (2020). Programming PHP: Creating dynamic web pages (4th ed.). O'Reilly Media.
- Ullman, L. (2016). PHP for the web: Visual QuickStart guide (5th ed.). Peachpit Press.
- PHP Group. (n.d.). PHP: Hypertext Preprocessor — official documentation. Retrieved October 19, 2025, from https://www.php.net

Thank You!