PHP Syntax

Overview

PHP scripts are embedded within HTML using `<?php ... ?>` tags. PHP code is executed on the server, and the result is sent to the client as plain HTML.

Standard Rules

- Always use `<?php` to start PHP code and `?>` to end it.
- PHP statements must end with a semicolon (`;`).
- PHP is case-sensitive for variables, but not for keywords (e.g., `echo`, `if`).

Edge Cases

- If a PHP file contains only PHP code, omit the closing `?>` to avoid unintended whitespace or newline characters.
- Mixing PHP and HTML requires careful attention to avoid syntax errors.

Example: Basic PHP Syntax

```
<?php
echo "Hello, World!"; // Correct
ECHO "Hello, World!"; // Also correct (keywords are case-insensitive)
?>
```

Variables

Overview

Variables in PHP are used to store data. They start with a `$` sign, followed by the variable name.

Standard Rules

- Variable names must start with a letter or underscore (`_`).
- Variable names cannot start with a number.
- Variable names can only contain alphanumeric characters and underscores.
- Variable names are case-sensitive.

Edge Cases

- Avoid using reserved keywords (e.g., `echo`, `if`) as variable names.
- Uninitialized variables will trigger a notice if accessed.

Example: Variables

```
<?php
$name = "John Doe"; // Valid
$_age = 25; // Valid
// $1name = "John"; // Invalid (starts with a number)
// $my-name = "John"; // Invalid (contains a hyphen)
?>
```

Data Types

Overview

PHP supports several data types, including:

1. **String**: Text data (e.g., `"Hello"`).
2. **Integer**: Whole numbers (e.g., `25`).

3. **Float**: Decimal numbers (e.g., `3.14`).
4. **Boolean**: `true` or `false`.
5. **Array**: Collection of values.
6. **Object**: Instances of classes.
7. **NULL**: Represents a variable with no value.

Standard Rules

- Use `var_dump()` to inspect the type and value of a variable.
- PHP automatically converts types in certain contexts (e.g., adding a string and an integer).

Edge Cases

- Be cautious with type juggling (automatic type conversion) to avoid unexpected behavior.
- Use `===` for strict comparison (checks both value and type).

Example: Data Types

```
<?php
$string = "Hello";
$integer = 25;
$float = 3.14;
$boolean = true;
$array = array("apple", "banana", "cherry");
$null = null;

var_dump($string); // Output: string(5) "Hello"
var_dump($integer + $string); // Output: int(25) (PHP converts "Hello" to 0)
?>
```

Strings

Overview

Strings are sequences of characters. PHP provides many functions to manipulate strings.

Common String Functions

- `strlen()`: Returns the length of a string.
- `strpos()`: Finds the position of a substring.
- `str_replace()`: Replaces text within a string.
- `substr()`: Returns a portion of a string.

Standard Rules

- Use single quotes (`'`) for simple strings and double quotes (`"`) for strings with variables or escape sequences.
- Concatenate strings using the `.` operator.

Edge Cases

- Be cautious with escape sequences in double-quoted strings (e.g., `\n`, `\t`).
- Use `htmlspecialchars()` to prevent XSS attacks when displaying user input.

Example: String Manipulation

```php
<?php
$text = "Hello, World!";
echo strlen($text); // Output: 13
echo strpos($text, "World"); // Output: 7
echo str_replace("World", "PHP", $text); // Output: Hello, PHP!
echo substr($text, 0, 5); // Output: Hello

// Edge Case: Escaping quotes
echo "He said, \"Hello!\""; // Output: He said, "Hello!"
?>
```

Constants

Overview

Constants are like variables but cannot be changed once defined. They are defined using the `define()` function.

Standard Rules

- Constants are case-sensitive by default. Use `true` as the third argument in `define()` to make them case-insensitive.
- Constants are global and can be accessed anywhere in the script.

Edge Cases

- Avoid redefining constants, as it will trigger a warning.

Example: Constants

```php
<?php
define("PI", 3.14);
echo PI; // Output: 3.14

// Edge Case: Case-insensitive constant
define("GREETING", "Hello", true);
echo GREETING; // Output: Hello
echo greeting; // Output: Hello (case-insensitive)
?>
```

Operators

Overview

Operators are used to perform operations on variables and values.

Types of Operators

1. **Arithmetic Operators**: `+`, `-`, `*`, `/`, `%`.
2. **Assignment Operators**: `=`, `+=`, `-=`, `*=`, `/=`.
3. **Comparison Operators**: `==`, `!=`, `>`, `<`, `>=`, `<=`.
4. **Logical Operators**: `&&`, `||`, `!`.

Standard Rules

- Use `==` for loose comparison (checks value only).
- Use `===` for strict comparison (checks both value and type).

Edge Cases

- Be cautious with `==` as it may produce unexpected results due to type juggling.

- Use `&&` and `||` instead of `and` and `or` for logical operations, as they have higher precedence.

Example: Operators

```
<?php
$a = 10;
$b = "10";

echo $a == $b; // Output: true (loose comparison)
echo $a === $b; // Output: false (strict comparison)

// Edge Case: Logical operator precedence
$result = ($a == 10 && $b == "10"); // Correct
$result = ($a == 10 and $b == "10"); // Avoid (lower precedence)
?>
```

Control Structures

Overview

Control structures are used to control the flow of execution in a PHP script.

Types of Control Structures

1. **Conditional Statements**:
   - `if`, `else`, `elseif`.
   - `switch`.
2. **Loops**:
   - `for`, `while`, `do-while`.
   - `foreach`.

Standard Rules

- Always use curly braces `{}` for blocks of code, even if they contain only one statement.
- Use `break` in `switch` statements to prevent fall-through.

Edge Cases

- Be cautious with infinite loops (e.g., `while (true)`).
- Use `elseif` instead of `else if` for consistency.

Example: Control Structures

```
<?php
$age = 18;

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

// Edge Case: Infinite loop
// while (true) {
//     echo "This will run forever!";
// }
?>
```

Example: Loops

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

$colors = array("red", "green", "blue");
foreach ($colors as $color) {
    echo $color . "<br>";
}
?>
```

Functions

Overview

Functions are blocks of code that perform a specific task. They help in organizing code and reusing logic.

Defining and Calling Functions

- Use the `function` keyword to define a function.
- Call a function by using its name followed by parentheses.

Standard Rules

- Use descriptive names for functions.
- Avoid using global variables inside functions; pass them as parameters instead.

Edge Cases

- Be cautious with recursive functions to avoid stack overflow.
- Use `return` to exit a function early if needed.

Example: Functions

```php
<?php
function greet($name) {
    return "Hello, $name!";
}

echo greet("John"); // Output: Hello, John!

// Edge Case: Recursive function
function factorial($n) {
    if ($n <= 1) return 1;
    return $n * factorial($n - 1);
}
echo factorial(5); // Output: 120
?>
```

Arrays

Overview

Arrays are used to store multiple values in a single variable.

Types of Arrays

1. **Indexed Arrays**: Arrays with numeric indexes.
2. **Associative Arrays**: Arrays with named keys.

3. **Multidimensional Arrays**: Arrays containing one or more arrays.

Standard Rules

- Use `array()` or `[]` to create arrays.
- Use `foreach` to iterate over arrays.

Edge Cases

- Be cautious with associative arrays when using numeric keys, as they may behave unexpectedly.
- Use `array_key_exists()` to check if a key exists in an array.

Example: Arrays

```
<?php
// Indexed Array
$fruits = array("apple", "banana", "cherry");
echo $fruits[0]; // Output: apple

// Edge Case: Associative array with numeric keys
$person = array(1 => "John", 2 => "Doe");
echo $person[1]; // Output: John
?>
```

Summary of Key Points

TopicDescriptionPHP Syntax

- Use `<?php ... ?>` and end statements with `;`.
- **Variables** Start with `$`, follow naming rules, and avoid reserved keywords.
- **Data Types** Use `var_dump()` to inspect types and be cautious with type juggling.
- **Strings** Use single/double quotes, concatenate with `.`, and escape special characters.
- **Constants** Use `define()` and avoid redefining constants.
- **Operators** Use `===` for strict comparison and `&&`/`||` for logical operations.
- **Control Structures** Use `{}` for blocks, `break` in `switch`, and avoid infinite loops.
- **Functions** Use descriptive names, avoid globals, and be cautious with recursion.
- **Arrays** Use `array()` or `[]`, iterate with `foreach`, and check keys with `array_key_exists()`.

Practical Questions

1. Create a PHP script to display your name and age using variables. Handle the case where the age is not set.
2. Write a function to calculate the area of a rectangle and call it. Handle invalid inputs (e.g., negative values).
3. Use a `foreach` loop to display all elements of an indexed array. Handle empty arrays gracefully.
4. Create an associative array to store student details (name, age, grade) and display them. Handle missing keys.
5. Use an `if-else` statement to check if a number is even or odd. Handle non-numeric inputs.