Answers

1. **Original PHP Acronym:** It originally stood for **Personal Home Page**.
2. **Creator: Rasmus Lerdorf** created the first iteration in 1994.
3. **Current Recursive Definition:** PHP now stands for **PHP: Hypertext Preprocessor**.
4. **PHP 3.0 Rewriters:** The parser was rewritten by **Andi Gutmans** and **Zeev Suraski**.
5. **Core Scripting Engine (2000):** The **Zend Engine**.
6. **Full OOP Support:** Full support for Object-Oriented Programming was introduced in **PHP 5**.
7. **File Extension:** The standard file extension is **`.php`**.
8. **Data Type:** This range represents the **Integer** data type.
9. **Array Definition:** An array is a data structure variable capable of storing multiple values under a single name.
10. **Three Array Types: Indexed** arrays, **Associative** arrays, and **Multidimensional** arrays.
11. **Concatenation Operator:** The `.` (dot) operator.
12. **Syntax Tags:** Scripts open with **`<?php`** and close with **`?>`**.
13. **Jumping Control Statements:** **`break`**, **`continue`**, and **`return`** (or `goto`).
14. **Delete File Function:** The **`unlink()`** function.
15. **File Upload Superglobal:** **`$_FILES`**.
16. **XAMPP "A":** The "A" stands for **Apache** (the web server).
17. **New Object Keyword:** The **`new`** keyword.

18. **Static vs. Dynamic Web Server:** A static web server sends pre-existing, unchanging files (like HTML/CSS) directly to the client's browser exactly as they are stored. A dynamic server runs a processing engine (like PHP) to execute scripts, interact with databases, and build a custom HTML page on the fly before sending it to the browser.
19. **`==` vs `===`:** The `==` operator checks if values are equal after attempting to convert them to the same data type. The `===` operator strictly checks if both the value *and* the data type are entirely identical.
20. **Constructor Function:** A magic method (usually `__construct()`) that is automatically triggered the moment a new object is created, primarily used to set up initial property values.
21. **Indexed vs. Associative:** Indexed arrays assign elements to numeric keys (0, 1, 2). Associative arrays assign elements to named string keys (e.g., "username" => "admin").
22. **Break vs. Continue:** `break` completely terminates the loop and moves to the code below it. `continue` only stops the current iteration and jumps back to the top to evaluate the next loop iteration.
23. **Try-Catch Purpose:** To safely handle unpredictable runtime errors (exceptions) so the script can execute alternative error-handling logic instead of crashing the entire application.
24. **Error vs. Exception:** Errors are typically fatal flaws (like syntax mistakes) that halt execution and cannot be caught. Exceptions are manageable runtime anomalies that can be caught and resolved programmatically.
25. **Session vs. Cookies:** Session data is securely stored on the server side. Cookie data is stored locally on the user's browser, making it less secure.
26. **`fopen()` in "w" mode:** It opens a file for writing. If the file already exists, it completely erases (truncates) the current content. If it doesn't exist, it creates a new file.
27. **Zend Engine Role:** It acts as the core compiler and execution environment that translates human-readable PHP code into bytecode and runs it.

28. **Benefits of Comments:** They improve readability, explain complex logic for future developers, and allow you to temporarily disable code blocks while debugging.
29. **Exit Statement:** When triggered, it immediately terminates the execution of the entire PHP script; nothing further is processed.
30. **Logical Error:** A flaw in the actual thought process or math of the code (e.g., adding instead of subtracting). The system produces no warning because the code is syntactically valid; it just produces the wrong outcome.
31. **File Downloading Concept:** The web server sends specific HTTP headers (like `Content-Disposition: attachment`) to the browser, instructing it to save the incoming data stream as a local file instead of trying to render it on the screen.
32. **Encapsulation:** The practice of hiding the internal state of an object and requiring all interaction to be performed through an object's methods, bundling data and behavior together.
33. **Accessibility:** `public` properties are accessible from anywhere. `protected` properties are accessible within the defining class and its children. `private` properties are strictly accessible only within the class that defines them.
34. **`md5()` for Visitor Counter:** It creates a unique, irreversible hash of a visitor's IP address. This allows the developer to recognize a returning user without storing highly sensitive, identifiable IP data in plain text.

35. **Smart Ride Rwanda Trip Cost:**

```
<?php
function calculateTripCost($distance, $pricePerKm)
 {
 return $distance * $pricePerKm;
}
Echo  calculateTripCost (20, 30);
?>
```

36. **Attendance System Foreach Loop:**

```
<? php

$students = ["John" => "Present", "Mary" => "Absent", "Paul" => "Present"];
foreach ($students as $name => $status) {
    echo "Student: $name - Status: $status<br>";
}
?>
```

37. **While Loop (1 to 10):**

```
<?php
$i = 1;
while ($i <= 10) { echo $i . " "; $i++; }
?>
```

38. **`str_replace()`:**

```
</php
echo str_replace("world", "Dolly", "Hello world!");
?>
```

39. **Switch Statement for Days:**

```
<?php
$day = 3;
switch ($day) {
    case 1: echo "Monday"; break;
    case 2: echo "Tuesday"; break;
    case 3: echo "Wednesday"; break;
    /* ... 4 to 6 ... */
    case 7: echo "Sunday"; break;
    default: echo "Invalid input";
}
?>
```

40. **New Generation Academy Library Array:**

```
<?php
$library = [
    ["title" => "Physics Vol 1", "author" => "Newton", "status" => "Available"],
    ["title" => "African History", "author" => "Diop", "status" => "Checked Out"],
    ["title" => "Advanced PHP", "author" => "Lerdorf", "status" => "Available"]
];
?>
```

41. **`setcookie()`:**

```
<?php
setcookie("login_pref", "stay_logged_in", time() + 3600, "/");
?>
```

42. **Pulse Logs Append:**

```
<?php

$logFile = fopen("pulse_logs.txt", "a");
fwrite($logFile, "Render completed successfully.\n");
fclose($logFile);
?>
```

43. **`move_uploaded_file()`:**

```
<?php
move_uploaded_file($_FILES["uploadFile"]["tmp_name"], "uploads/" .
$_FILES["uploadFile"]["name"]);
?>
```

44. **Grades Check:**

```
<?php
if ($marks >= 50)
{
echo "Pass";
 }
else
{
```

```
echo "Fail";
}
?>
```

45. **Pagination Calculation:**

```
<?php
$totalItems = 100; $perPage = 10;
$totalPages = ceil($totalItems / $perPage);
?>
```

46. **Database Connection:**

```
<?php
$conn = mysqli_connect("localhost", "root", "", "my_database");
?>
```

47. **Do-While Sum:**

```
<?php
$sum = 0; $i = 1;
do
{
$sum += $i; $i++;
}
while ($i <= 10);
?>
```

48. **String Length:**

```
<?php
echo strlen("Digital Solutions");
?>
```

49. **Instantiate Object:**

```
<?php
$myFruit = new Fruit();
$myFruit->name = "Mango";
?>
```

50. **Try-Catch Block:**

```
<?php
try {
    $divider = 0;
    if ($divider == 0) { throw new Exception("Division by zero error."); }
    echo 100 / $divider;
} catch (Exception $e) {
    echo "Caught Error: " . $e->getMessage();
}
?>
```

Analyzing

51. **For vs. While Loop:** A `for` loop is ideal when you know exactly how many times the loop should run (e.g., iterating through a fixed list). A `while` loop is better when the number of iterations is unknown and depends on a dynamic condition being met (e.g., reading database rows until there are none left).
52. **GET vs POST:** GET appends data directly into the URL, making it visible to anyone looking at the screen or browser history, offering zero security. POST packages data inside the HTTP request body, keeping it hidden from the URL and making it far more secure for sensitive transmissions.
53. **Class vs. Object:** A class is the abstract blueprint or template defining characteristics and behaviors. An object is a concrete, physical instance created from that blueprint.
54. **Inline Comment Analysis:** `$x = 5 /* + 15 */ + 5;` The PHP parser entirely ignores `/* + 15 */`. Therefore, it processes `$x = 5 + 5;`, making the final executed value **10**.
55. **Procedural vs OOP:** Procedural PHP relies on a top-down sequence of isolated functions and variables, which can become tangled in large projects. OOP groups related data and functions into dedicated, reusable objects, resulting in cleaner, more modular, and easier-to-maintain code architectures.
56. **`$_FILES` Role:** It captures essential metadata about the incoming file, specifically: the original `name`, the temporary server path (`tmp_name`), the file `size`, the MIME `type`, and any `error` codes generated during the upload.
57. **PHP 5 Performance:** PHP 5 introduced the Zend Engine II, which fundamentally changed how objects were handled. Objects were passed by reference rather than being duplicated by value, massively reducing memory overhead and boosting execution speed for large applications.
58. **Page Watermarking:** Watermarking dynamically intercepts a requested asset, utilizes an image processing library (like GD), stamps a transparent text/logo layer over the original asset in server memory, and serves the compounded result to the browser, ensuring the raw file is never directly downloaded.
59. **`goto` Execution Flow:** Normal execution reads line by line sequentially. The `goto` statement breaks this flow by immediately jumping the compiler to a specific labeled anchor elsewhere in the file, which can make debugging difficult.
60. **`strpos` Breakdown:** `strpos("Hello world!", "world")` returns **6** because strings are zero-indexed. Counting starts at H(0), e(1), l(2), l(3), o(4), space(5). The "w" in "world" sits exactly at index 6.
61. **`fread()` vs `file_get_contents()`:** `file_get_contents()` is structurally simpler. While `fread()` requires you to open the file stream, determine the file size, read it, and manually close it, `file_get_contents()` does all of that in a single, convenient function call.
62. **CRUD Mapping:** Create maps to **INSERT**, Read maps to **SELECT**, Update maps to **UPDATE**, and Delete maps to **DELETE**.
63. **Infinite Loop Analysis:** This occurs when the evaluating condition never becomes false during runtime. The most common logical error is forgetting to increment or decrement the counter variable inside a `while` block.
64. **XAMPP Components: X** (Cross-platform), **A** (Apache server), **M** (MariaDB/MySQL), **P** (PHP), **P** (Perl).
65. **Foreach Structural Benefits:** It is purpose-built for arrays. It automatically manages the internal array pointer, determines the array's length, and guarantees you won't trigger an "out of bounds" error.

66. **Syntax vs Runtime Error:** A syntax error is a grammatical violation (missing semicolon) that prevents the script from executing at all. A runtime error happens when syntactically perfect code encounters an impossible situation while running (like trying to open a file that was just deleted).

Evaluating

67. **Intego Corporates GET Security:** Using GET for a login form is highly risky. It places the username and password in plaintext directly into the browser's URL bar, making the credentials visible to shoulder-surfers, browser histories, and server logs.
68. **Sessions vs. Cookies for Auth:** Sessions are strictly more appropriate. Justification: Session data is stored securely on the server side, whereas cookies store data on the client device where it is highly vulnerable to interception, manipulation, and Cross-Site Scripting (XSS) attacks.
69. **Critique of `md5()` Tracking:** Its "one-way" nature means a hash cannot be reverse-engineered back into an IP address. This makes it highly effective for analytics; you can securely identify if a hash matches a previous visitor without ever violating their privacy by storing their actual IP.
70. **`unlink()` Consequences:** Without verification, `unlink()` permanently deletes the file, bypassing system recycling bins. A developer could inadvertently delete critical system configurations or a user's uploaded assets irrevocably, causing application failure or data loss.
71. **OOP for Startup Maintenance:** For a growing startup, OOP is vastly superior. Its modular nature allows multiple developers to work on different classes simultaneously without breaking each other's code. Features can be added by extending classes rather than rewriting thousands of lines of procedural logic.
72. **Necessity of `break` in Switch:** It is absolutely critical. Without it, PHP will execute the matched case *and* "fall through" to execute every single case beneath it regardless of whether they match, resulting in corrupted output.
73. **Static vs Dynamic for WordPress:** A static server is entirely inappropriate. A CMS like WordPress requires a dynamic server to execute PHP code, query a MySQL database for posts, and assemble the HTML templates dynamically. A static server cannot process this backend logic.
74. **Mathematical Logic Flaw:** `$total = $price + $discount;` is flawed because a discount represents a reduction in price. The logic adds the value, making the item more expensive. It must be `$total = $price - $discount;`.
75. **Logical Loop for Databases:** The `while` loop is the standard. Since you rarely know the exact row count beforehand, `while($row = mysqli_fetch_assoc($result))` perfectly dictates that the loop should run continuously *only* as long as there is another row to retrieve.
76. **Importance of Content-Disposition:** Without this header, a browser will attempt to open and render the file inline (e.g., displaying an image or playing an MP3 directly in the tab). The header forces the browser to prompt the user with a "Save As" dialogue.
77. **Justifying Abstraction & Encapsulation:** These principles ensure that a class's internal complex logic is hidden (abstraction) and its sensitive data cannot be modified directly from the outside (encapsulation). This protects the class's integrity and prevents other code from accidentally breaking it.
78. **If-Else vs Switch Readability:** Deeply nested if-else statements become visually cluttered and difficult to debug. A switch statement is highly streamlined, aligning conditions neatly, making it easier for a developer to read, maintain, and execute slightly faster when evaluating a single variable against many static outcomes.
79. **Bytecode Compilation Security:** Compiling to OpCache protects intellectual property by transforming human-readable PHP into machine-level instructions. While primarily designed for

speed, it prevents end-users or clients from easily reading, copying, or modifying the proprietary source code of a commercial product.

80. **Inheritance Benefits:** It ensures code reusability (DRY principle). If a parent class has database connection methods, a child class inherits them automatically. This drastically reduces redundant coding and ensures systemic consistency across the application architecture.
81. **Resource Problem of Forgotten `fclose()`:** If files are left open, it causes a "resource leak." The operating system has a limit on how many file handles can be open simultaneously; eventually, the server will hit this limit and crash when attempting subsequent file operations.
82. **PHP 8.0 JIT Evaluation:** The JIT (Just-In-Time) compiler in PHP 8 translates bytecode directly into machine code at runtime for repetitive tasks. This drastically improves execution speed for CPU-heavy processes and math calculations compared to older versions that interpreted code line-by-line.
83. **Validity of `var` keyword:** While valid for legacy PHP 4 code, evaluating properties using `var` is poor practice today. Modern PHP demands explicit visibility modifiers (`public`, `private`, `protected`) to strictly enforce proper encapsulation standards.

Creating

85. **Pulse App Video Class:**

```
<?php
class Video {
   public $title;
   public function __construct($title)
   {
      $this->title = $title;
   }
   public function display()
   {
      echo "Rendering Video: " . $this->title;
   }
}
$my_video = new Video("My Awesome Vacation");
$my_video->display();
?>
```

86. **Secure Form Interface:**

HTML

```
<form method="POST" action="process.php">
    <label>Student Name:</label>
    <input type="text" name="student_name" required>
    <button type="submit">Secure Submit</button>
</form>
```

87. **Mysqli Data Snippet:**

```
<?php
$query = mysqli_query($conn, "SELECT visitor_count FROM site_data LIMIT 1");
$data = mysqli_fetch_assoc($query);
echo "Total Visitors: " . $data['visitor_count'];
?>
```

88. **Custom Exception Catching:**

```
<?php
try {
    $userInput = -10;
    if ($userInput < 0) { throw new InvalidArgumentException("Input cannot be
negative."); }
} catch (InvalidArgumentException $e) {
    echo "Validation Failed: " . $e->getMessage();
}
?>
```

89. **Structural Pagination Algorithm:**

```
<?php
$page = isset($_GET['page']) ? $_GET['page'] : 1;
$perPage = 10;
$totalItems = 250; // Typically from a COUNT() DB query
$offset = ($page - 1) * $perPage;
$totalPages = ceil($totalItems / $perPage);

?>
```

90. **Nested Foreach for Marks:**

```
<?php
$subjects = [
    "Math" => ["Alice" => 85, "Bob" => 70],
    "Science" => ["Alice" => 92, "Bob" => 88]
];
foreach ($subjects as $subject => $students) {
    foreach ($students as $name => $mark) {
        echo "$name scored $mark in $subject.<br>";
    }
}
?>
```

91. **File Upload Script:**

```
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST' && isset($_FILES['doc'])) {
    $targetDir = "uploads/";
    $targetFile = $targetDir . basename($_FILES['doc']['name']);
    move_uploaded_file($_FILES['doc']['tmp_name'], $targetFile);
}
?>
```

92. **Session Redirect:**

```
<?php
session_start();
if (!isset($_SESSION['authenticated']) || $_SESSION['authenticated'] !== true) {
    header("Location: login.php");
    exit();
}
?>
```

93. **Stock Simulation While Loop:**

```
<?php
$stock = 3;
while ($stock > 0) {
    echo "Item sold! <br>";
    $stock--;
}
echo "Out of stock.";
?>
```

98. **Pyramid Pattern Builder:**

```
<?php
for ($row = 1; $row <= 5; $row++) {
    for ($col = 1; $col <= $row; $col++) {
        echo "*";
    }
    echo "<br>";
}
?>
```

99. **Continue Control Snippet:**

```
<?php
for ($i = 1; $i <= 10; $i++) {
    if ($i == 4 || $i == 7) { continue; } // Skips 4 and 7
    echo "Processing object $i<br>";
}
?>
```

100. **Encapsulated Database Template:**

```
<?php
class DatabaseConnection {
    private $host = "localhost";
    private $username = "db_admin";
    private $password = "securePass123";

    public function connect() {
        // Connection logic using private credentials
    }
}
?>
```