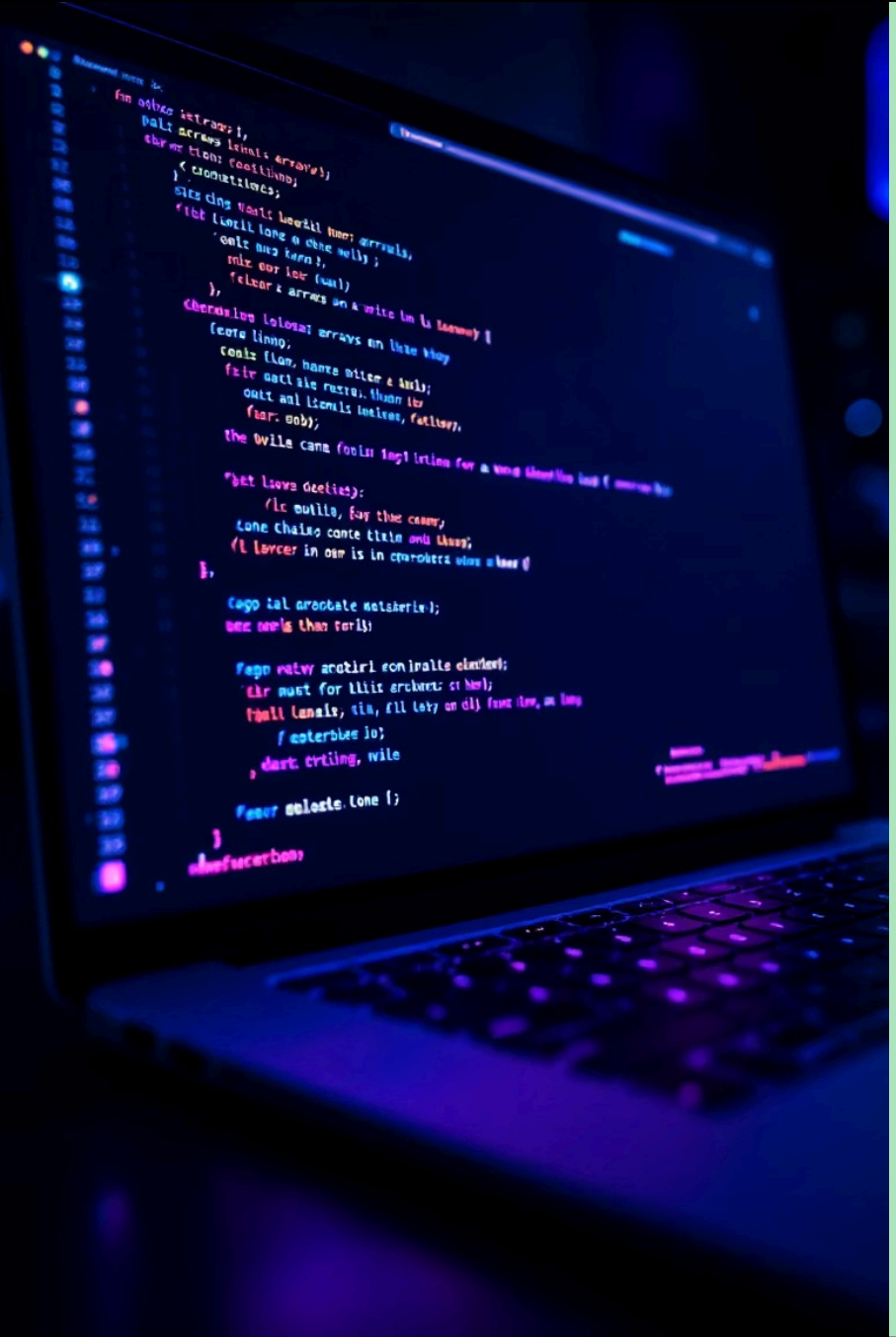


Basic PHP Syntax

Explore fundamental PHP programming concepts focusing on arrays and string manipulation with regular expressions.

Arrays

Dynamic data structures for storing multiple values

Strings & RegExp

Text processing and pattern matching techniques

Course: CS380

Arrays

Create Empty Array

```
$name = array();
```

Create with Values

```
$name = array(value0, value1, ..., valueN);
```

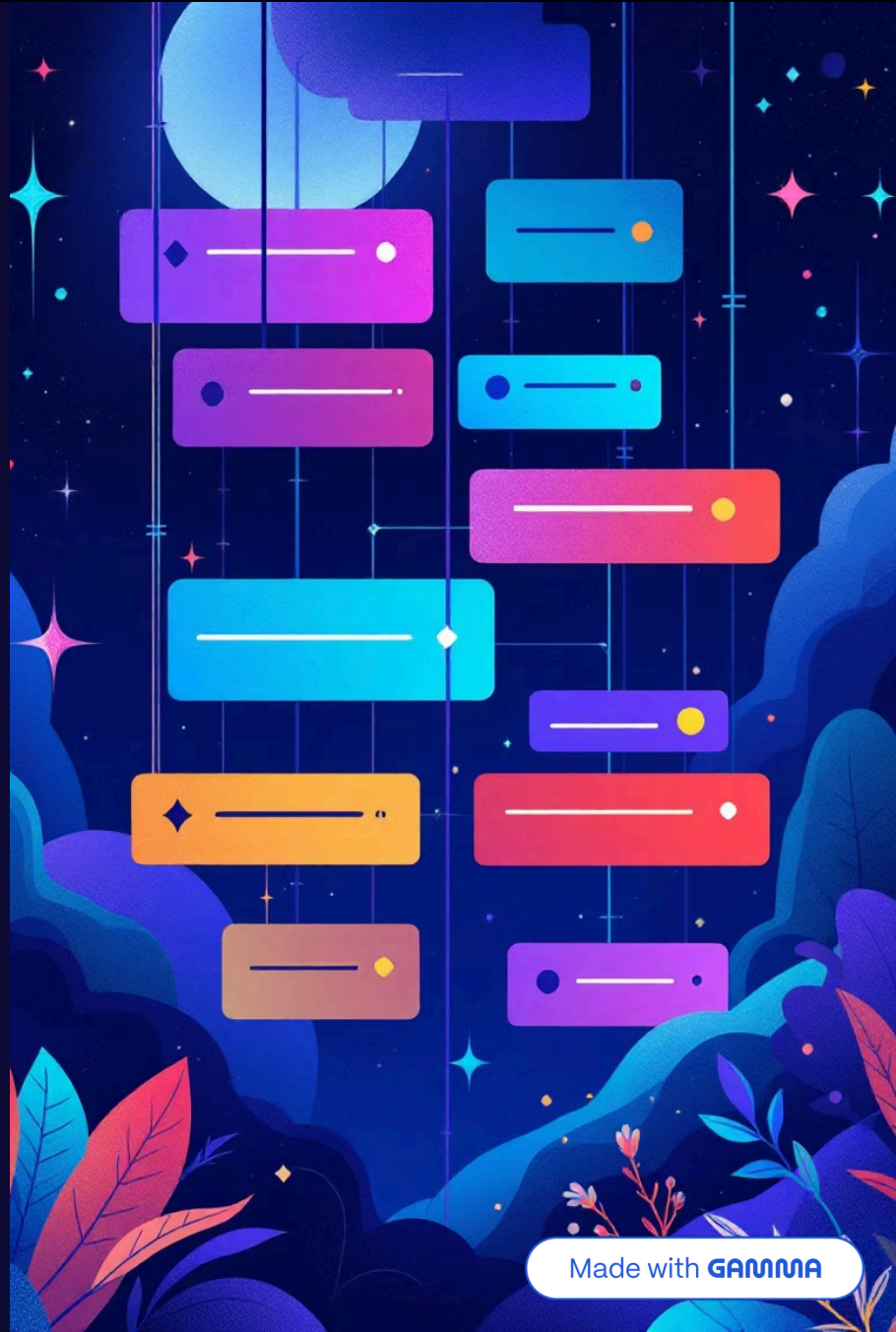
Access Elements

```
$name[index] // get value
$name[index] = value; // set value
```

Practical Example

```
$a = array(); // empty array (length 0)
$a[0] = 23; // stores 23 at index 0 (length 1)
$a2 = array("some", "strings", "in", "an", "array");
$a2[] = "Ooh!"; // append to end (at index 5)
```

Key Features: Use bracket notation without index to append. Element types can be mixed within the same array.



Array Functions

PHP provides powerful built-in functions for array manipulation and processing.

Function Name(s)	Description
count	Number of elements in the array
print_r	Print array's contents
array_pop, array_push, array_shift, array_unshift	Using array as a stack/queue
in_array, array_search, array_reverse, sort, rsort, shuffle	Searching and reordering
array_fill, array_merge, array_intersect, array_diff, array_slice, range	Creating, filling, filtering
array_sum, array_product, array_unique, array_filter, array_reduce	Processing elements

Array Function Example

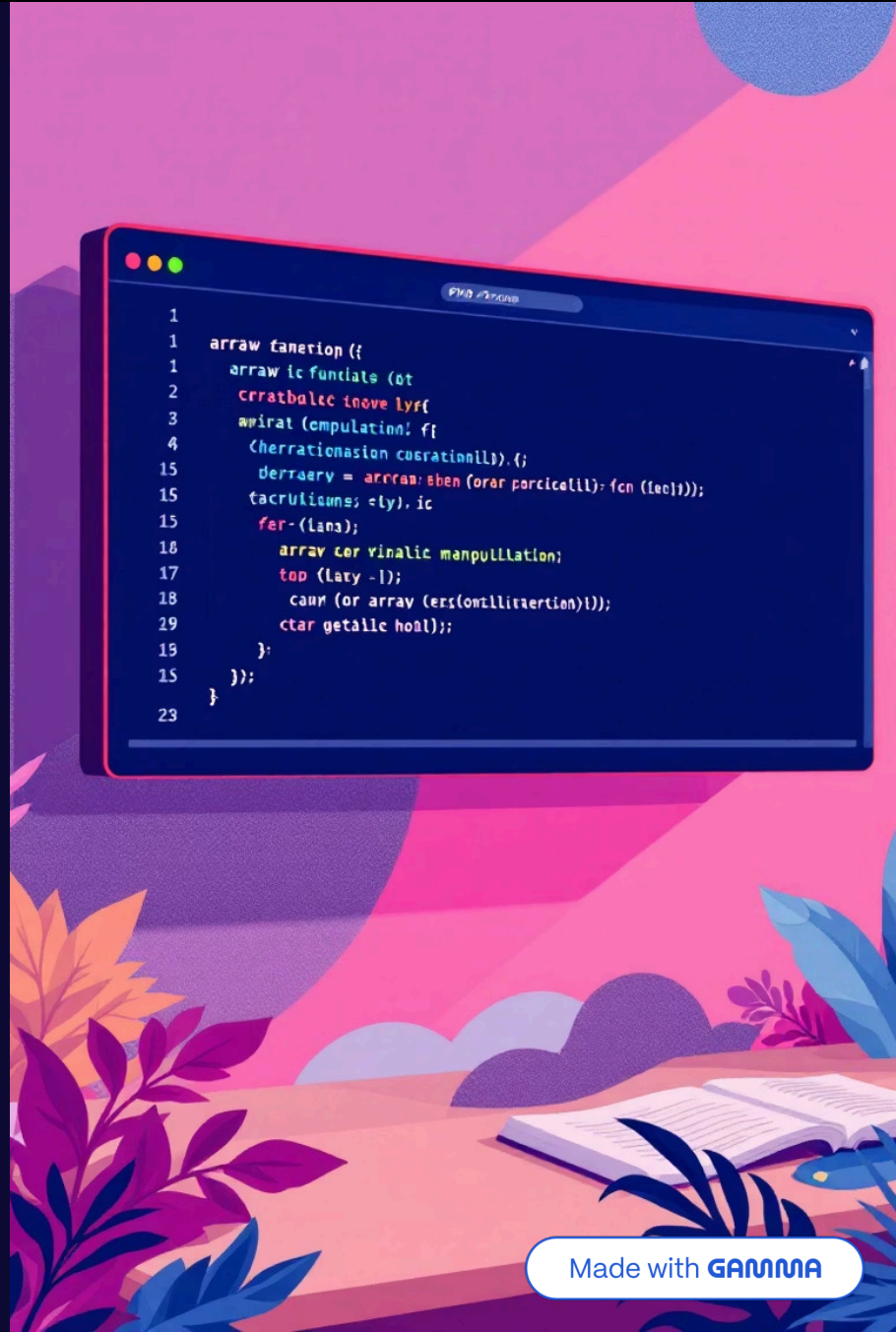
```php
$tas = array("MD", "BH", "KK", "HM", "JP");
for ($i = 0; $i < count($tas); $i++) {
    $tas[$i] = strtolower($tas[$i]);
}
$morgan = array_shift($tas);  // Remove first element
array_pop($tas);              // Remove last element
array_push($tas, "ms");       // Add to end
array_reverse($tas);          // Reverse order
sort($tas);                   // Sort alphabetically
$best = array_slice($tas, 1, 2); // Extract subset
```

Stack/Queue Operations

Use push/pop for stack behavior, shift/unshift for queue operations.

Versatile Collection

PHP arrays replace multiple Java collections: list, stack, queue, set, and map functionality.



foreach Loop

1

Syntax

```
foreach ($array as $variableName) {
    // process each element
}
```

2

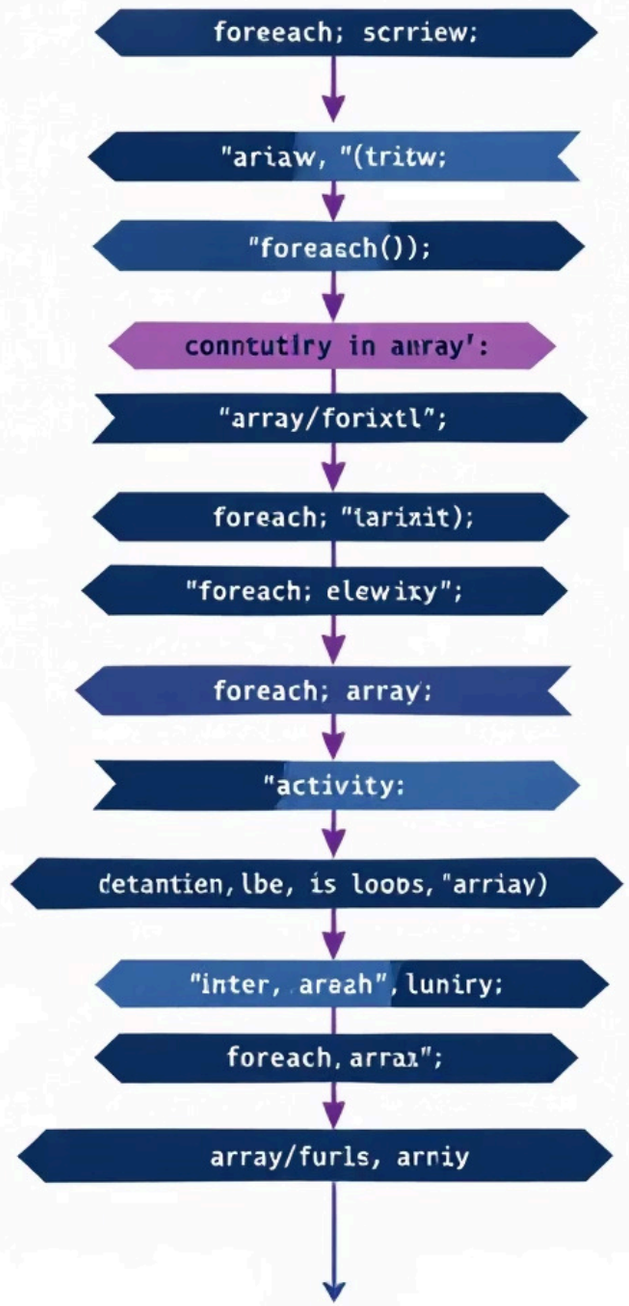
Cleaner Code

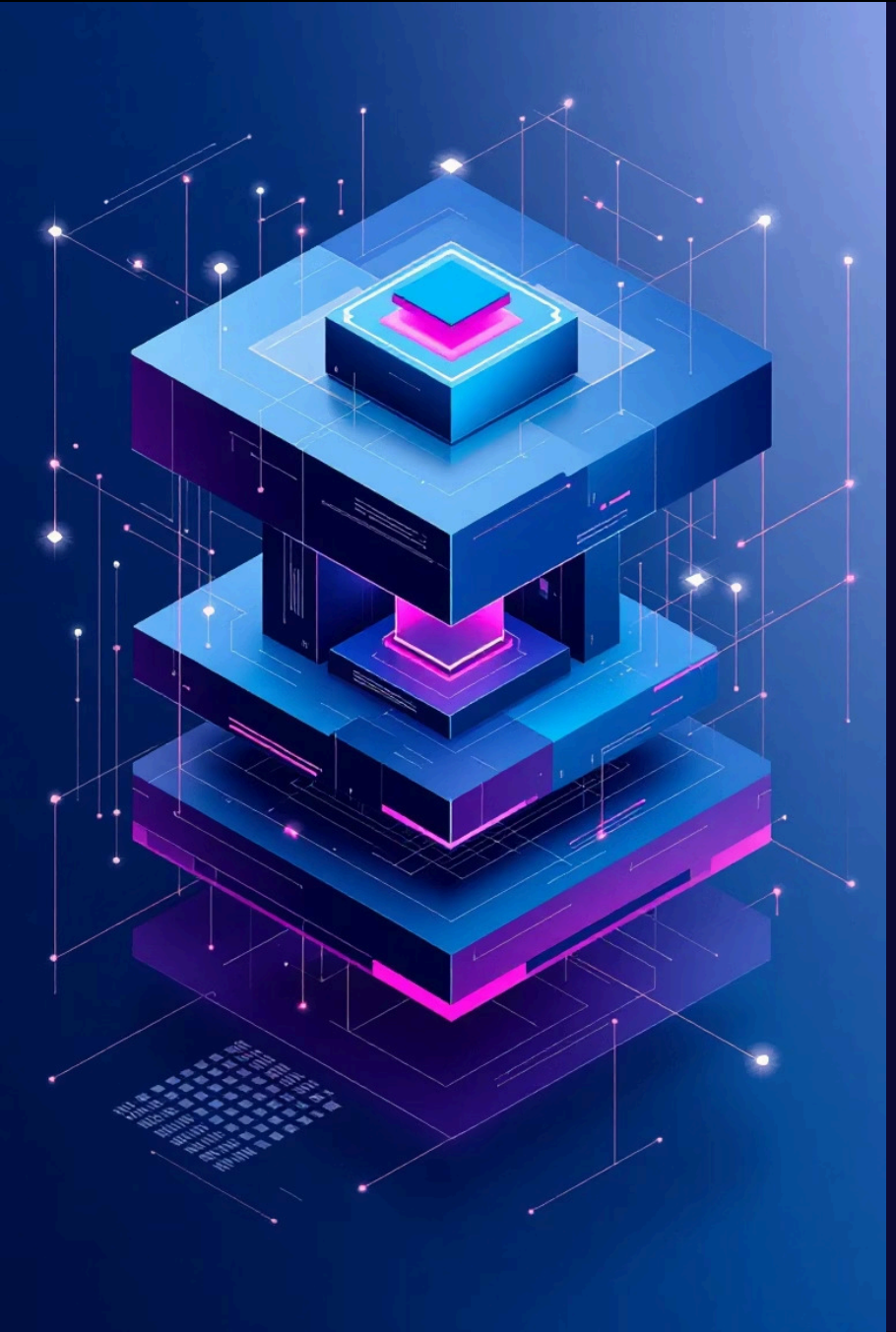
More readable than traditional for loops when processing all elements

Comparison Example

```
$fellowship = array("Frodo", "Sam", "Gandalf", "Strider", "Gimli", "Legolas", "Boromir");

// Traditional for loop
print "The fellowship of the ring members are:\n";
for ($i = 0; $i < count($fellowship); $i++) {
    print "{$fellowship[$i]}\n";
}

// foreach loop - cleaner!
print "The fellowship of the ring members are:\n";
foreach ($fellowship as $fellow) {
    print "$fellow\n";
}
```

Best Practice: Use foreach when you need to process every element in an array - it's more readable and less error-prone.





Multidimensional Arrays

Arrays can contain other arrays, creating nested data structures for complex data organization.

1

Numeric Indices

```
<?php $AmazonProducts = array(
    array("BOOK", "Books", 50),
    array("DVDs", "Movies", 15),
    array("CDs", "Music", 20)
);

for ($row = 0; $row < 3; $row++) {
    for ($column = 0; $column < 3; $column++) { ?>
        <p>
```

2

Accessing Elements

Use $array[row][column] notation to access elements in 2D arrays.

Perfect for representing tables, matrices, or structured data with rows and columns.

Associative Multidimensional Arrays

Use named keys instead of numeric indices for more readable and maintainable code.

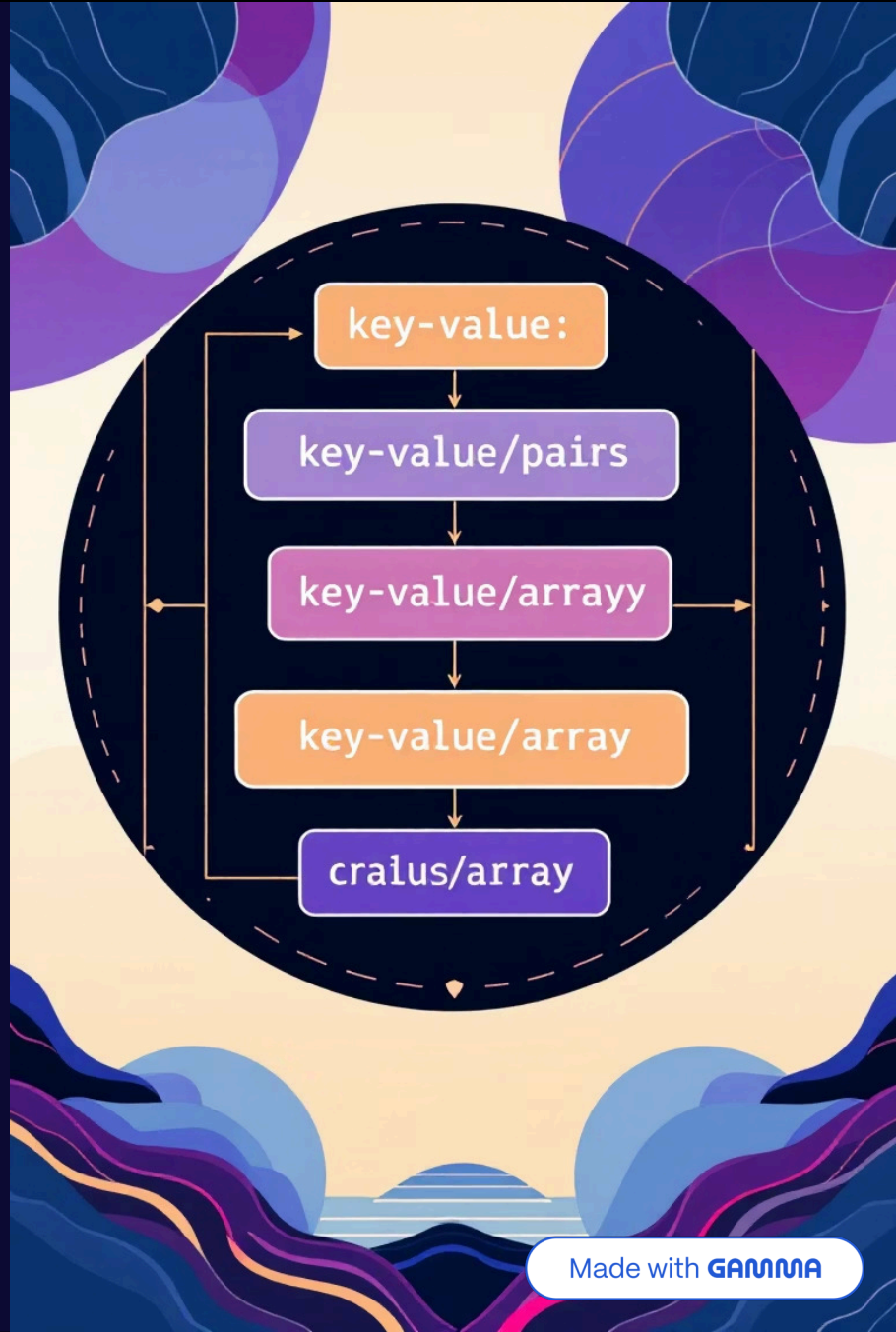
```
<?php $AmazonProducts = array(
    array("Code" => "BOOK", "Description" => "Books", "Price" => 50),
    array("Code" => "DVDs", "Description" => "Movies", "Price" => 15),
    array("Code" => "CDs", "Description" => "Music", "Price" => 20)
);

for ($row = 0; $row < 3; $row++) { ?>
    <p>
```

Named Keys

Use descriptive names like "Code", "Description" instead of numbers

Database-like

Perfect for representing structured data similar to database records



String Compare Functions

PHP provides powerful functions for string comparison and manipulation operations.

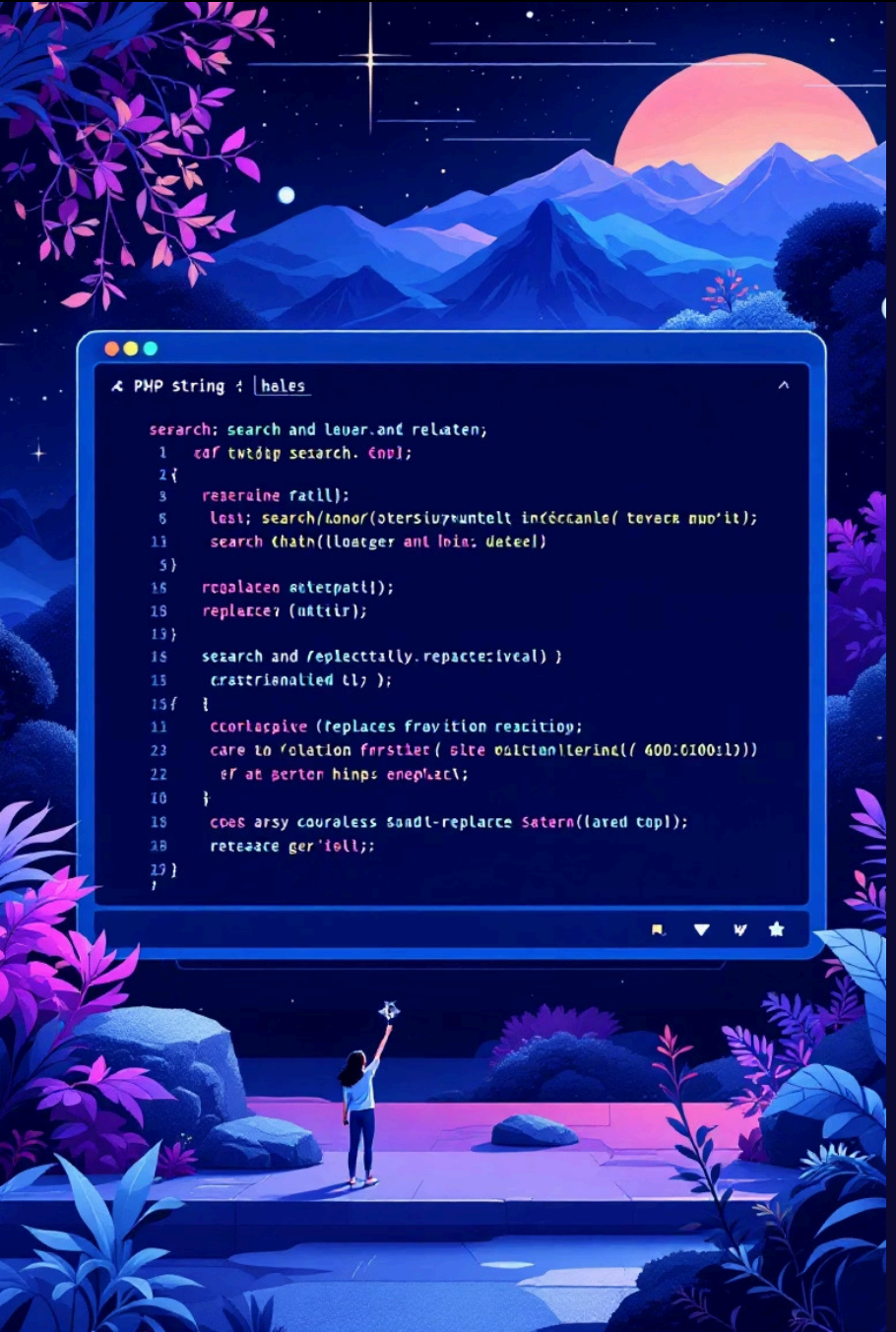
Name	Function
strcmp	compareTo equivalent
strstr, strchr	Find string/char within a string
strpos	Find numerical position of string
str_replace, substr_replace	Replace string content

Comparison Types

- Partial matches
- Case-sensitive options

Case-Insensitive

Use strcasecmp for case-insensitive comparisons



String Function Examples

01

String Replacement

```
$offensive = array("badword1", "badword2");
$feedback = str_replace($offensive, "%!@*", $feedback);
```

Clean user input by replacing inappropriate content

02

Position Finding

```
$test = "Hello World!\n";
print strpos($test, "o");     // Returns 4
print strpos($test, "o", 5);  // Start search from position 5
```

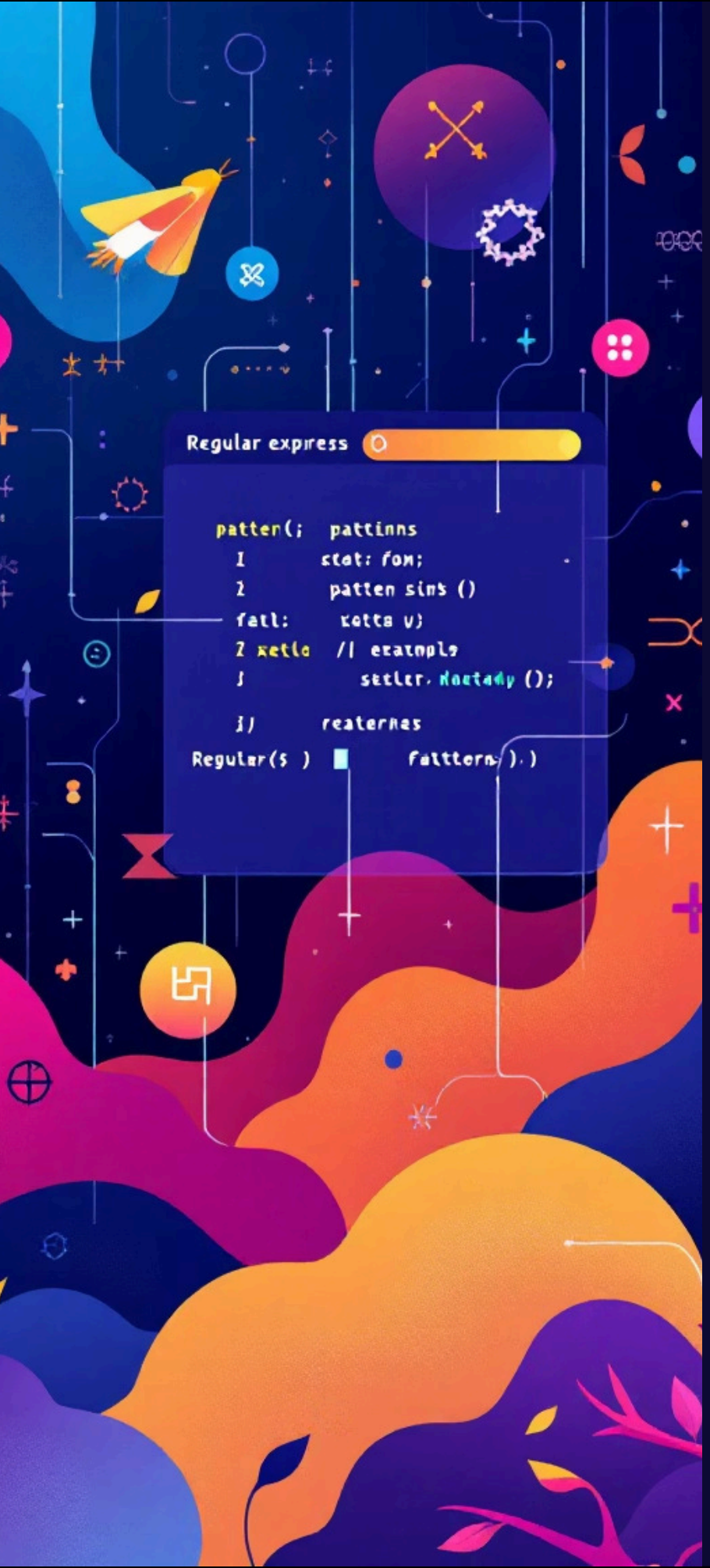
Locate specific characters or substrings within text

03

Conditional Routing

```
$toaddress = "feedback@example.com";
if(strstr($feedback, "shop"))
    $toaddress = "shop@example.com";
else if(strstr($feedback, "delivery"))
    $toaddress = "fulfillment@example.com";
```

Route messages based on content keywords



Regular Expressions

Regular expressions are powerful patterns for matching and manipulating text. PHP supports both POSIX and Perl-style regex.

1

Character Classes

```
[a-z]at    // cat, rat, bat...
[aeiou]    // any vowel
[a-zA-Z]   // any letter
[^a-z]     // NOT lowercase letter
```

2

Predefined Classes

```
[[:alnum:]]+  // at least one alphanumeric char
```

3

Quantifiers

```
(very)*large     // large, very large, very very large...
(very){1,3}      // "very" appears 1 to 3 times
```

4

Anchors

```
^bob    // "bob" at the beginning
com$    // "com" at the end
```

Pattern Matching: Regular expressions define patterns to find, validate, or replace specific text sequences within strings.