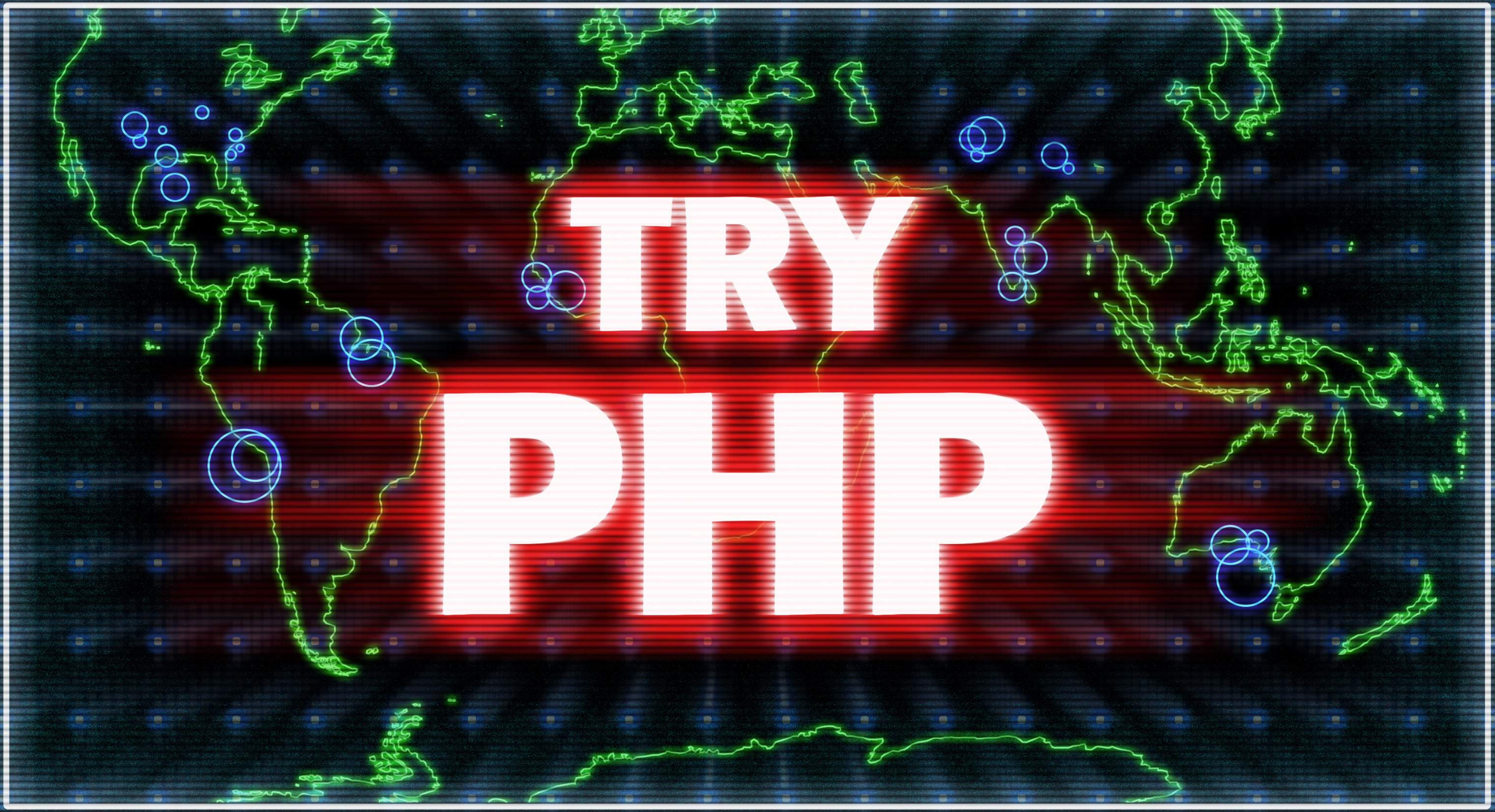

TRY
PHP

Level 1

Getting Started

The Basics of PHP

What Will This Course Cover?

Here’s what we’ll go over in this course.

Level 1

Syntax Basics & Variables

Level 2

Simple Arrays & Associative Arrays

Level 3

Operators & Comparison Statements

Level 4

Looping Constructs

What Do You Need to Know?

Suggested prerequisites:

Basic HTML & CSS

Front-end Foundations & Front-end Formations

Why PHP, Why Now?

PHP is a server-side scripting language that has been around since 1997 and has grown into a modern and performant tool for building websites and applications.

Allows execution of code inline with our HTML markup

Simple reading and processing of files and images

Request and response processing with forms

High performance, scales easily

Let's get started!

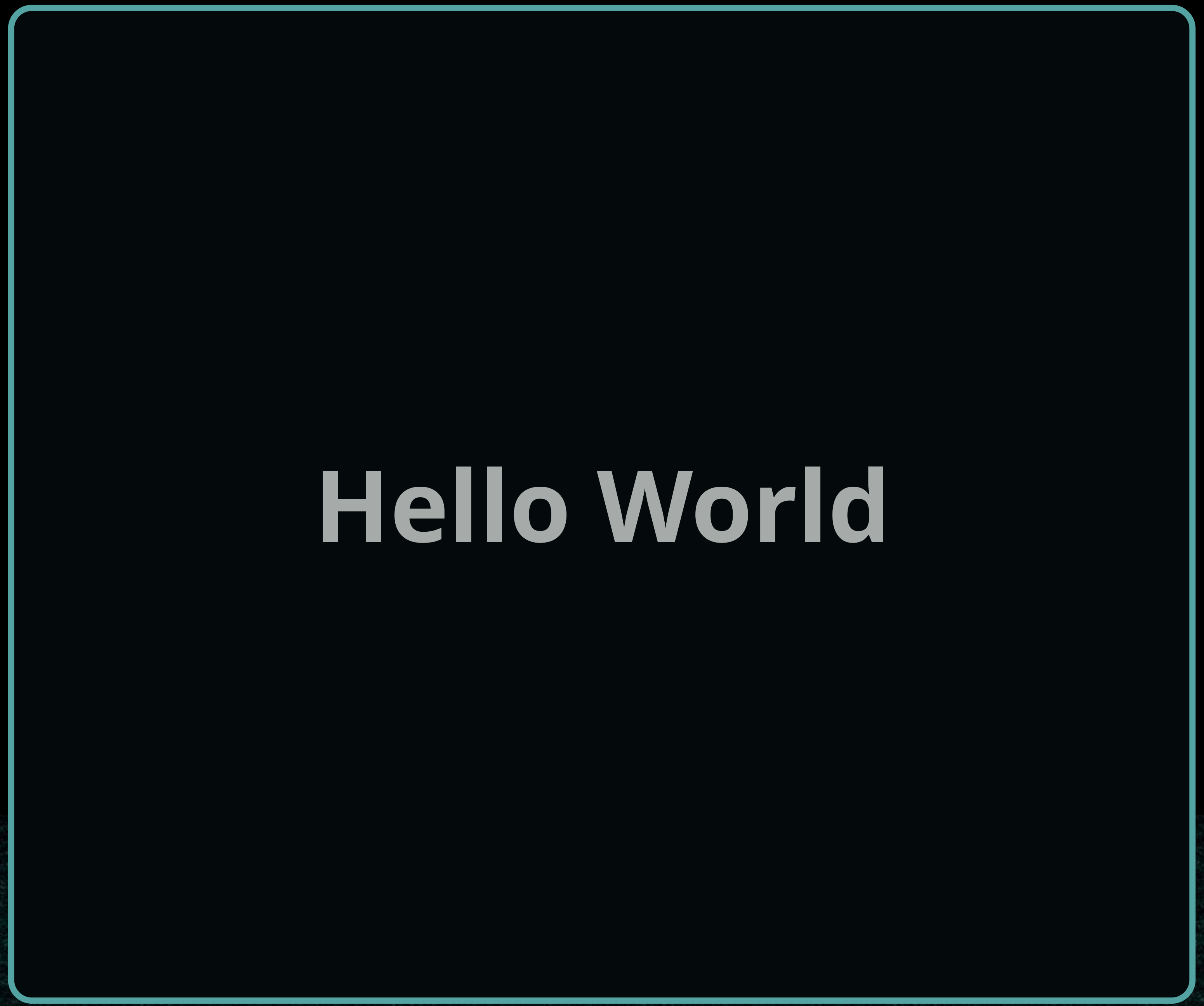
Starting From Scratch

Example of a simple HTML file:

index.html HTML

```
<!DOCTYPE html>
<html>
 <head>
  <title></title>
 </head>
 <body>
   <p>Hello World</p>
 </body>
</html>
```

Output



Hello World

Renaming Our File

Let's change the file from .html to .php so it can be processed by the server.

index.php

PHP

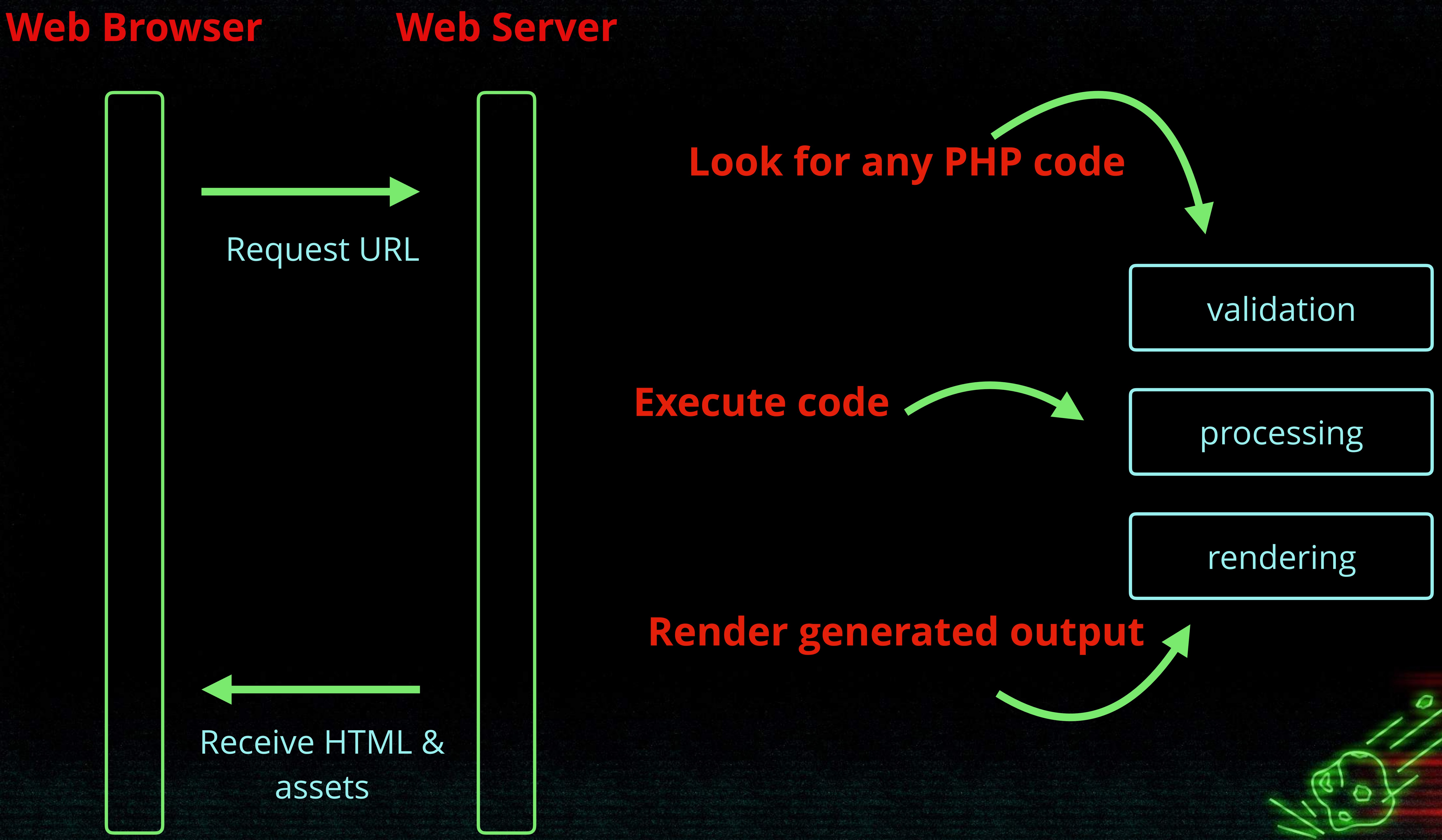
```
<!DOCTYPE html>
<html>
 <head>
  <title></title>
 </head>
 <body>
   <p>Hello World</p>
 </body>
</html>
```

PHP files can render HTML as well as PHP!

Output

Hello World

What Is Different Now?



Web Browser

Web Server

Request URL

Receive HTML & assets

Look for any PHP code

Execute code

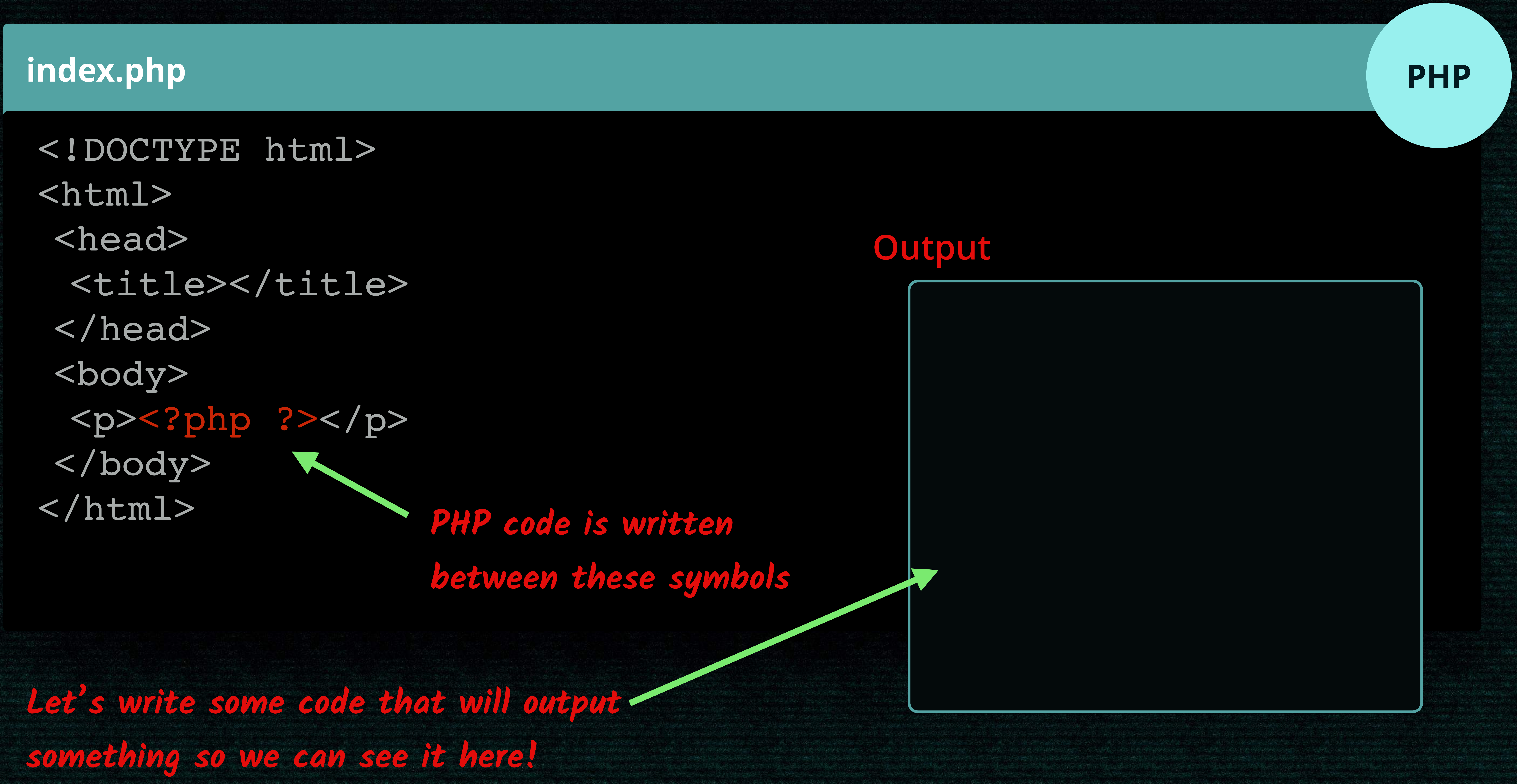
Render generated output

validation

processing

rendering

Creating a Code Block



index.php

PHP

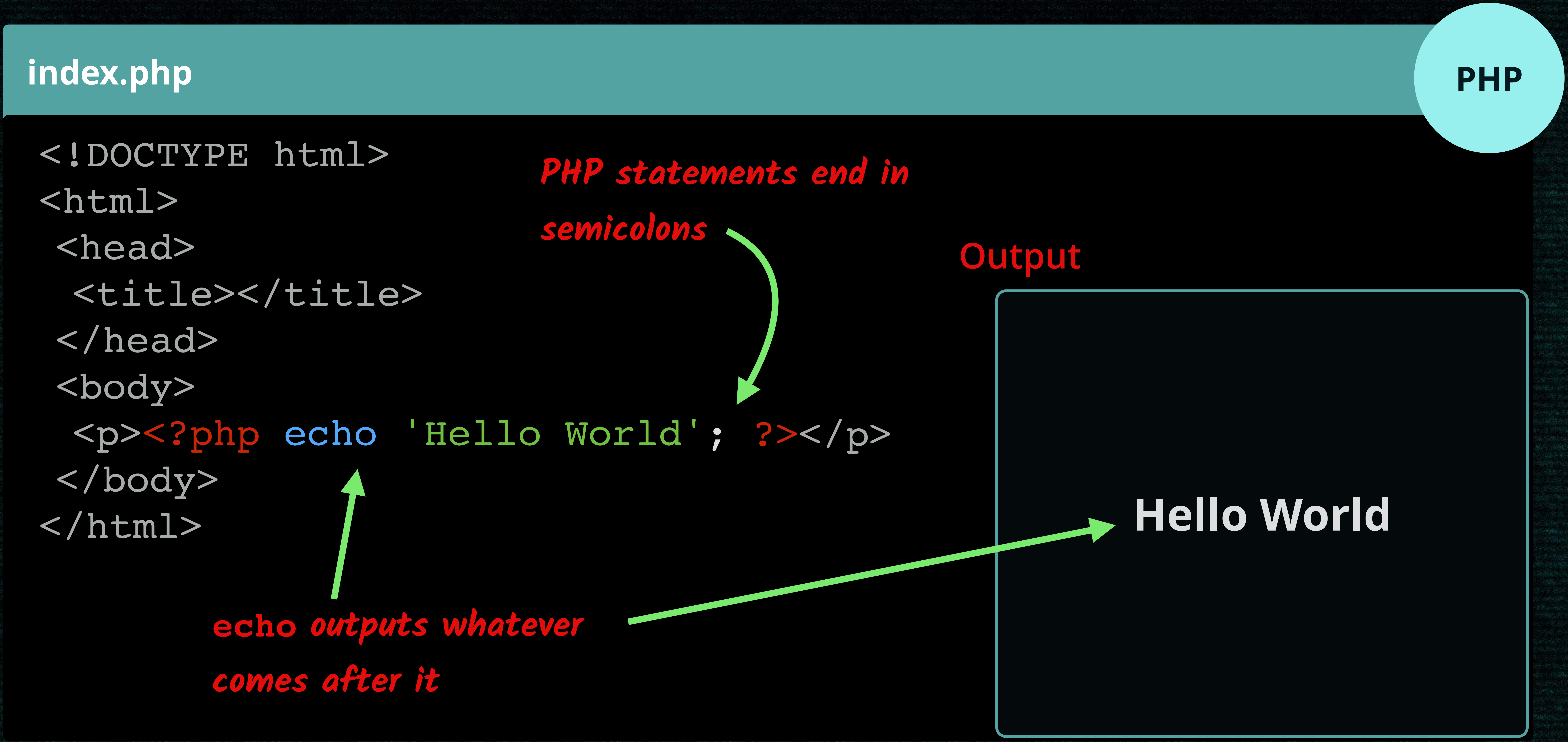
```
<!DOCTYPE html>
<html>
 <head>
  <title></title>
 </head>
 <body>
  <p><?php ?></p>
 </body>
</html>
```

Output

PHP code is written between these symbols

Let’s write some code that will output something so we can see it here!

Our First PHP Code



index.php

PHP

```
<!DOCTYPE html>
<html>
 <head>
  <title></title>
 </head>
 <body>
  <p><?php echo 'Hello World'; ?></p>
 </body>
</html>
```

PHP statements end in semicolons

echo outputs whatever comes after it

Output

Hello World

Variables and Data Subject to Change

index.php

PHP

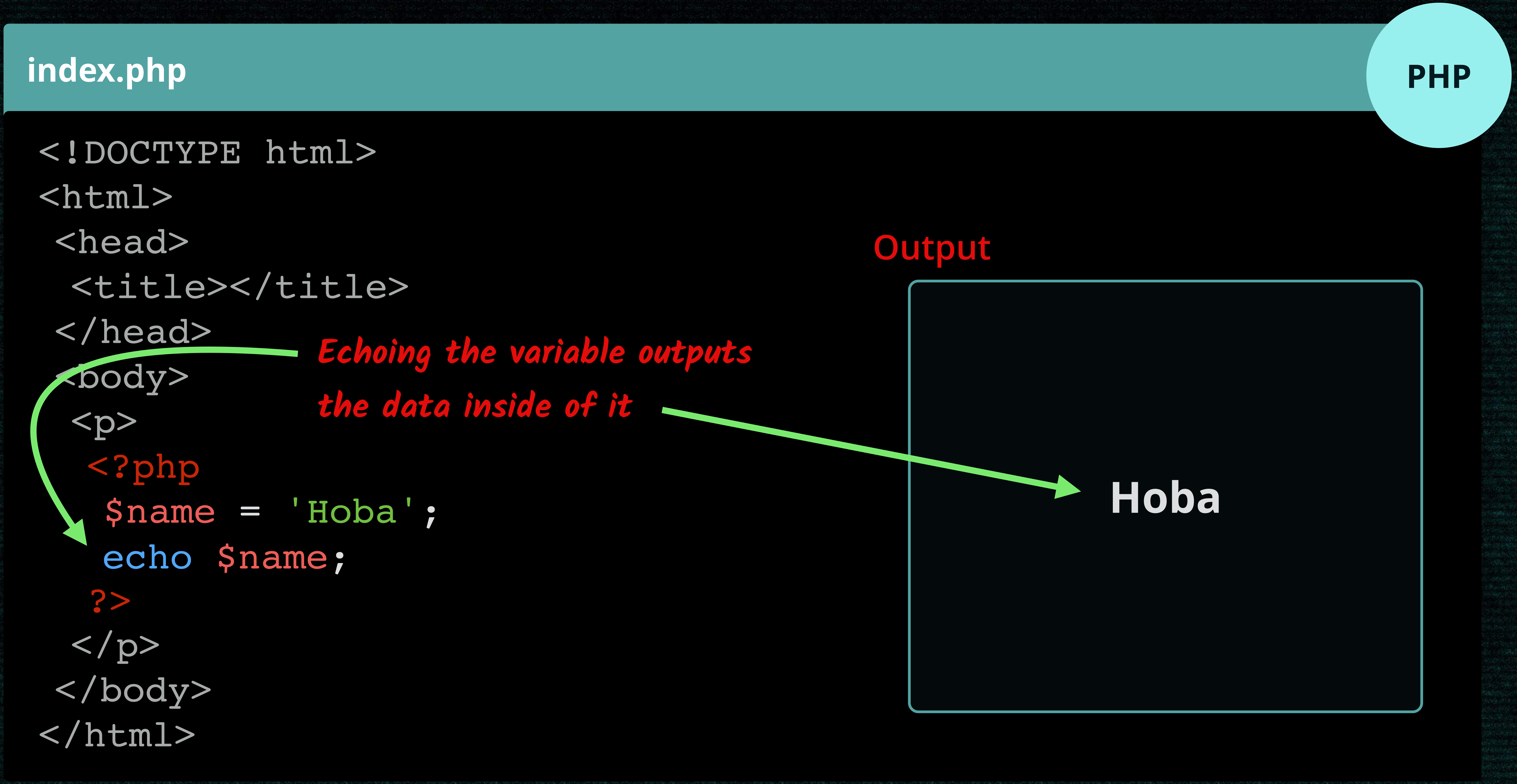
```
<!DOCTYPE html>
<html>
 <head>
  <title></title>
 </head>
 <body>
  <p><?php $name = 'Hoba'; ?></p>
 </body>
</html>
```

Variables in PHP always start with a $

Output

Notice that the data in the variable isn't printed out automatically

Outputting Data That’s Stored in Variables



index.php

PHP

```
<!DOCTYPE html>
<html>
 <head>
  <title></title>
 </head>
 <body>
  <p>
   <?php
    $name = 'Hoba';
    echo $name;
   ?>
  </p>
 </body>
</html>
```

Echoing the variable outputs the data inside of it

Output

```
Hoba
```

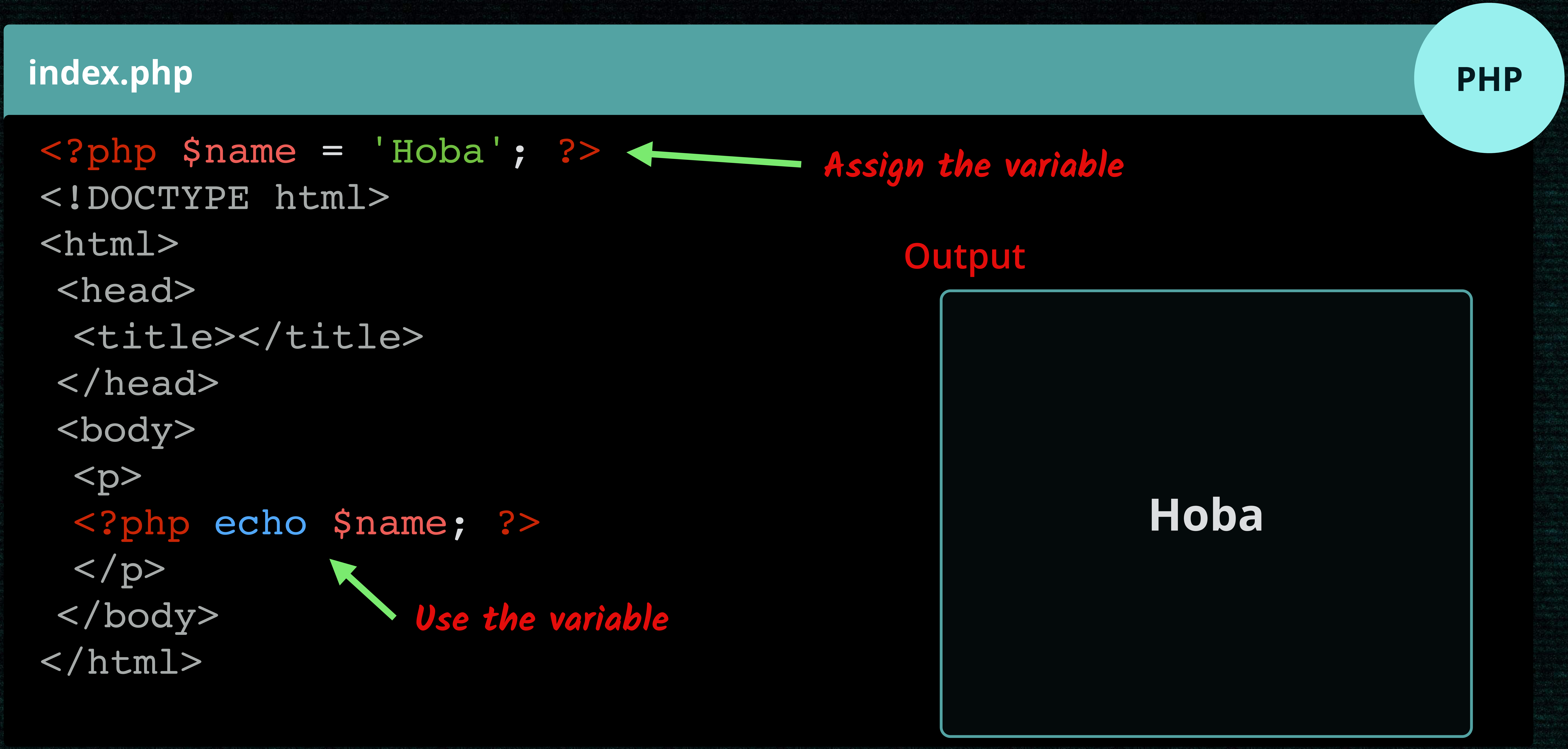
Variable Naming Conventions

Variables must always begin with a $ followed by a letter.

- ✓ $name
- ✓ $_age
- ✓ $full_name
- ✓ $war_1984

- ✗ $10_best_targets
- ✗ $so-very-invalid
- ✗ stalemate

PHP Code Can Go Anywhere



index.php

PHP

```php
<?php $name = 'Hoba'; ?>
<!DOCTYPE html>
<html>
 <head>
  <title></title>
 </head>
 <body>
  <p>
  <?php echo $name; ?>
  </p>
 </body>
</html>
```

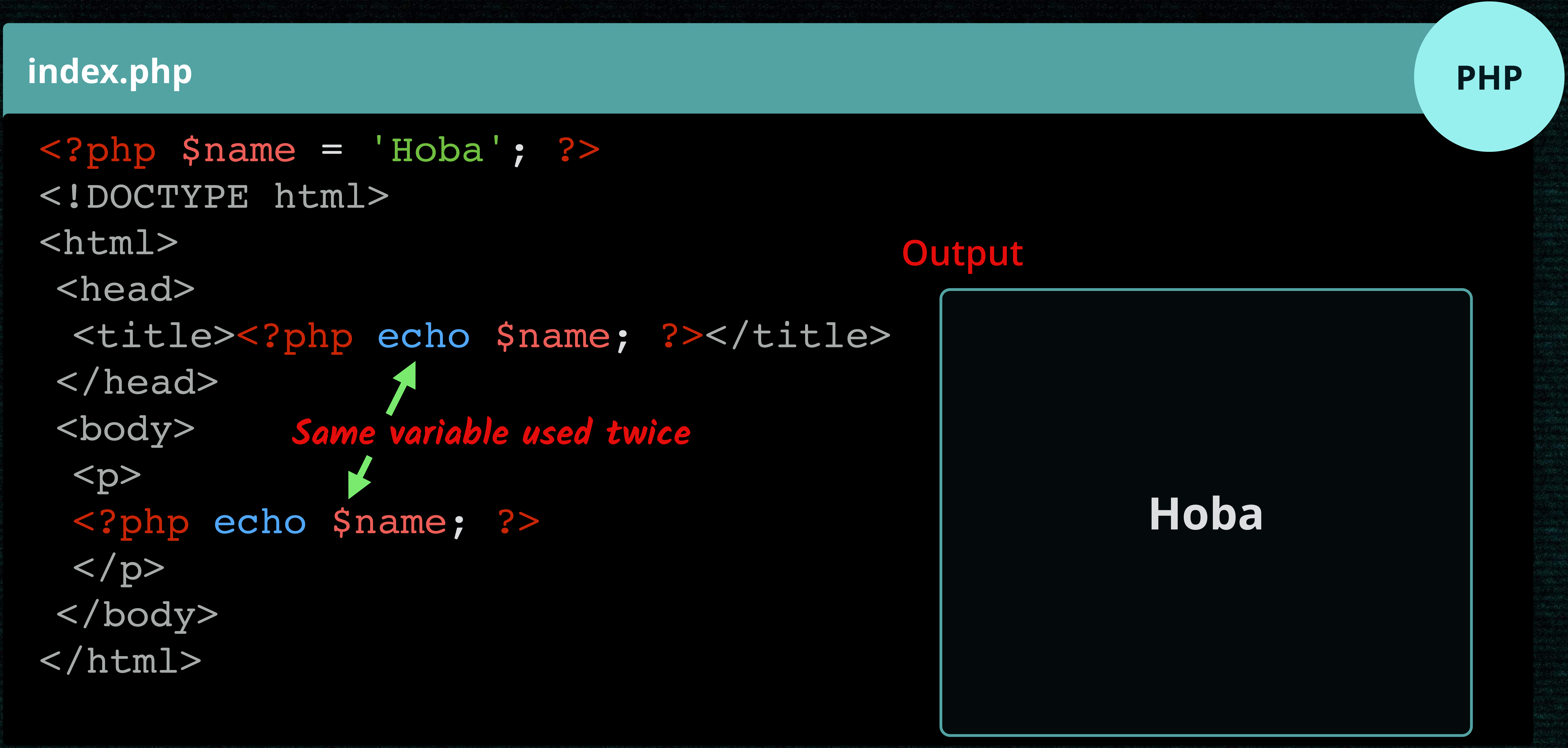
Assign the variable

Use the variable

Output

```
Hoba
```

Variables Can Be Used in Multiple Locations



index.php

PHP

```php
<?php $name = 'Hoba'; ?>
<!DOCTYPE html>
<html>
 <head>
  <title><?php echo $name; ?></title>
 </head>
 <body>
  <p>
  <?php echo $name; ?>
  </p>
 </body>
</html>
```

Same variable used twice

Output

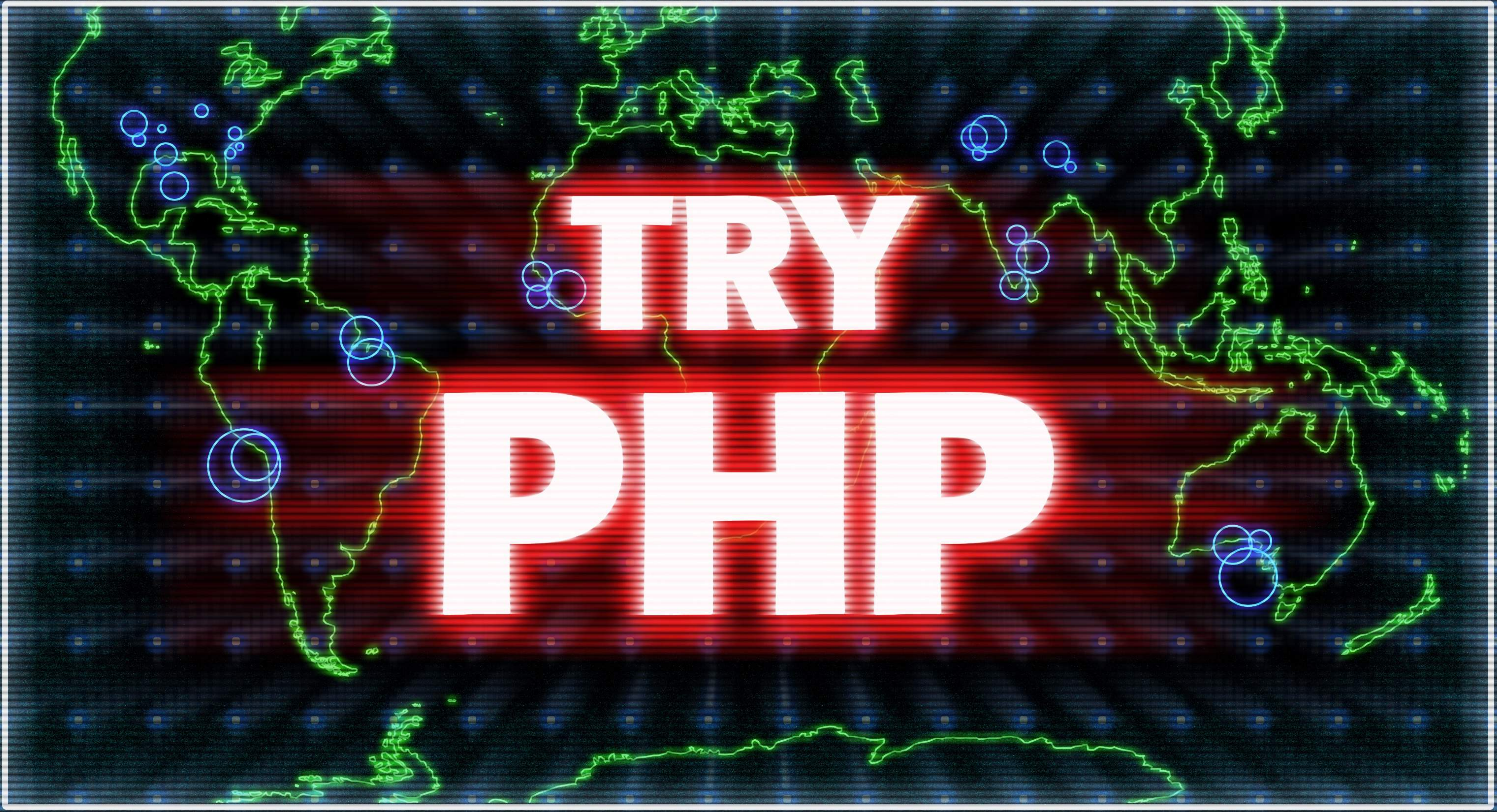
```
Hoba
```

What Have We Learned?

Let's have a quick review.

- Syntax basics
- Code blocks
- PHP request and response server workflow
- Variables and naming rules
- The `echo` statement

Now we can move on to some challenges!



TRY PHP

Level 1

Getting Started

Strings & Data

Our Code So Far

index.php

PHP

```php
<?php $name = 'Hoba'; ?>
<!DOCTYPE html>
<html>
 <head>
  <title><?php echo $name; ?></title>
 </head>
 <body>
  <p>
   <?php echo $name; ?>
  </p>
 </body>
</html>
```

Combining More Data With Variables

index.php

PHP

```php
<?php $name = 'Hoba'; ?>
<!DOCTYPE html>
<html>
 <head>
  <title><?php echo $name; ?></title>
 </head>
 <body>
  <p>
   <?php echo 'Meteor Name: ' . $name; ?>
  </p>
 </body>
</html>
```

Notice the space?

The dot means "combine these two things"

Output

```
Meteor Name: Hoba
```

String Evaluation

index.php

PHP

```
<?php $name = 'Hoba'; ?>
<!DOCTYPE html>
<html>
 <head>
  <title><?php echo $name; ?></title>
 </head>
 <body>
  <p>
   <?php echo "Meteor Name: $name"; ?>
  </p>
 </body>
</html>
```

Variables will print inside strings as long as you wrap them in double quotes

Output

```
Meteor Name: Hoba
```

PHP Data Types: Strings

To define a string, we will surround our information in single quotes during assignment.

```php
<?php

$size = 'epic';
$weight = '600 Million Grams';
```

PHP Data Types: Integers

An integer is a number, either positive or negative, without a decimal point.

```php
<?php

$discovered = 1920;
$speed = 720;
```

PHP Data Types: Floats

Floating point numbers will be any number that decimal point can “float.”

```
<?php

$width = 8.9;
$latitude = -19.5833333333;
```

PHP Data Types: Booleans

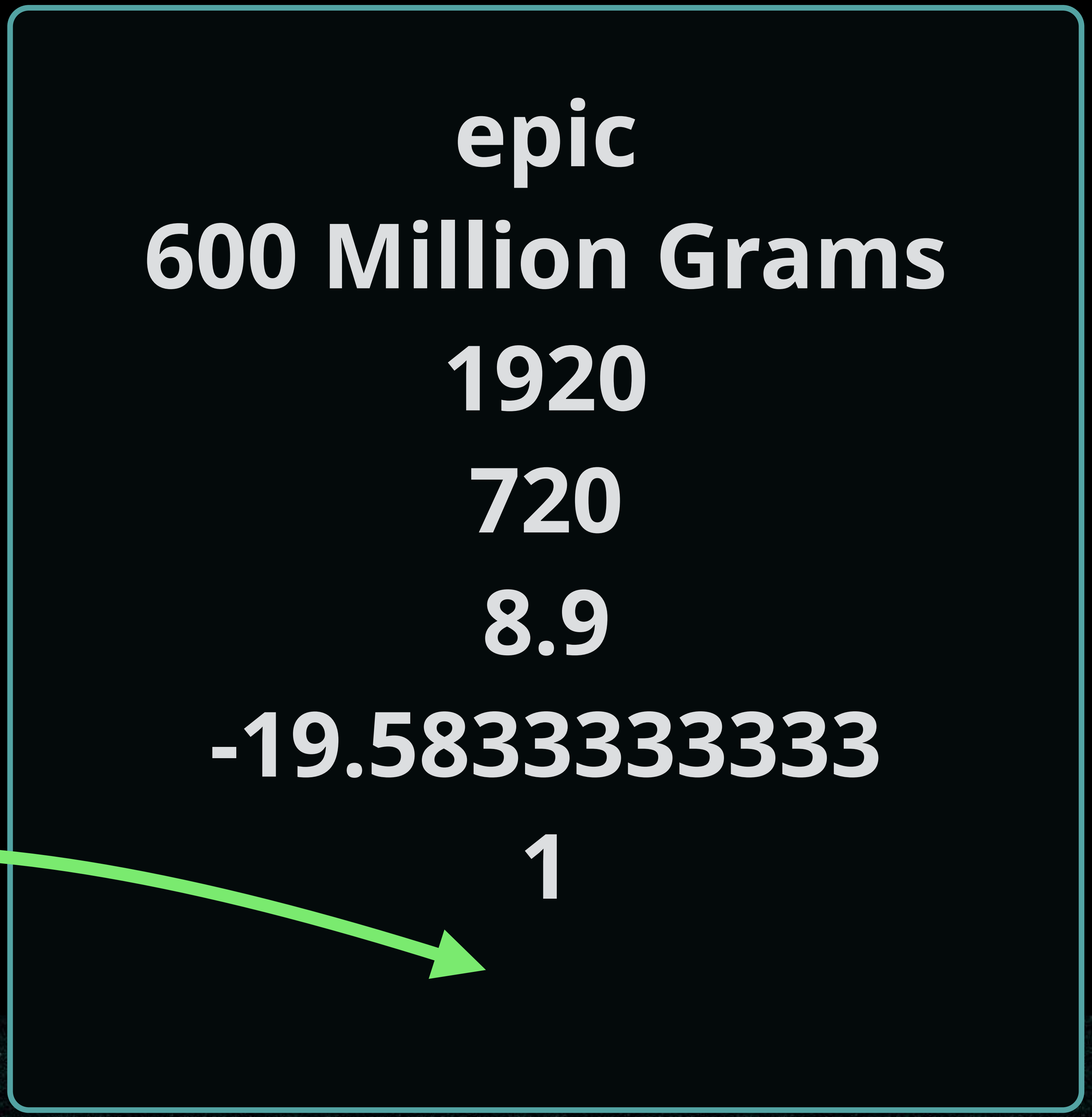
Boolean is a data type that can contain one of two values: a true or a false.

```php
<?php

$largest = true;
$destroyed = false;
```

PHP Data Types: The Results

```
…
  <body>
  <?php
   echo "<p>$size</p>";
   echo "<p>$weight</p>";
   echo "<p>$discovered</p>";
   echo "<p>$speed</p>";
   echo "<p>$width</p>";
   echo "<p>$latitude</p>";
   echo "<p>$largest</p>";
   echo "<p>$destroyed</p>";
  ?>
  </body>
```

Output



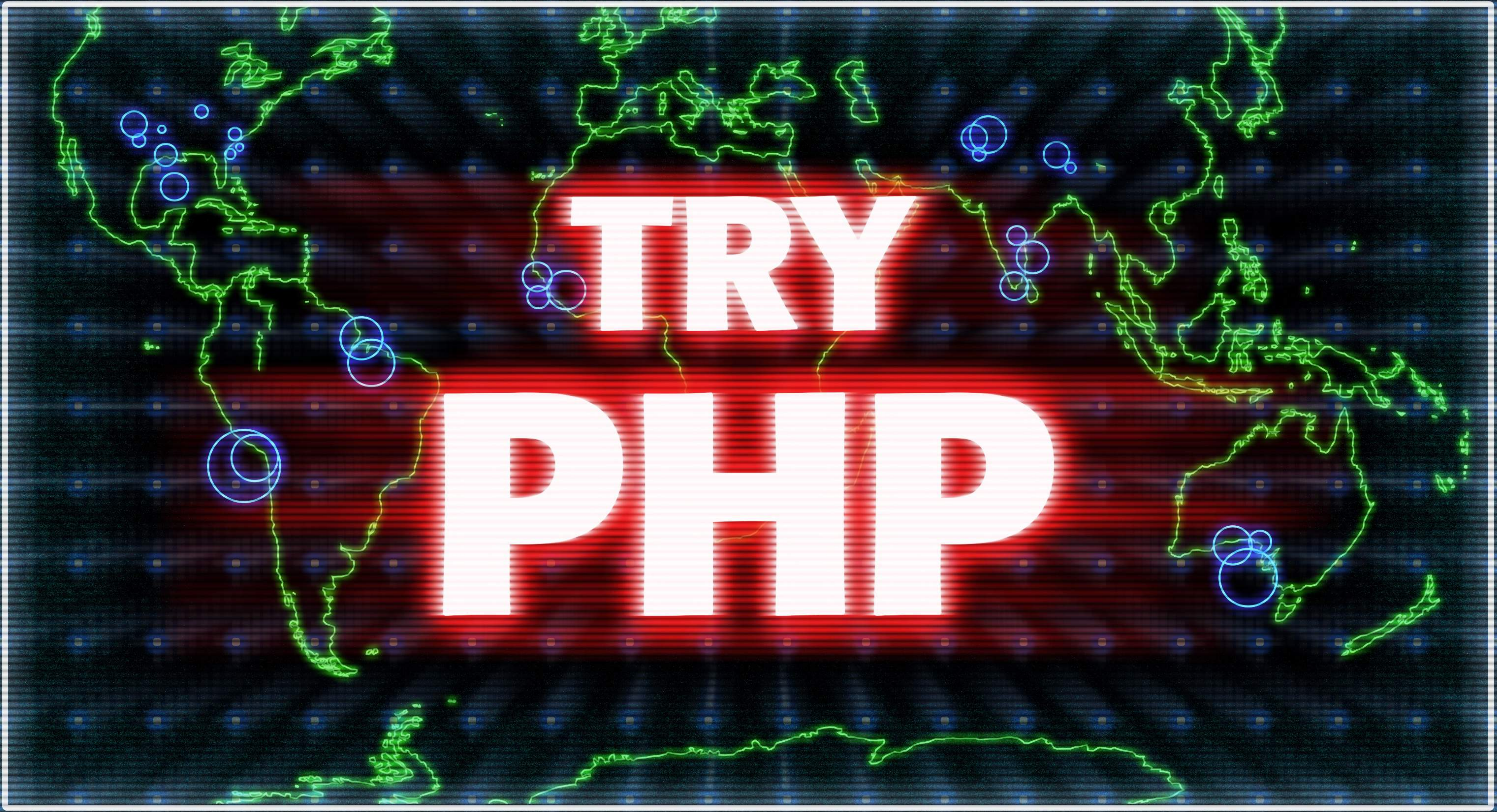
```
epic
600 Million Grams
1920
720
8.9
-19.5833333333
1
```

If you echo a false boolean, nothing will appear

What Have We Learned?

Let’s have a quick review.

- String concatenation
- Strings
- Integers
- Floating point numbers or floats
- Booleans



TRY PHP

Level 2

Arrays

Simple & Associative Arrays

Why an Array?

Variables alone will not scale. We need a better way to keep our data.

index.php

```php
<?php
// We could keep going with variables
$meteor_1 = 'Hoba';
$meteor_2 = 'Cape York';
$meteor_3 = 'Campo del Cielo';
$meteor_4 = 'Canyon Diablo';
…
$meteor_42 = 'Prefect';
```

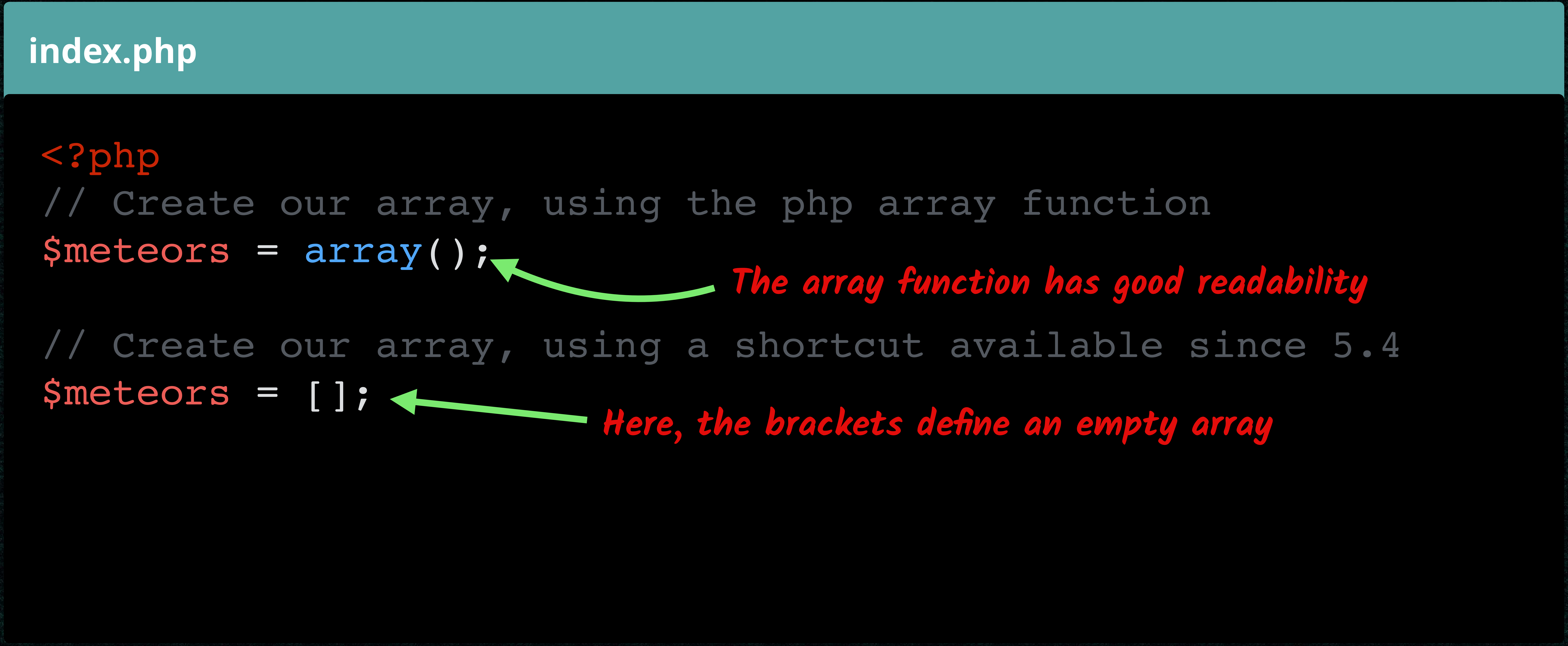
Arrays, a Map

An array maps values to keys, like an address for setting and recalling.

Key	Value
0	Hoba
1	Cape York
2	Campo del Cielo
3	Canyon Diablo

Creating an Array

Let’s create an empty array to hold our meteorite data.



index.php

```php
<?php
// Create our array, using the php array function
$meteors = array();

// Create our array, using a shortcut available since 5.4
$meteors = [];
```

The array function has good readability

Here, the brackets define an empty array

Array With Values

We can create an array with one or more key value pairs using the same function.

index.php

```php
<?php
// Create our array with a single value
$meteors = array('Hoba');
$meteors = ['Hoba'];

// Create array with multiple values
$meteors = array('Hoba','Cape York');

// Echo the array
echo($meteors);
```

echo *will not show the data within the array*

Output

```
Array
```

Array With Values

We can create an array with one or more key value pairs using the same function.

index.php

```php
<?php
// Create our array with a single value
$meteors = array('Hoba');
$meteors = ['Hoba'];

// Create array with multiple values
$meteors = array('Hoba','Cape York');

// Let's take a look at our array
with an internal function
print_r($meteors);
```

print_r will echo human-readable output

Output

```
Array (
[0] => Hoba
[1] => Cape York
)
```

Adding More Data to Our Array

We can append new values by placing square brackets after the array variable.

index.php

```php
<?php
...
// Let's add two more items
$meteors[] = 'Campo del Cielo';
$meteors[] = 'Canyon Diablo';

print_r($meteors);
```

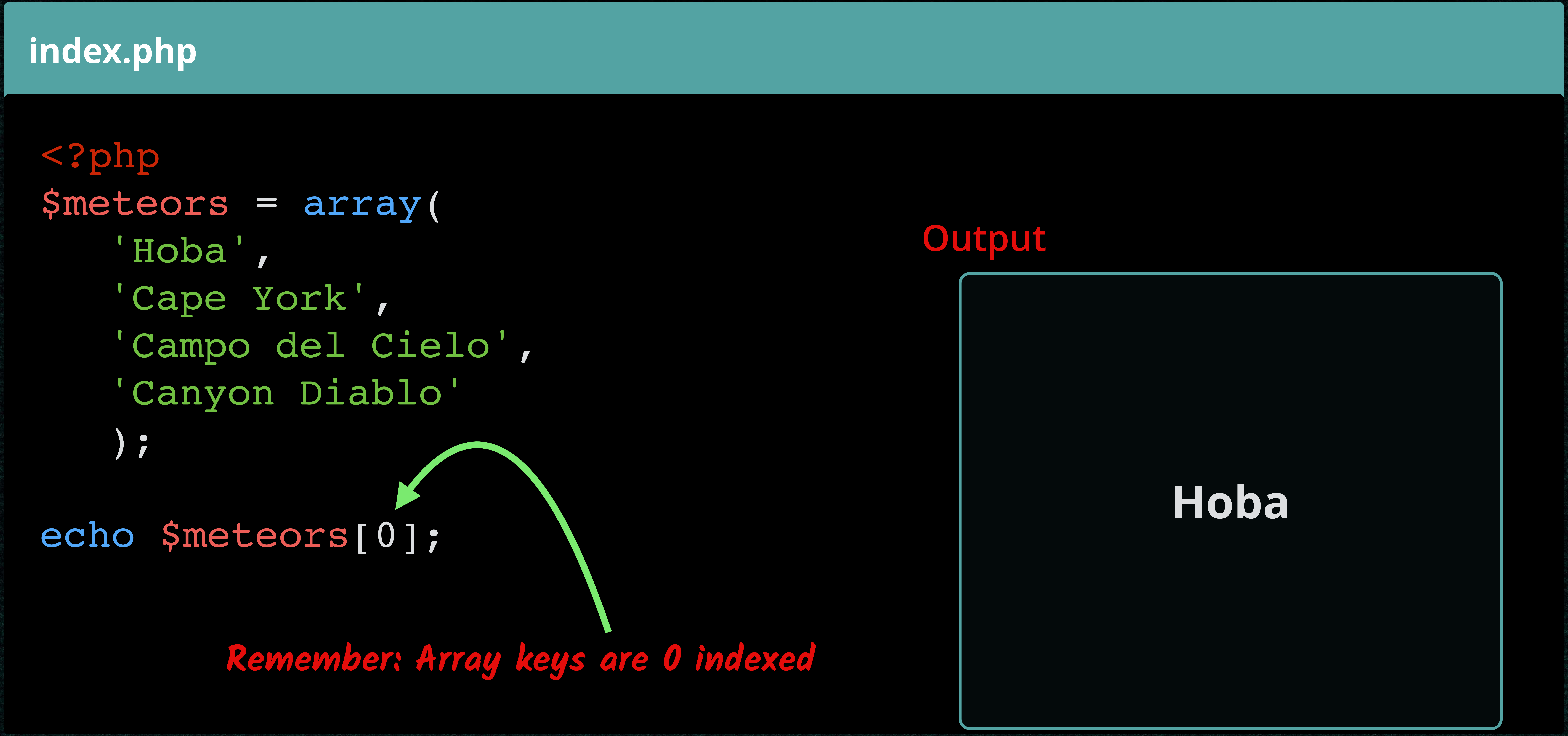
Empty brackets after the variable name indicate a new item in the array

Output

```
Array (
[0] => Hoba
[1] => Cape York
[2] => Campo del Cielo
[3] => Canyon Diablo
)
```

How Can We Access This Data?

Placing the key, or index, inside the square bracket gives us access to the value.



index.php

```php
<?php
$meteors = array(
    'Hoba',
    'Cape York',
    'Campo del Cielo',
    'Canyon Diablo'
    );

echo $meteors[0];
```

Remember: Array keys are 0 indexed

Output

```
Hoba
```

How Can We Access This Data?

Placing the key, or index, inside the square bracket gives us access to the value.

index.php

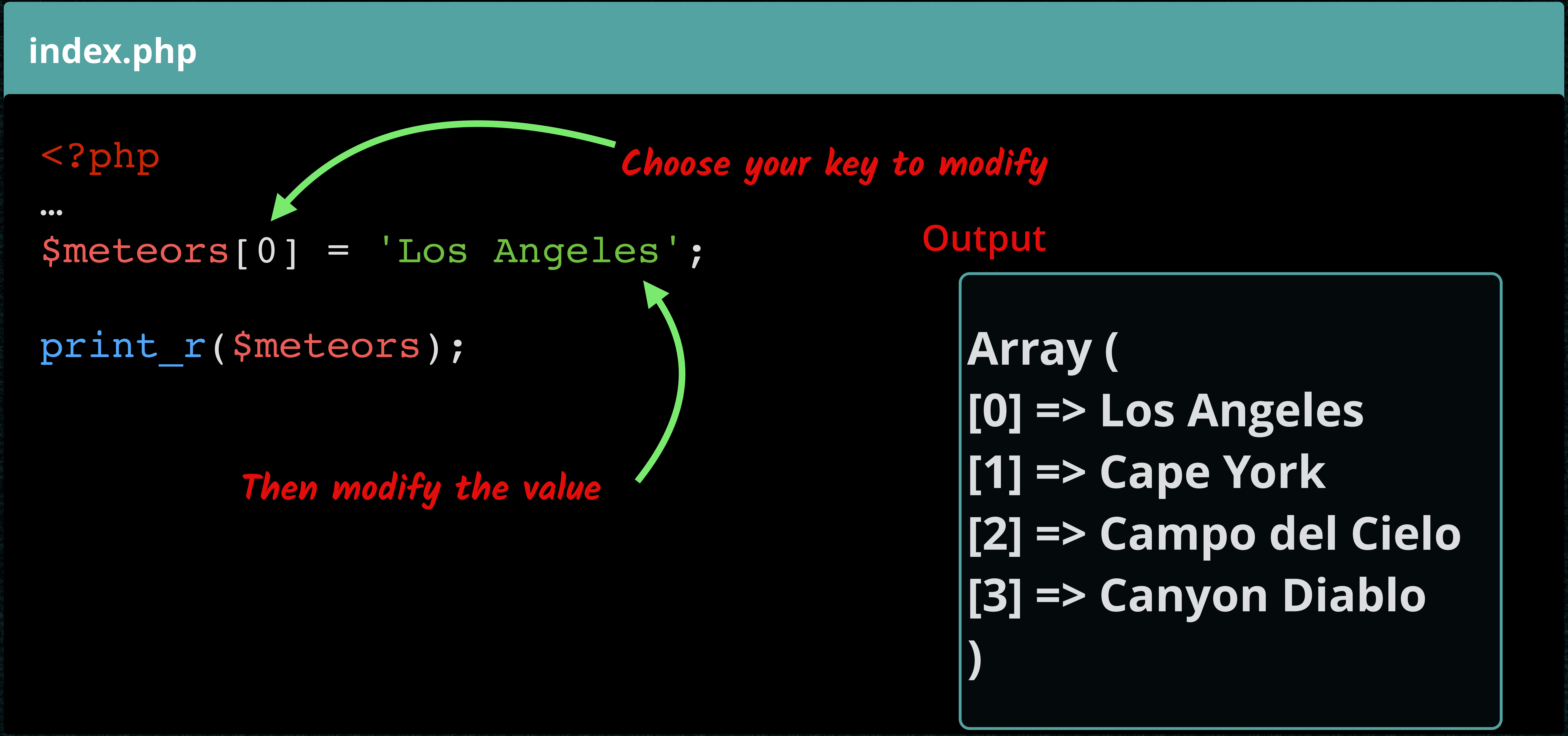
```php
<?php
$meteors = array(
    'Hoba',
    'Cape York',
    'Campo del Cielo',
    'Canyon Diablo'
    );

echo $meteors[0];
echo $meteors[1];
echo $meteors[3];
```

Output

```
Hoba
Cape York
Canyon Diablo
```

Modifying an Existing Item

Placing the key inside also allows us access to modify the value.



index.php

```php
<?php
...
$meteors[0] = 'Los Angeles';

print_r($meteors);
```

Choose your key to modify

Then modify the value

Output

```
Array (
[0] => Los Angeles
[1] => Cape York
[2] => Campo del Cielo
[3] => Canyon Diablo
)
```

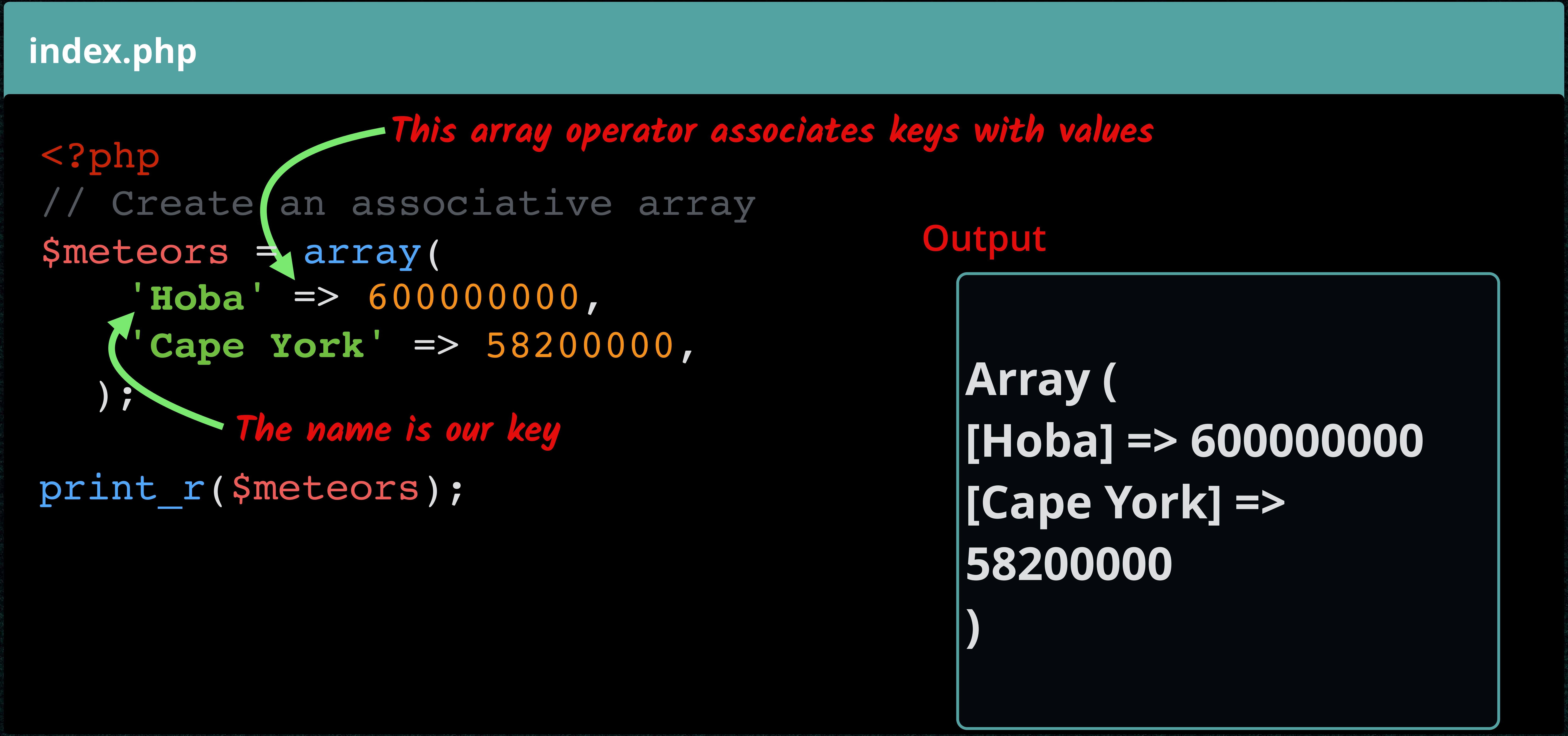
Storing Even More Data in an Array

What if we want to store more information about the meteorite?

Name	Weight	Location	Year
Hoba	600000000	-19.58333, 17.91667	1920
Cape York	58200000	76.13333, -64.93333	1818
Campo del Cielo	50000000	-27.46667, -60.58333	1576
Canyon Diablo	30000000	35.05, -111.03333	1891

Associative vs. Index Arrays

Associative arrays allow us to use strings as the key.



index.php

```
<?php
// Create an associative array
$meteors = array(
     'Hoba' => 600000000,
     'Cape York' => 58200000,
   );

print_r($meteors);
```

This array operator associates keys with values

The name is our key

Output

```
Array (
[Hoba] => 600000000
[Cape York] =>
58200000
)
```

Accessing an Item in the Array

Instead of the numerical index, we now use the string key for access.

index.php

```php
<?php

// Access our data.
echo $meteors['Hoba'];
echo $meteors['Cape York'];
```

Output

```
600000000
58020000
```

Appending a New Item

Using a string key, we can add a new item as well.

index.php

```php
<?php
// Add new meteorite data.
$meteors['Canyon Diablo'] = 30000000;

print_r($meteors);
```

Place the key inside of square brackets

Then set your value

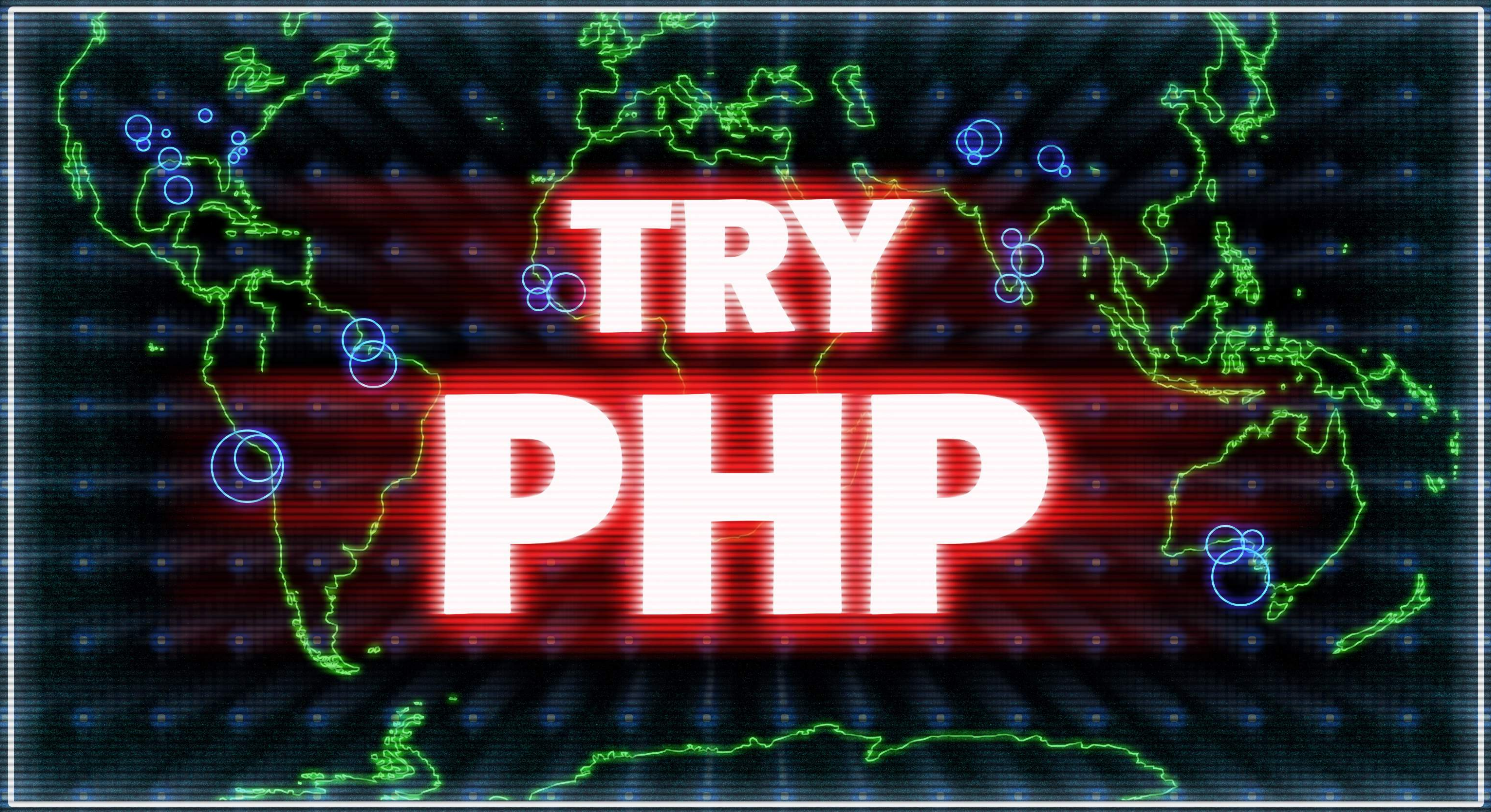
Output

```
Array(
  ...
  [Canyon Diablo]
    => 30000000
)
```

What Have We Learned?

Let’s have a quick review.

- Numerical indexed arrays
- Associative arrays
- Array creation with values
- Modification of array data


TRY
PHP

Level 2

Arrays

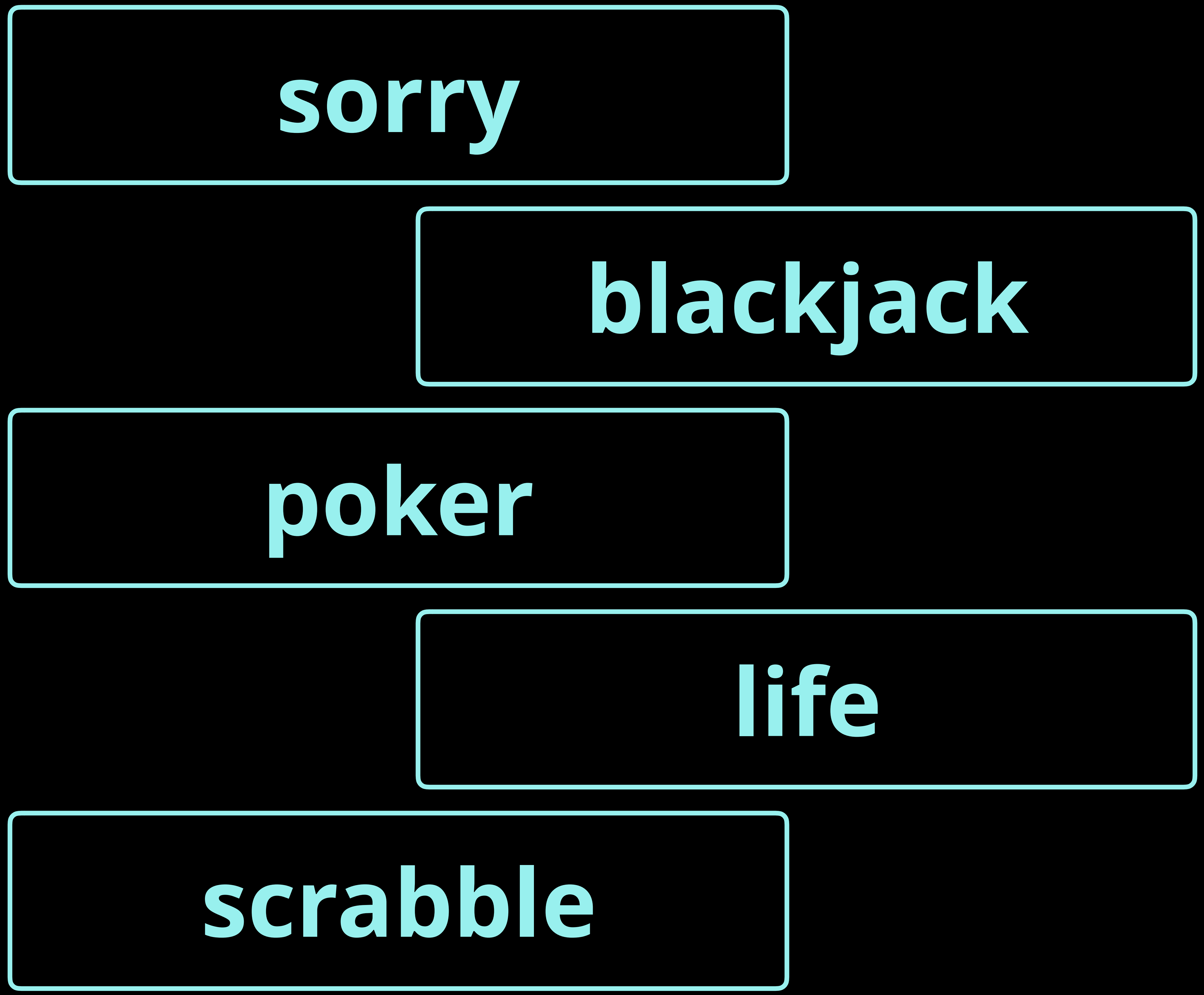
Multidimensional Arrays & Array Functions

An Array of Games

Arrays can help us organize data.

index.php

```php
<?php

$games = array(
    'sorry',
    'blackjack',
    'poker',
    'life',
    'scrabble',
);
```



Groups of Games

How can we better organize our list of games?

sorry

blackjack

poker

life

scrabble

Imagining Two Groups

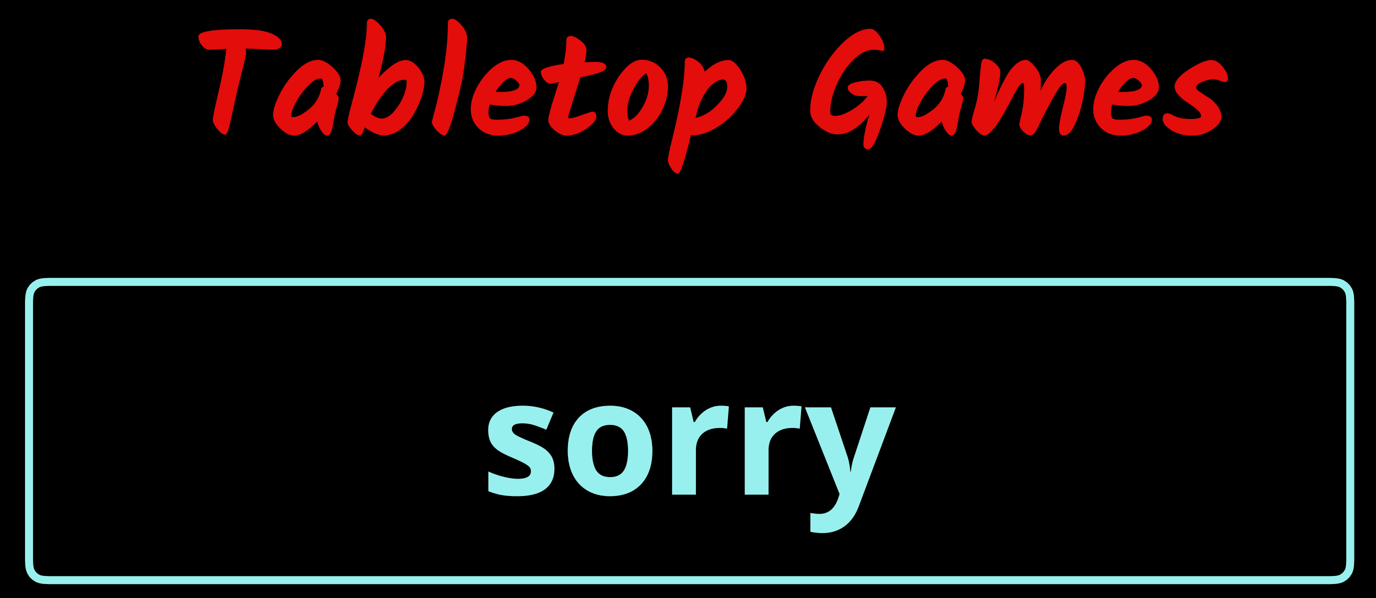
Splitting our games into two groups can help us sort and recall the data.

Tabletop Games	Card Games
sorry	blackjack
life	poker
scrabble	

Another Dimension

index.php

```php
<?php
$games = array(
    'tabletop' => 'sorry'
);
```



Tabletop Games

sorry

Another Dimension

index.php

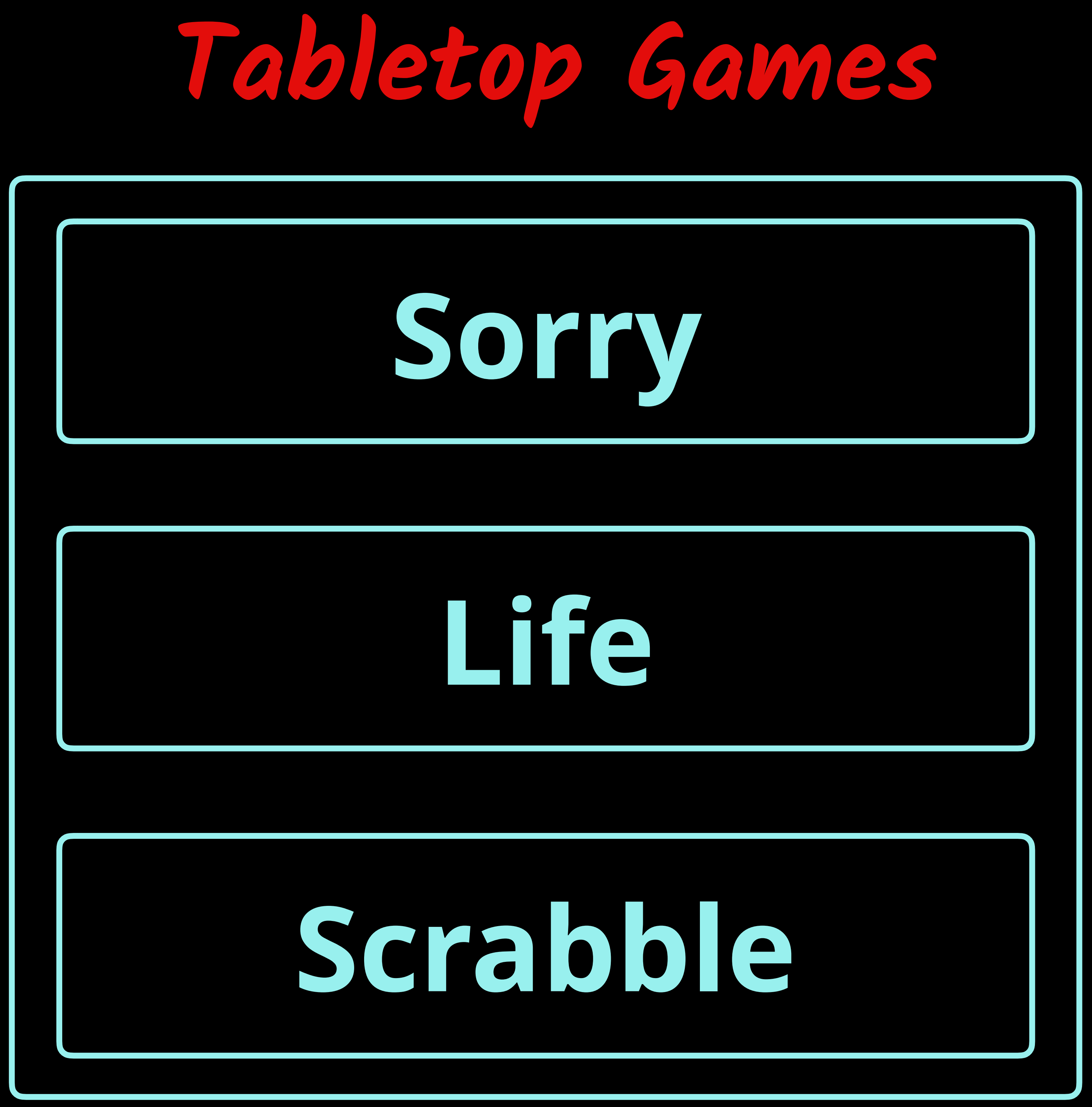
```php
<?php
$games = array(
   'tabletop' => array()
);
```

Tabletop Games

Another Dimension

index.php

```php
<?php
$games = array(
    'tabletop' => array(
        'sorry',
        'life',
        'scrabble',
    ),
);
```



Tabletop Games

Sorry

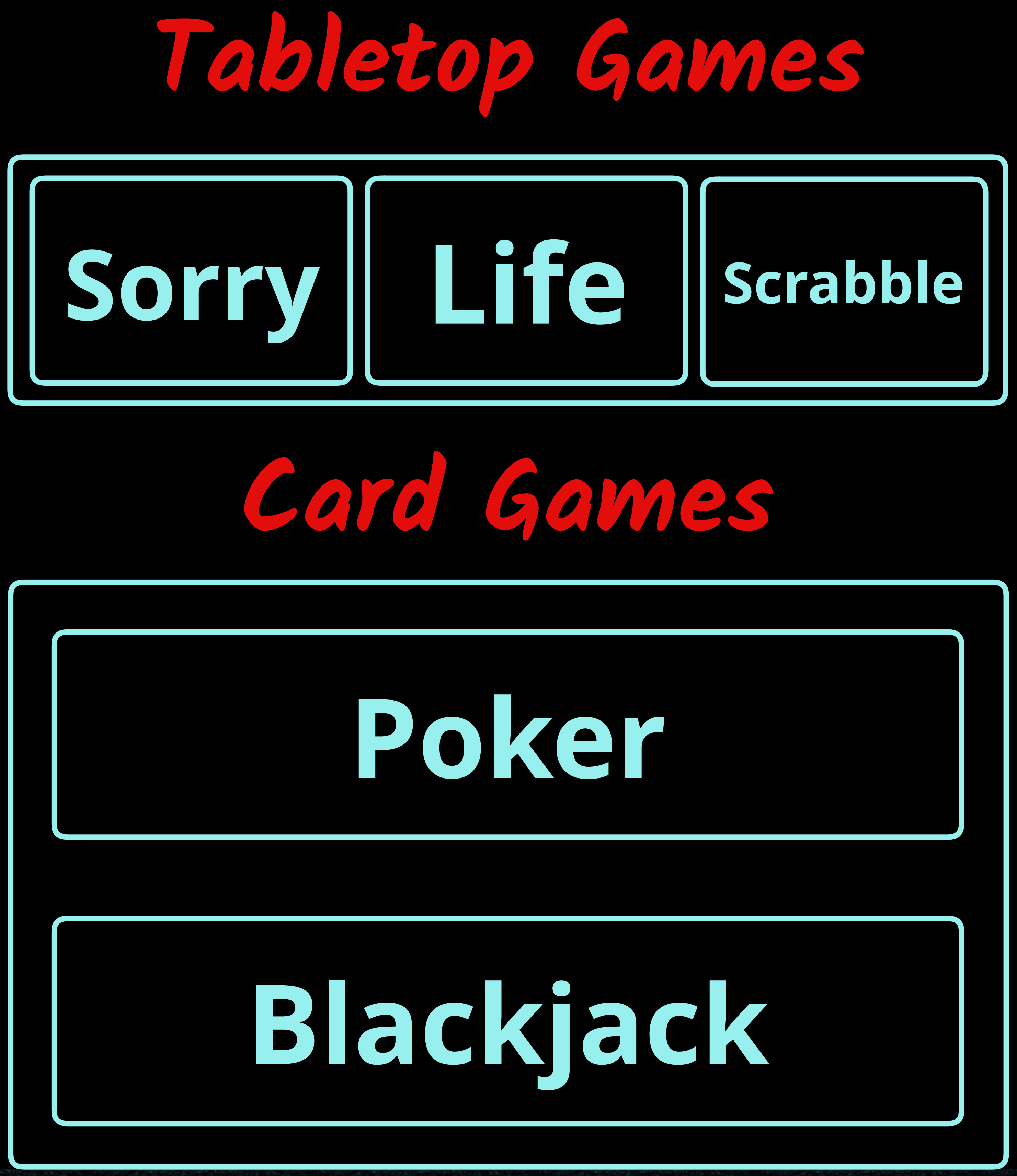
Life

Scrabble

Another Dimension

index.php

```php
<?php
$games = array(
    'tabletop' => array(
        'sorry',
        'life',
        'scrabble',
    ),
    'card' => array(
        'poker',
        'blackjack',
    ),
);
```



Tabletop Games

Sorry

Life

Scrabble

Card Games

Poker

Blackjack

Array Inspection

If we print_r our $games array, you can see the multidimensional structure.

```php
<?php
print_r($games);
```

Output

```
Array(
  [tabletop] => Array(
    [0] => sorry
    [1] => life
    [2] => scrabble
  [card] => Array(
    [0] => poker
    [1] => blackjack
)
```

Accessing Data

By using the array’s key, we can see the array value.

index.php

```php
<?php

print_r($games['tabletop']);
```

Output

```
Array (
[0] => sorry
[1] => life
[2] => scrabble
)
```

Accessing Data

By using the array's key, we can see the array value.

index.php

```php
<?php

print_r($games['tabletop']);

print_r($games['card']);
```

Output

```
Array (
[0] => poker
[1] => blackjack
)
```

Accessing Data

By using the array's key, we can see the array value.

index.php

```php
<?php

print_r($games['tabletop']);

print_r($games['card']);

echo $games['tabletop'][0];
```

Output

```
sorry
```

Modifying the Data

Instead of single item access, we can use the same methods to change a value.

index.php

```php
<?php

$games['card'][0] = 'rummy';

print_r($games['card']);
```

Output

```
Array (
[0] => rummy
[1] => blackjack
)
```

Array Functions: count

This function lets us count all the items in an array.

index.php

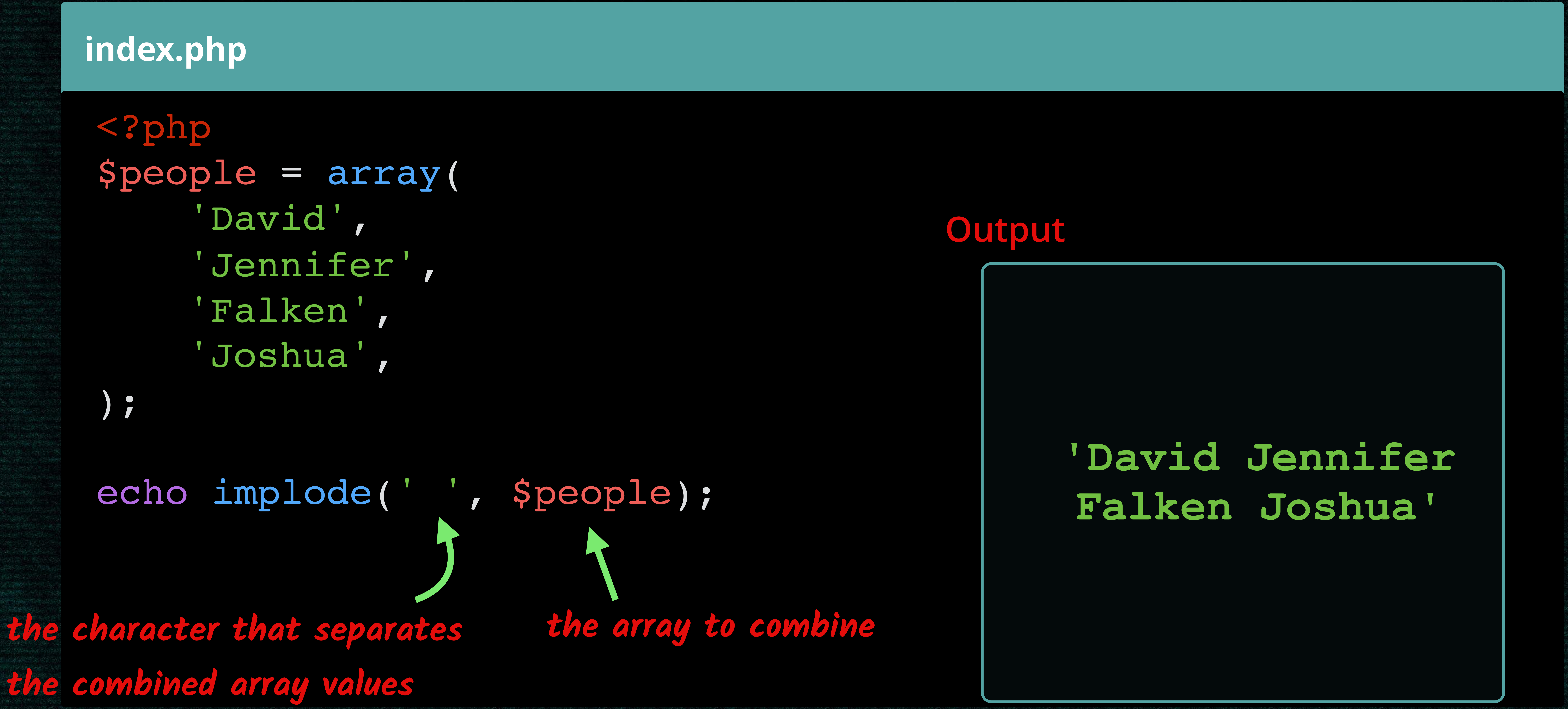
```php
<?php
$people = array(
    'David',
    'Jennifer',
    'Falken',
    'Joshua',
);

echo count($people);
```

Output

```
4
```

Array Functions: implode

implode **joins all elements of the array into a string.**



index.php

```php
<?php
$people = array(
    'David',
    'Jennifer',
    'Falken',
    'Joshua',
);

echo implode(' ', $people);
```

the character that separates

the array to combine

the combined array values

Output

```
'David Jennifer
Falken Joshua'
```

Array Functions: shuffle

shuffle changes the array in place to a random order.

index.php

```php
<?php
$people = array(
    'David',
    'Jennifer',
    'Falken',
    'Joshua',
);

// Randomize the array.
shuffle($people);

echo implode(' ', $people);
```

Output

```
'Jennifer David
Joshua Falken'
```

Array Functions: asort

asort will sort the array values, in place, in alphabetical order.

index.php

```php
<?php
$people = array('David','Jennifer','Falken','Joshua');

// Sort the array alphabetically.
asort($people);

echo implode(' ', $people);
```

Output

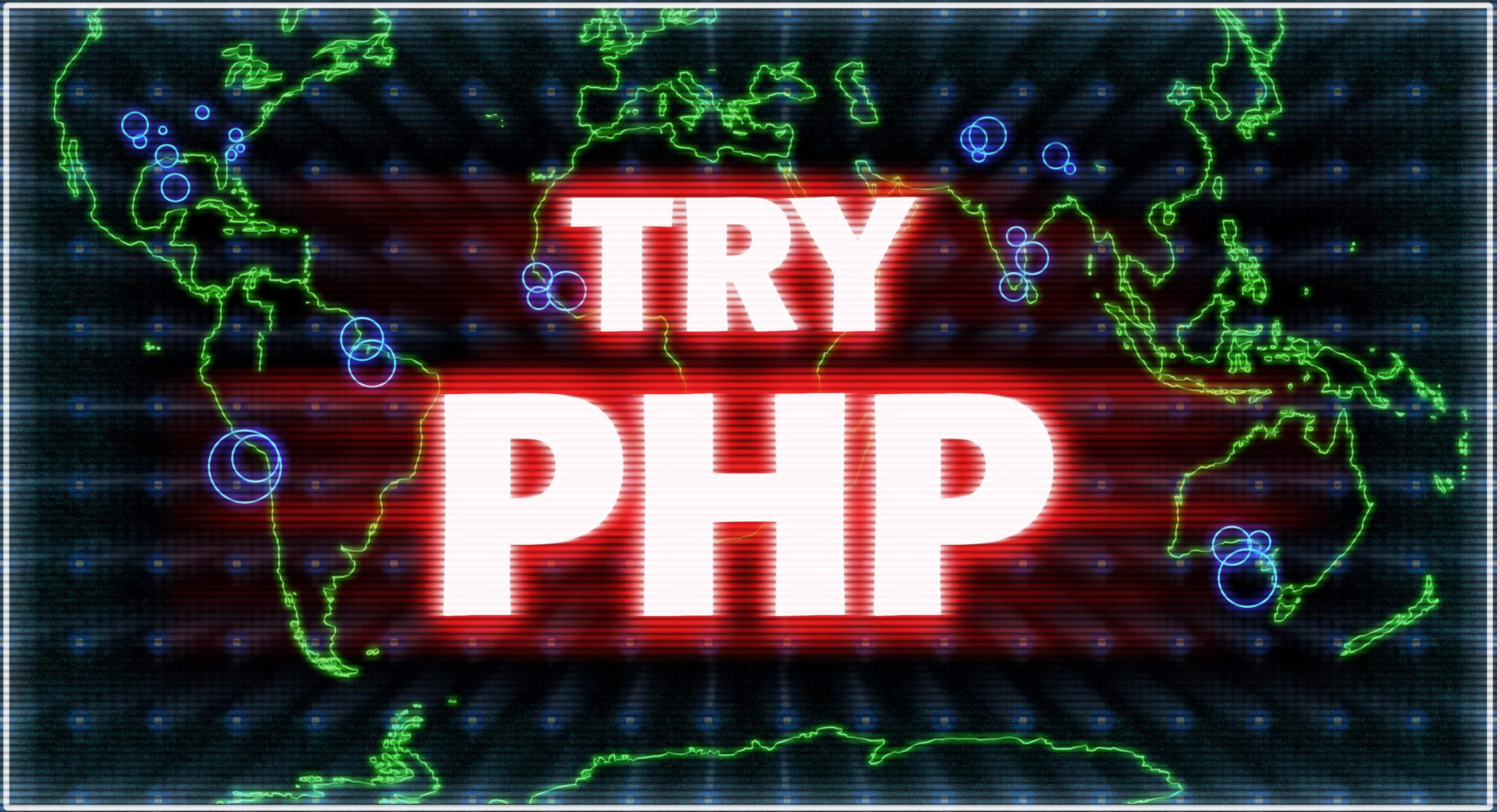
```
 'David Falken
Jennifer Joshua'
```

What Have We Learned?

Let’s have a quick review.

- Multidimensional array basics
- Some simple array functions
  - shuffle
  - implode
  - asort
  - count

And Now, It’s Challenge Time!



TRY PHP

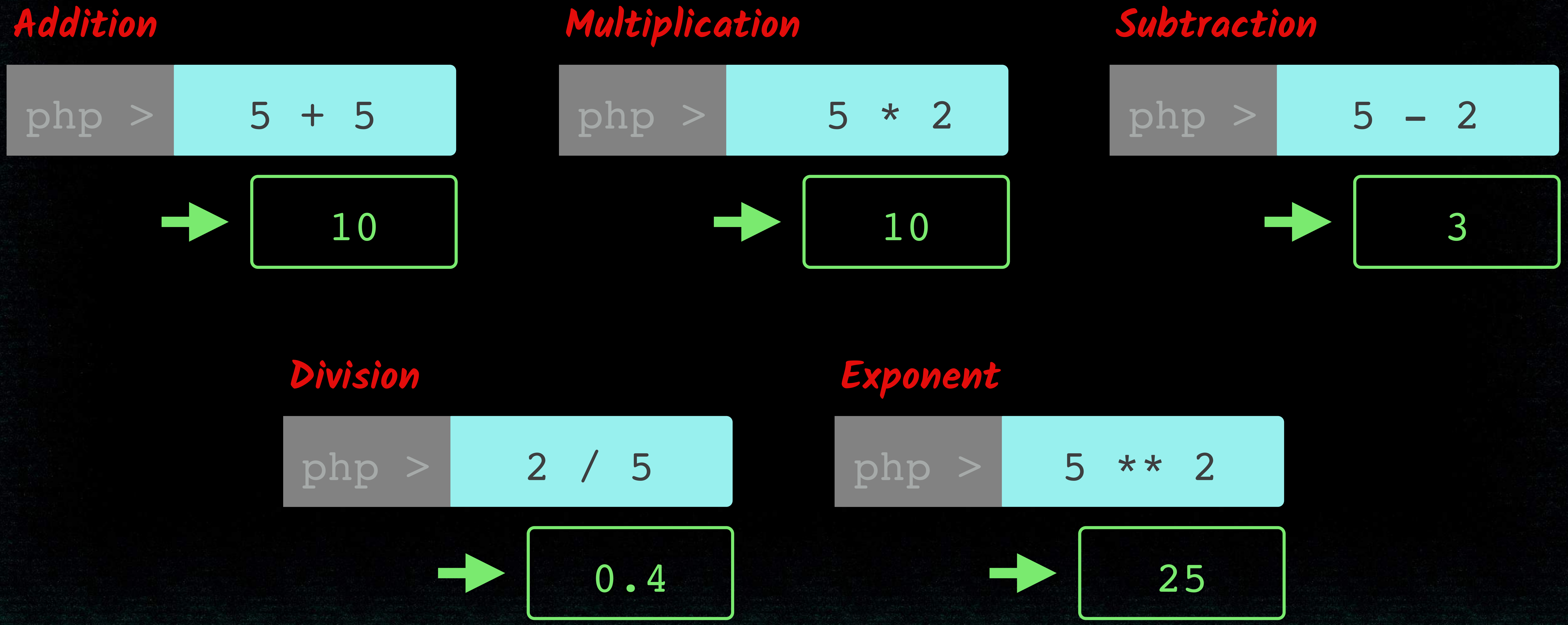
Level 3

Conditionals & Operators

What If? Now What? What Else?

Arithmetic Operators

These are some of the arithmetic operators available to us in PHP.



Addition

```
php > 5 + 5
10
```

Multiplication

```
php > 5 * 2
10
```

Subtraction

```
php > 5 - 2
3
```

Division

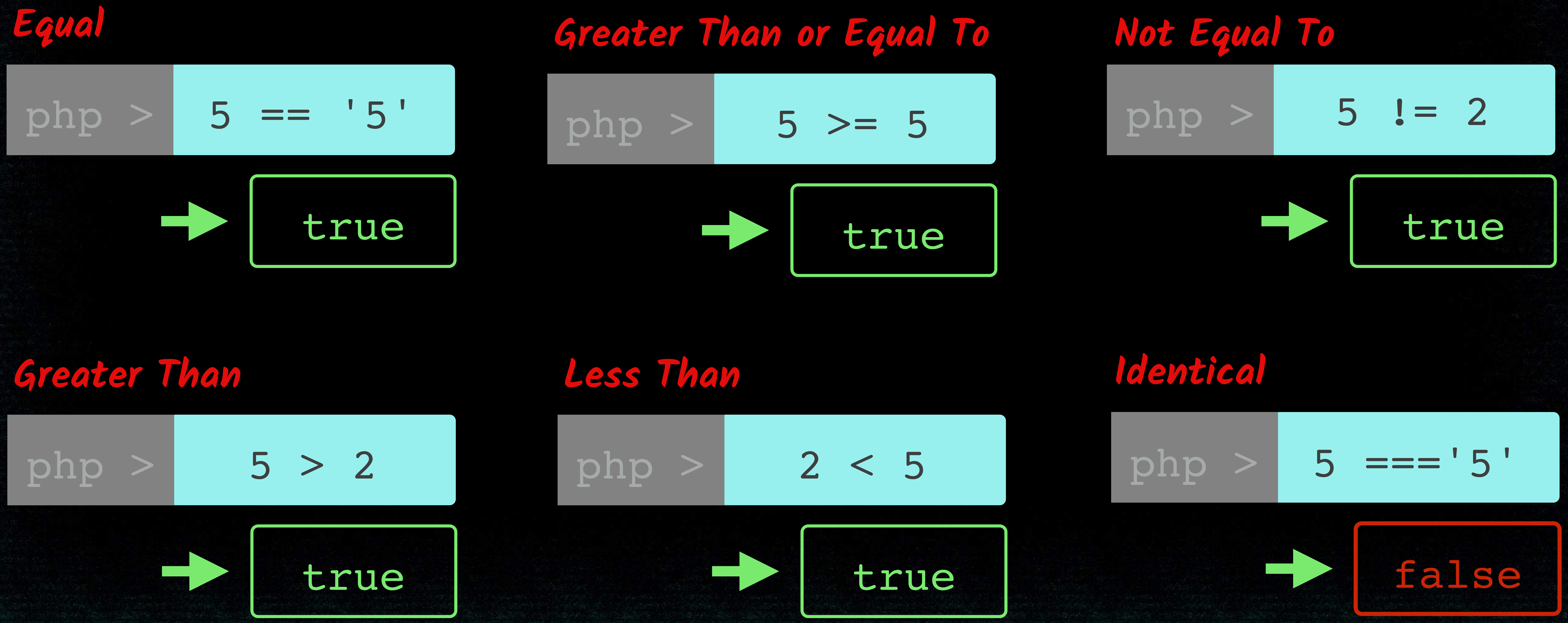
```
php > 2 / 5
0.4
```

Exponent

```
php > 5 ** 2
25
```

Comparison Operators

These are some of the comparison operators available to us in PHP.



Equal

```
php > 5 == '5'
true
```

Greater Than or Equal To

```
php > 5 >= 5
true
```

Not Equal To

```
php > 5 != 2
true
```

Greater Than

```
php > 5 > 2
true
```

Less Than

```
php > 2 < 5
true
```

Identical

```
php > 5 ==='5'
false
```

Identical Comparison Operator

To be identical, the items must be of the same type and value.

Identical

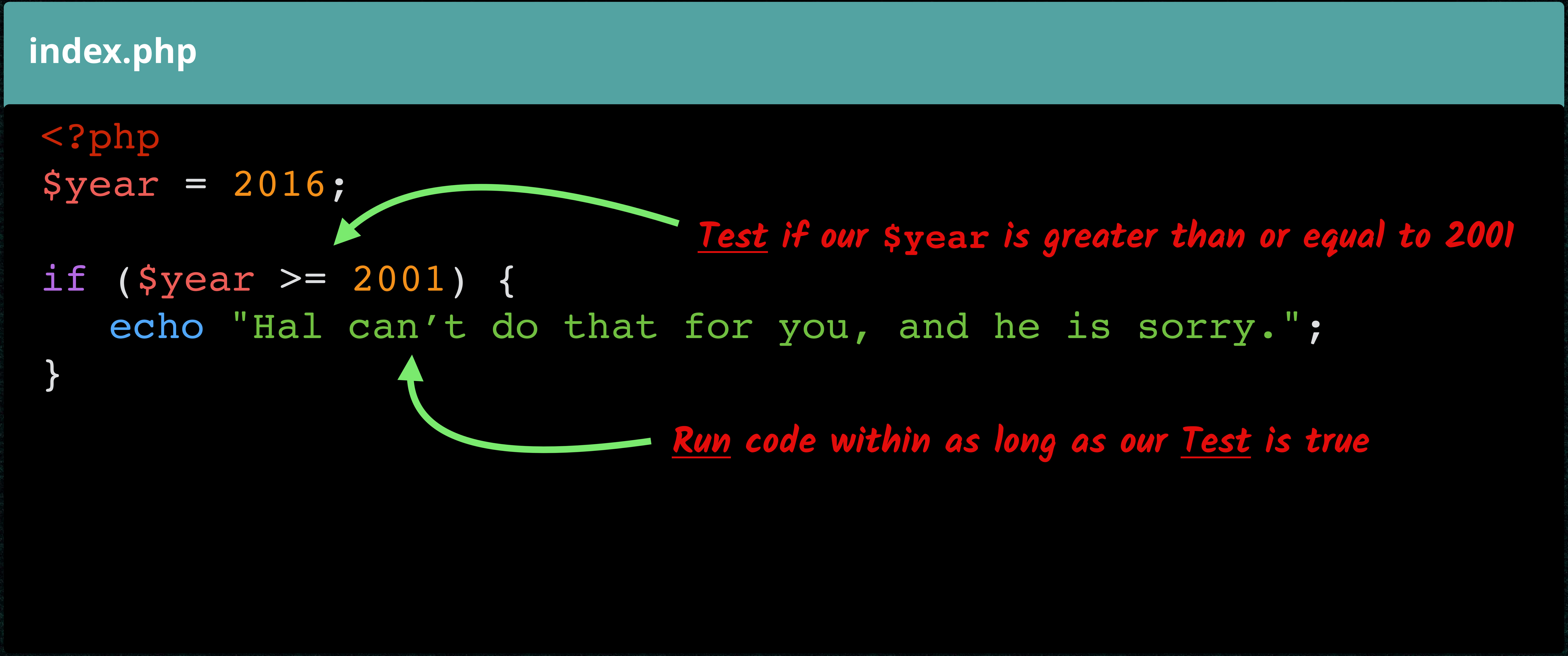
```
php > 5 ==='5'
```

```
false
```

```
5    // is integer data
'5'  // is string data
```

Control Flow

The if statement allows us to execute code based on a condition.



index.php

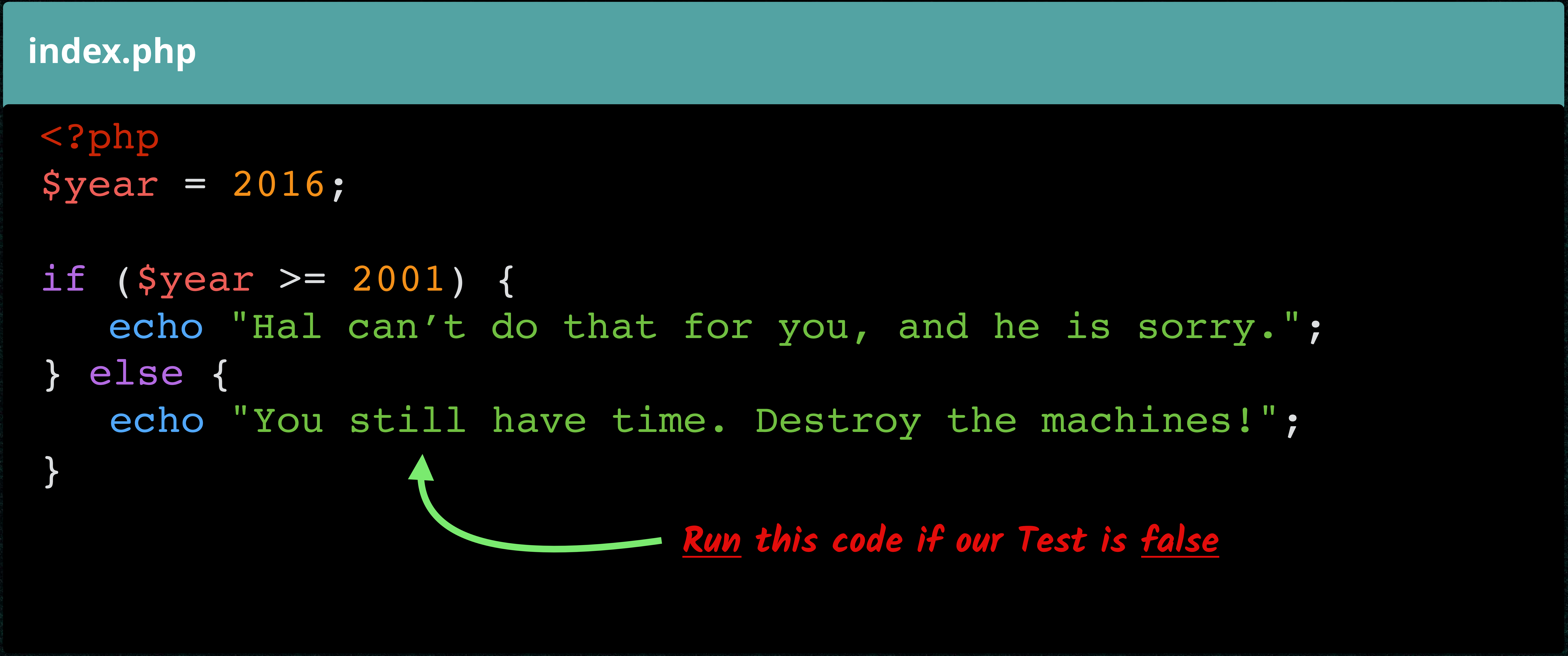
```php
<?php
$year = 2016;

if ($year >= 2001) {
   echo "Hal can’t do that for you, and he is sorry.";
}
```

Test if our $year is greater than or equal to 2001

Run code within as long as our Test is true

Default Condition

The else statement allows us to run code when the if returns false.



index.php

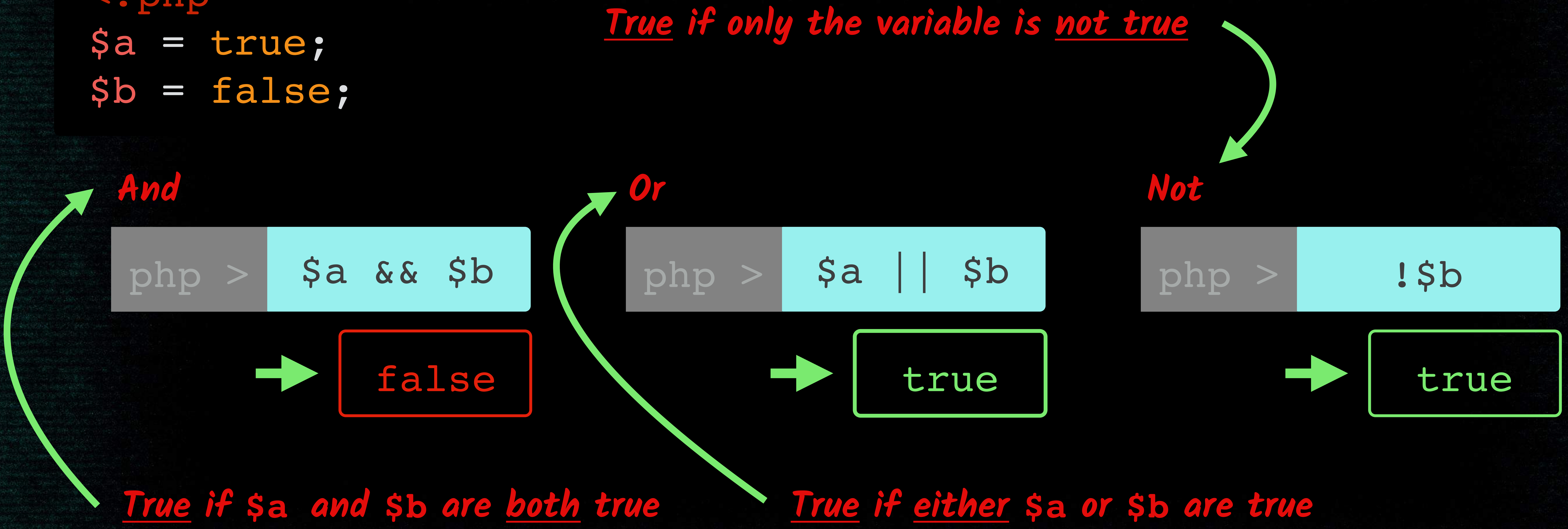
```php
<?php
$year = 2016;

if ($year >= 2001) {
   echo "Hal can’t do that for you, and he is sorry.";
} else {
   echo "You still have time. Destroy the machines!";
}
```

Run this code if our Test is false

Logical Operators

These are some of the logical operators available to us in PHP.

```
<?php
$a = true;
$b = false;
```



True if only the variable is not true

And

```
php > $a && $b
```

false

Or

```
php > $a || $b
```

true

Not

```
php > !$b
```

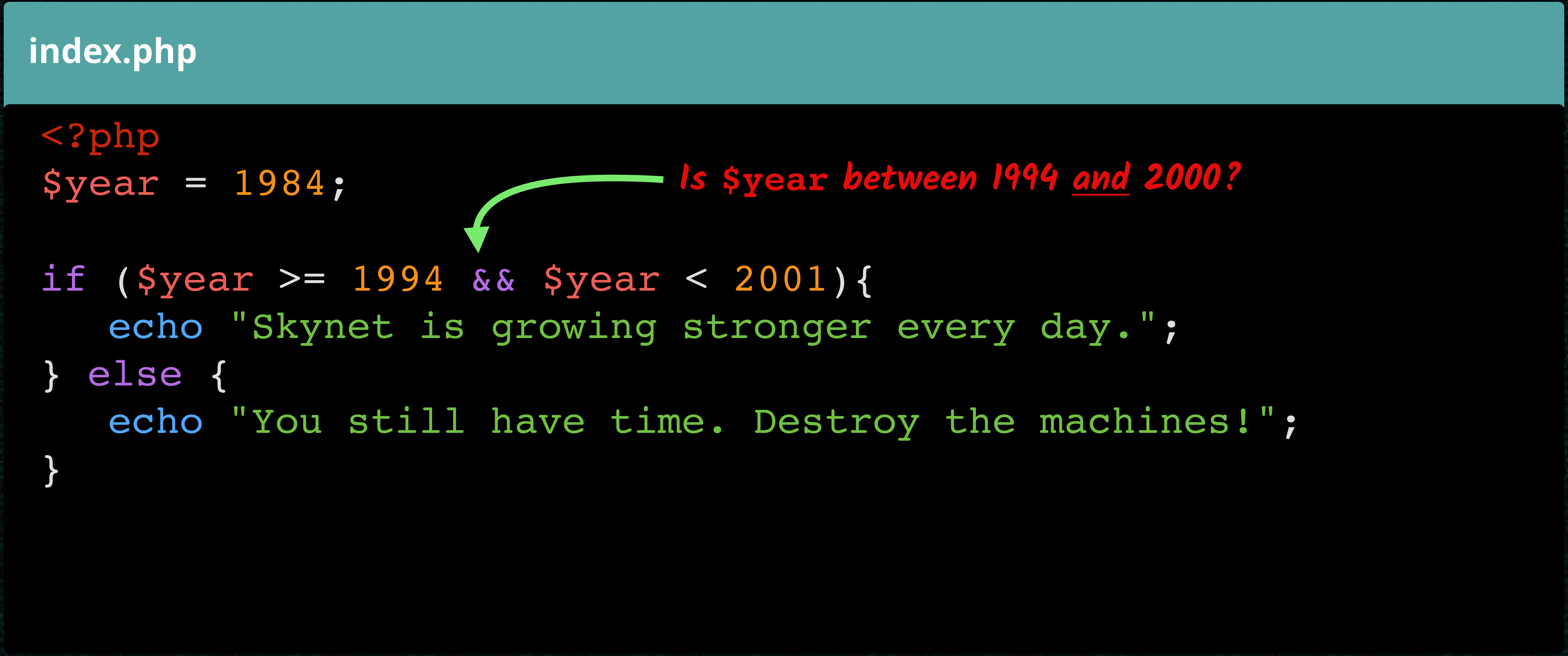
true

True if $a and $b are both true

True if either $a or $b are true

Testing Multiple Conditions

Using the logical operator and, we can test to see if multiple conditions are true.



index.php

```php
<?php
$year = 1984;

if ($year >= 1994 && $year < 2001){
   echo "Skynet is growing stronger every day.";
} else {
   echo "You still have time. Destroy the machines!";
}
```

Is $year between 1994 and 2000?

Multiple if Statements

The elseif statement allows us to have multiple conditions.

index.php

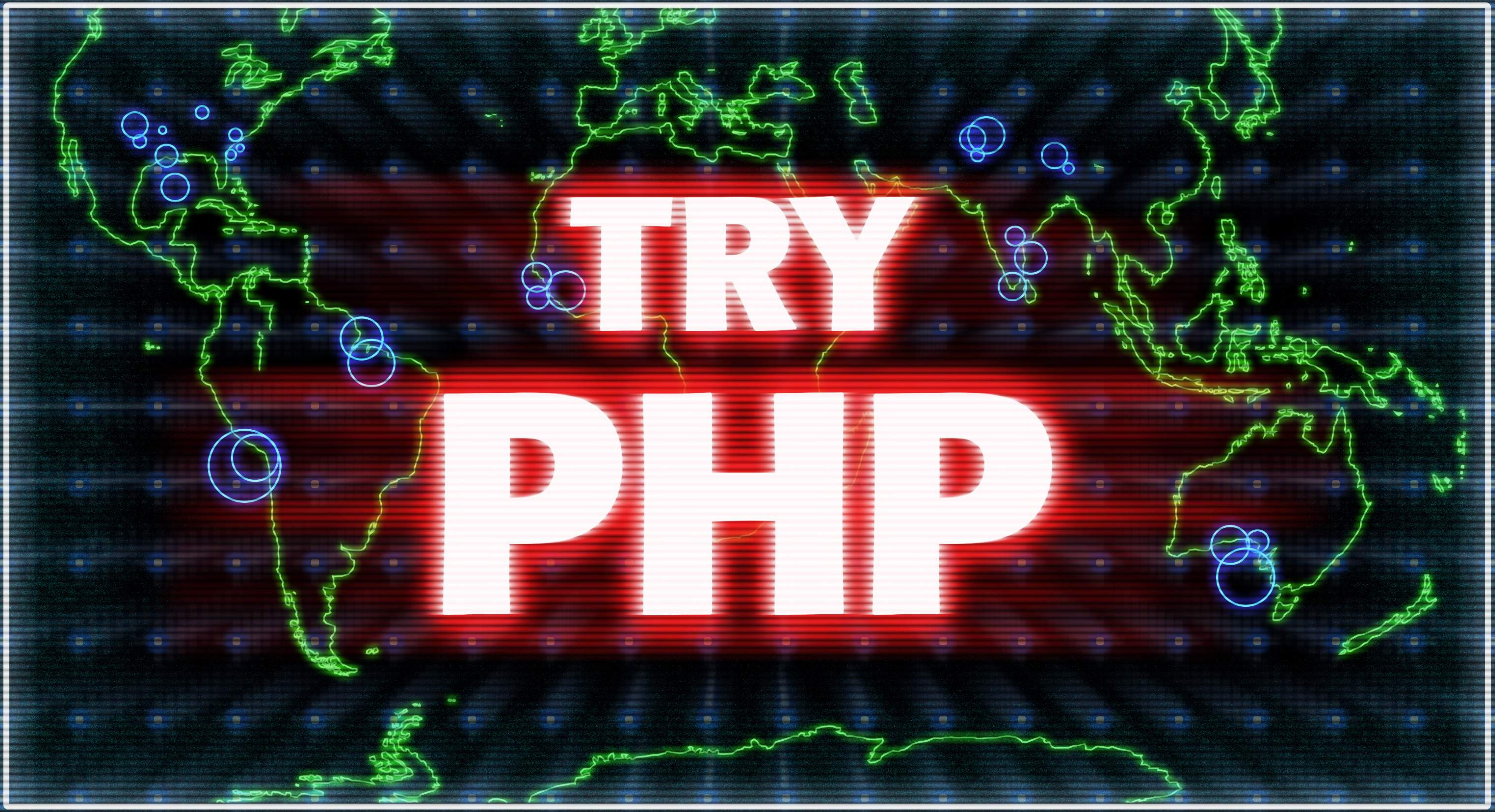
```php
<?php
$year = 1984;

if ($year >= 2001) {
   echo "Hal can’t do that for you, and he is sorry.";
} elseif ($year >= 1984) {
   echo "Eurasia has fallen! Rejoice with Big Brother.";
} else {
   echo "You still have time. Destroy the machines!";
}
```

Test this if the first condition is false

What Have We Learned?

Let’s have a quick review.

- Comparison operators
- Arithmetic operators
- if, if-else, else comparisons
- Logical operators


TRY
PHP

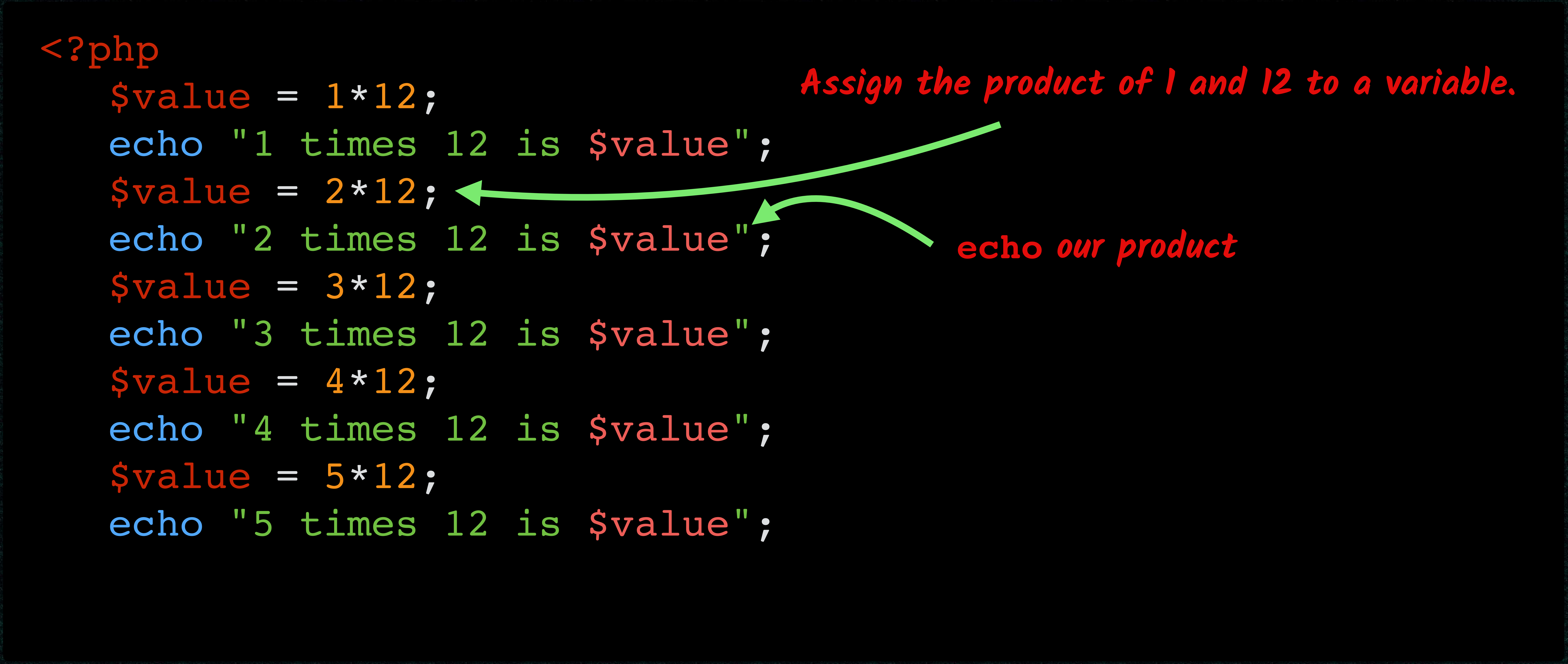
Level 4

Loops

Cycle Through All the Data

Don’t Repeat Yourself

The DRY (or “Don’t Repeat Yourself”) method helps us keep our code efficient.



```
<?php
   $value = 1*12;
   echo "1 times 12 is $value";
   $value = 2*12;
   echo "2 times 12 is $value";
   $value = 3*12;
   echo "3 times 12 is $value";
   $value = 4*12;
   echo "4 times 12 is $value";
   $value = 5*12;
   echo "5 times 12 is $value";
```

Assign the product of 1 and 12 to a variable.

echo our product

while **Loops**

Now let’s initialize, test, and increment.

Initialize an integer variable and set it to 1

```
<?php
$i = 1;

while( $i <= 12 ) {
   $value = $i * 12;
   echo "$i times 12 is $value";
   $i++;
}
```

while **Loops**

Now let’s initialize, test, and increment.

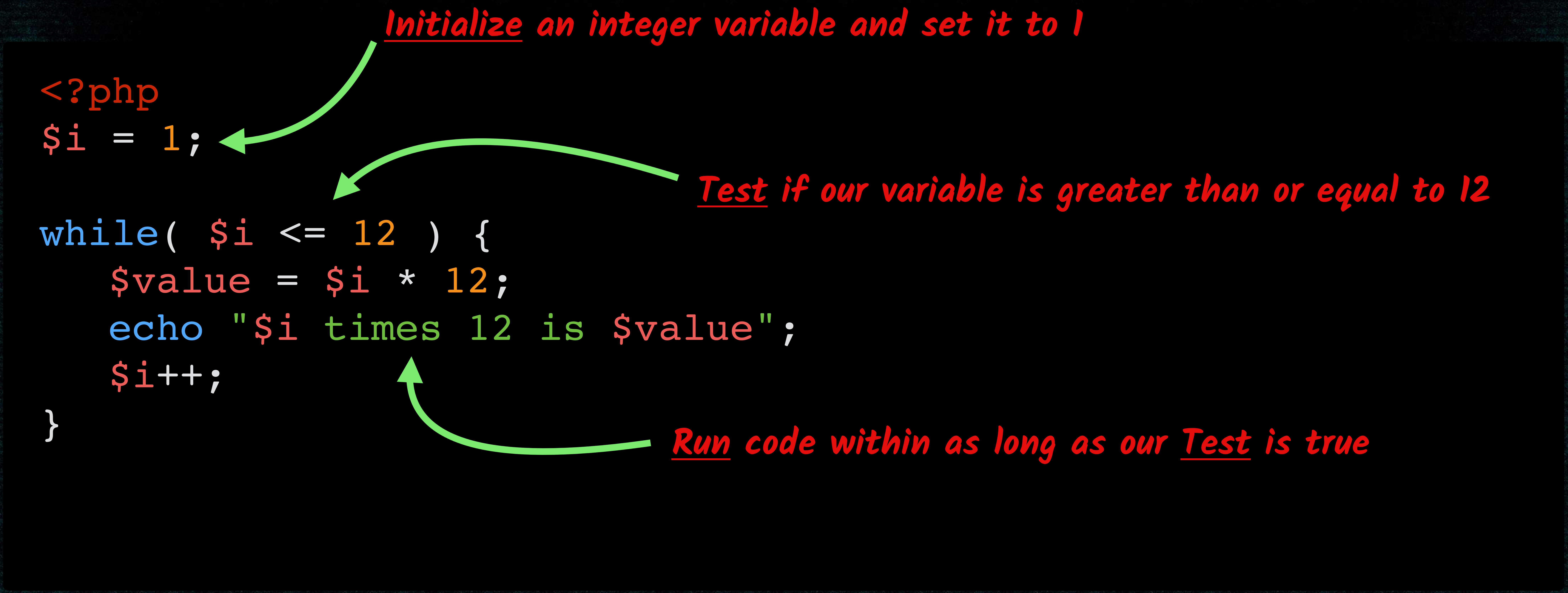
Initialize an integer variable and set it to 1

```
<?php
$i = 1;

while( $i <= 12 ) {
   $value = $i * 12;
   echo "$i times 12 is $value";
   $i++;
}
```

Test if our variable is greater than or equal to 12

while **Loops**

Now let’s initialize, test, and increment.



Initialize an integer variable and set it to 1

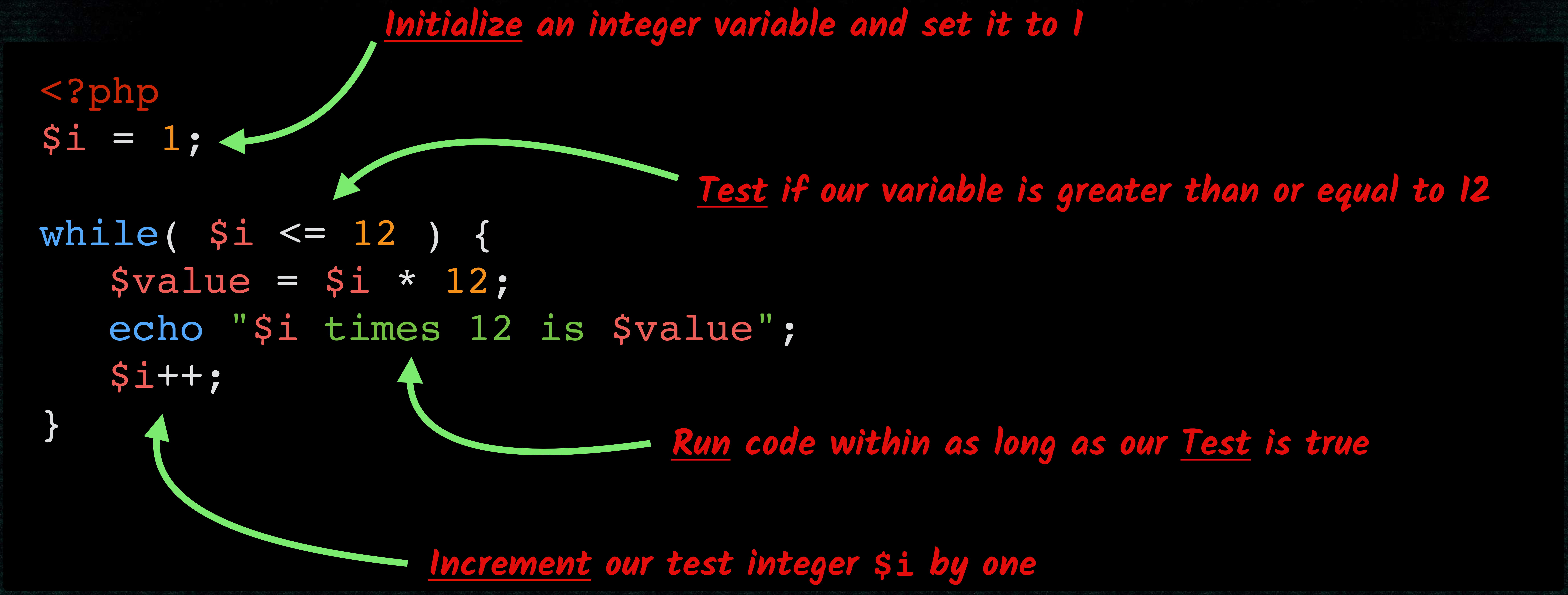
```php
<?php
$i = 1;

while( $i <= 12 ) {
   $value = $i * 12;
   echo "$i times 12 is $value";
   $i++;
}
```

Test if our variable is greater than or equal to 12

Run code within as long as our Test is true

while **Loops**

Now let’s initialize, test, and increment.



```php
<?php
$i = 1;

while( $i <= 12 ) {
    $value = $i * 12;
    echo "$i times 12 is $value";
    $i++;
}
```

Initialize an integer variable and set it to 1

Test if our variable is greater than or equal to 12

Run code within as long as our Test is true

Increment our test integer $i by one

while **Loops**

Now let’s initialize, test, and increment.

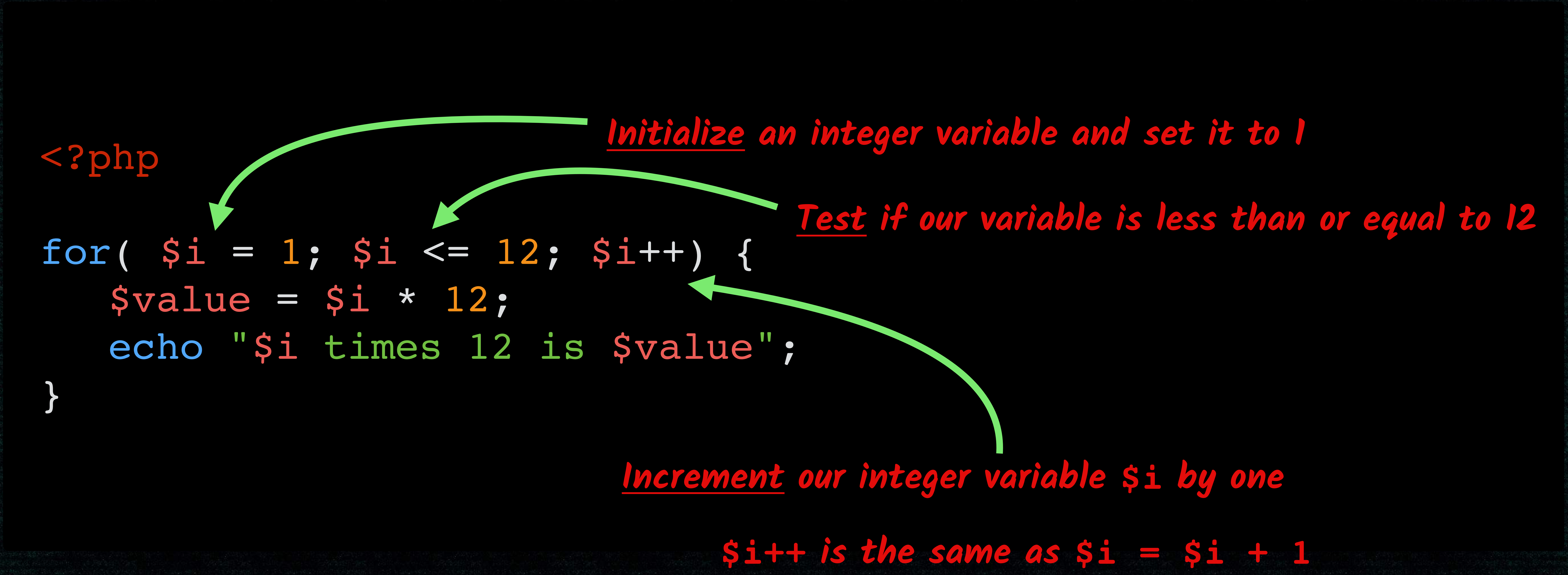
```
<?php
$i = 1;

while( $i <= 12 ) {
   $value = $i * 12;
   echo "$i times 12 is $value";
   $i++;
}
```

Output

```
1 times 12 is 12
2 times 12 is 24
3 times 12 is 36
…
10 times 12 is 120
11 times 12 is 132
12 times 12 is 144
```

Using a for Loop

Now let’s initialize, test, and increment.



```
<?php

for( $i = 1; $i <= 12; $i++) {
   $value = $i * 12;
   echo "$i times 12 is $value";
}
```

Initialize an integer variable and set it to 1

Test if our variable is less than or equal to 12

Increment our integer variable $i by one

$i++ is the same as $i = $i + 1

Using a for Loop

Now let’s initialize, test, and increment.

```php
<?php

for( $i = 1; $i <= 12; $i++) {
   $value = $i * 12;
   echo "$i times 12 is $value";
}
```

Output

```
1 times 12 is 12
2 times 12 is 24
3 times 12 is 36
...
10 times 12 is 120
11 times 12 is 132
12 times 12 is 144
```

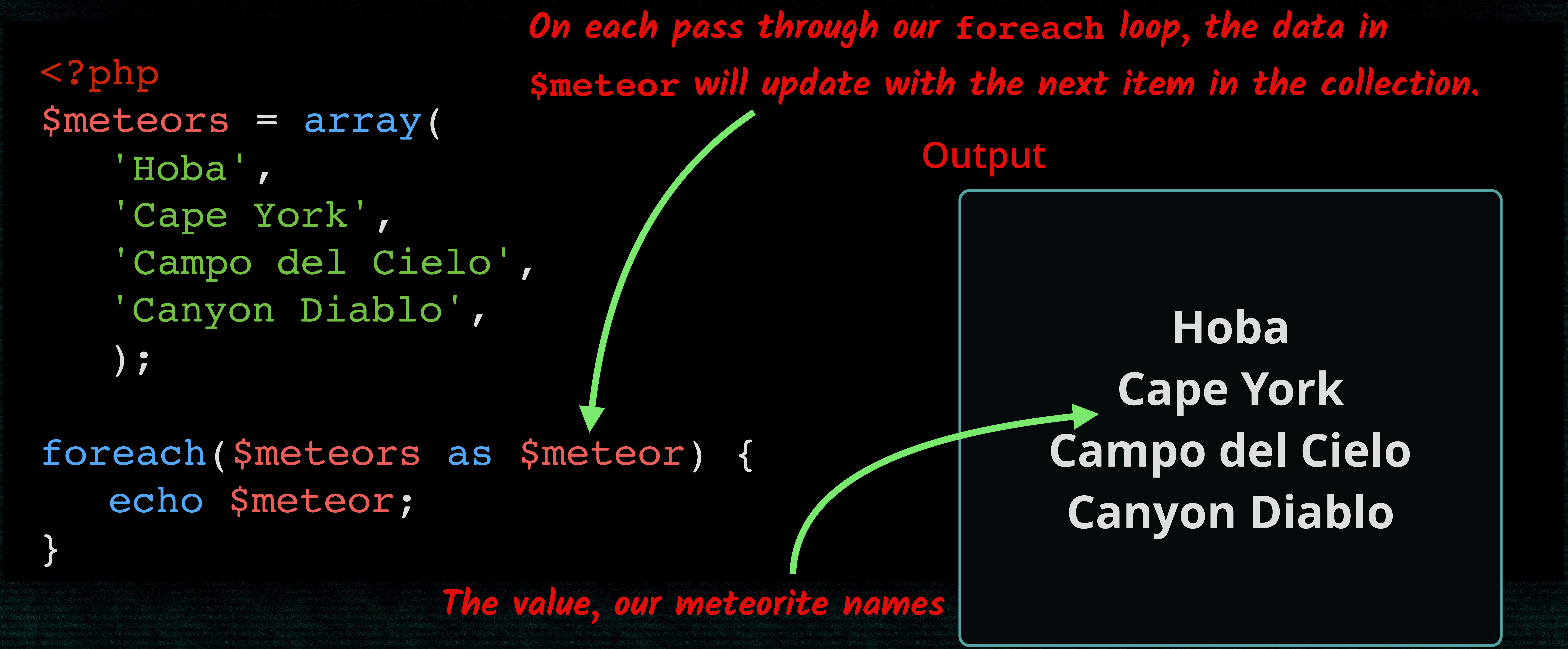
The Simple Meteorite Array

How else could we extract each item in the array other than direct access?

```php
<?php
$meteors = array(
    'Hoba',
    'Cape York',
    'Campo del Cielo',
    'Canyon Diablo',
    );
```

Looping Access to the Array

The foreach and as will allow us to cycle through each item in our array.



On each pass through our `foreach` *loop, the data in* `$meteor` *will update with the next item in the collection.*

```
<?php
$meteors = array(
    'Hoba',
    'Cape York',
    'Campo del Cielo',
    'Canyon Diablo',
    );

foreach($meteors as $meteor) {
    echo $meteor;
}
```

Output

```
Hoba
Cape York
Campo del Cielo
Canyon Diablo
```

The value, our meteorite names

Associative Meteorite Array

What would happen if we ran this array through our existing foreach loop?

```php
<?php
$meteors = array(
   'Hoba' => 600000000,
   'Cape York' => 58200000,
   'Campo del Cielo' => 50000000,
   'Canyon Diablo' => 30000000,
   );
```

Looping Through an Associative Array

What would happen if we ran this array through our existing foreach loop?

```php
<?php
$meteors = array(
    'Hoba' => 600000000,
    'Cape York' => 58200000,
    'Campo del Cielo' => 50000000,
    'Canyon Diablo' => 30000000,
    );

foreach($meteors as $meteor) {
    echo $meteor;
}
```

Output

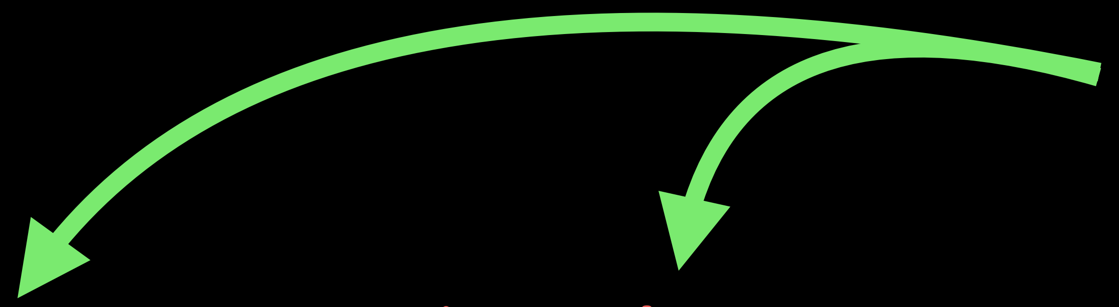
```
600000000
58200000
50000000
30000000
```

The value is our meteorite weight!

How Can We Access the Key and Value?

We can use the array operator => to set up the key and value variables.

```php
<?php
$meteors = array(
    'Hoba' => 600000000,
    'Cape York' => 58200000,
    'Campo del Cielo' => 50000000,
    'Canyon Diablo' => 30000000,
    );
```



$name and $weight will change values with each pass

```php
foreach($meteors as $name => $weight){

}
```

How Can We Access the Key and Value?

We can use the object operator => to set up the key and value variables.

```php
<?php
$meteors = array(
   'Hoba' => 600000000,
   'Cape York' => 58200000,
   'Campo del Cielo' => 50000000,
   'Canyon Diablo' => 30000000,
   );

foreach($meteors as $name => $weight){
   echo "$name weighs $weight grams.";
}
```

Output

```
Hoba weighs
600000000 grams.
...
Canyon Diablo weighs
30000000 grams.
```

A Complete Picture

```php
<?php
$meteors = array(
    'Hoba' => 600000000,
    'Cape York' => 58200000,
    'Campo del Cielo' => 50000000,
    'Canyon Diablo' => 30000000,
    );
$epic = 600000000; // 600 million grams
$huge = 50000000; // 50 million grams
?>
```

A Complete Picture

```php
<?php
$meteors = array(
   'Hoba' => 600000000,
   'Cape York' => 58200000,
   'Campo del Cielo' => 50000000,
   'Canyon Diablo' => 30000000,
   );
$epic = 600000000; // 600 million grams
$huge = 50000000; // 50 million grams
foreach ($meteors as $name => $weight) {

}
?>
```

A Complete Picture

```php
<?php
$meteors = array(
   'Hoba' => 600000000,
   'Cape York' => 58200000,
   'Campo del Cielo' => 50000000,
   'Canyon Diablo' => 30000000,
   );
$epic = 600000000; // 600 million grams
$huge = 50000000; // 50 million grams
foreach ($meteors as $name => $weight) {
    if ($weight >= $epic) {
        echo 'You have found an epic meteorite!<br>';
        echo 'Your meteorite\'s name is ' . $name . '<br>';
    }
}
?>
```

A Complete Picture

```php
<?php
$meteors = array('Hoba' => 600000000, ...);
$epic = 600000000; // 600 million grams
$huge = 50000000; // 50 million grams
foreach ($meteors as $name => $weight) {
    if ($weight >= $epic) {
        echo 'You have found an epic meteorite!<br>';
        echo 'Your meteorite\'s name is ' . $name . '<br>';
    } elseif ($weight >= $huge) {
        echo 'You have found a huge meteorite!<br>';
        echo 'Your meteorite\'s name is ' . $name . ‘<br>';
    }
}
?>
```

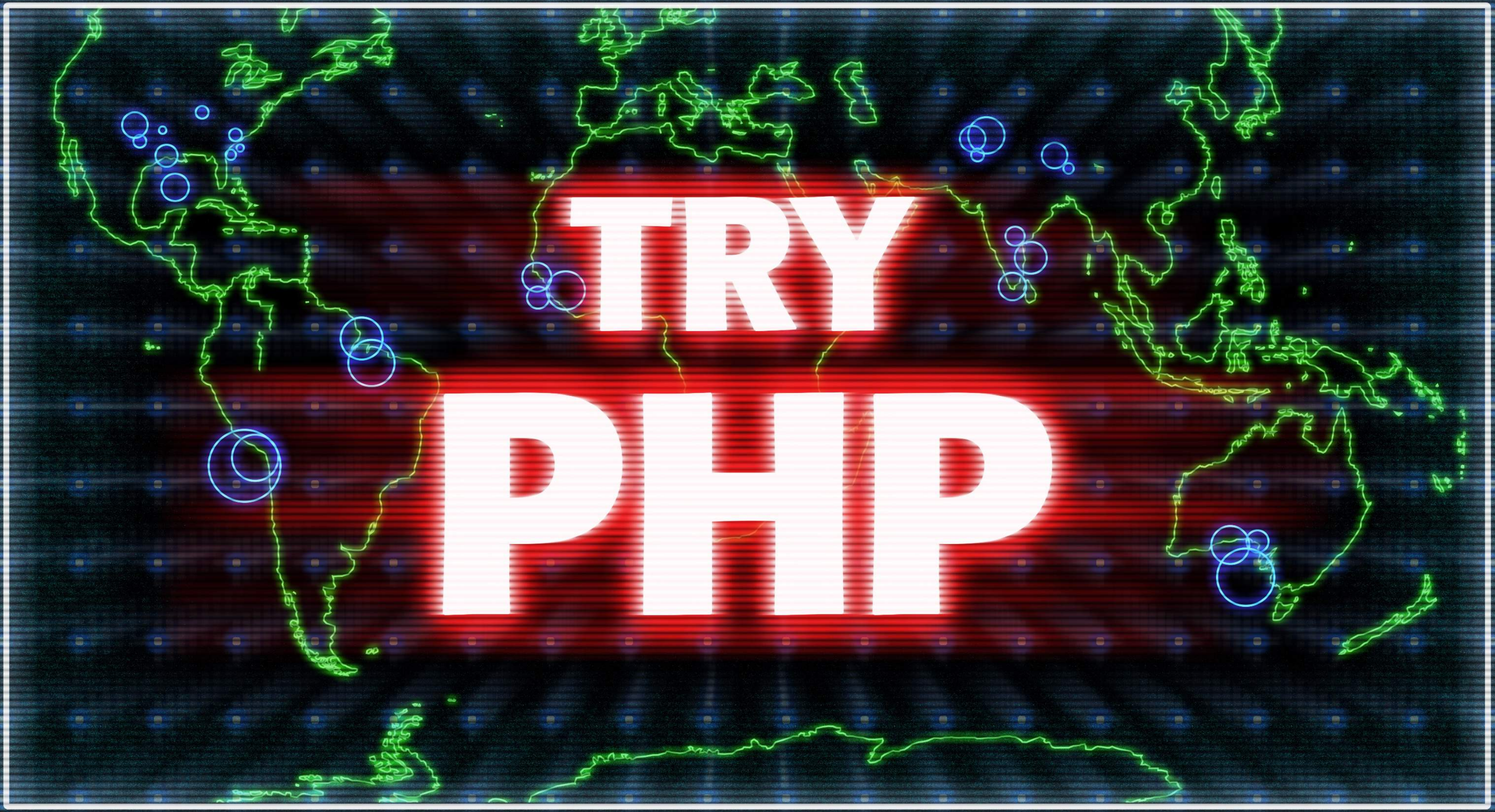
A Complete Picture

```php
<?php
$meteors = array('Hoba' => 600000000, ...);
$epic = 600000000; // 600 million grams
$huge = 50000000; // 50 million grams
foreach ($meteors as $name => $weight) {
  if ($weight >= $epic) {
    echo 'You have found an epic meteorite!<br>';
    echo 'Your meteorite\'s name is ' . $name . '<br>';
  } elseif ($weight >= $huge) {
    echo 'You have found a huge meteorite!<br>';
    echo 'Your meteorite\'s name is ' . $name . ‘<br>';
  } else {
    echo 'You have found a meteorite, awesome!<br>';
    echo 'Your meteorite\'s name is ' . $name . '<br>';
  }
}
?>
```

What Have We Learned?

Let’s have a quick review.

- while **loop**
- for **loop**
- foreach **loop**
- foreach **with key/value**
- **Combining loops and conditionals**



TRY

PHP