For everyone who contributed to the publication of this book

Table of Contents

List of Abbreviations

Abbreviation	Meaning
XSS	Cross-Site Scripting

INTRODUCTION TO BOOK

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in **1991** and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions: **Python 2** and **Python 3**. Both are quite different. In this book, the basics of the programming language: **python** and **php**.

This book can benefit anyone who wants to learn these two languages, and then it can be developed from his skill to learn more.


PHP Programming
Language

PHP PROGRAMMING LANGUAGE

PHP Introduction

PHP code is executed on the server.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- CSS
- JavaScript

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension "**.php**"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data

PHP PROGRAMMING LANGUAGE

- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data
- With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

PHP PROGRAMMING LANGUAGE

PHP Installation

What Do I Need?

To start using PHP, you can:

- Find a web host with **PHP** and **MySQL** support
- Install a web server on your own PC, and then install **PHP** and **MySQL**

Use a Web Host with PHP Support

If your server has activated support for PHP you do not need to do anything.

Just create some .php files, place them in your web directory, and the server will automatically parse them for you.

You do not need to compile anything or install any extra tools.

Because PHP is free, most web hosts offer PHP support.

Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

The official PHP website (PHP.net) has installation instructions for PHP: **http://php.net/manual/en/install.php**

PHP PROGRAMMING LANGUAGE

PHP Tutorial

➢ PHP Syntax

A **PHP script** is executed on the server, and the plain **HTML** result is sent back to the browser.

Basic PHP Syntax

A **PHP** script can be placed anywhere in the document.

A **PHP** script starts with **<?php and ends with ?>**:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is "**.php**".

A **PHP** file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

My first PHP page

Hello World!

Note: **PHP statements end with a semicolon (;).**

PHP PROGRAMMING LANGUAGE

PHP Case Sensitivity

In PHP, keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are **not case-sensitive**.

In the example below, all three echo statements below are equal and legal:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>
```

Hello World!
Hello World!
Hello World!

Note: However; all variable names are case-sensitive!

Look at the example below; only the first statement will display the value of the $color variable! This is because **$color**, **$COLOR**, and **$coLOR** are treated as three different variables:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>

</body>
</html>
```

My car is red
My house is
My boat is

➢ PHP Comments

Comments in PHP

A comment in **PHP** code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand your code
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting:

Example

Syntax for single-line comments:

```
<!DOCTYPE html>
<html>
<body>

<?php
// This is a single-line comment

# This is also a single-line comment
?>

</body>
</html>
```

Example

Syntax for multiple-line comments:

```
<!DOCTYPE html>
<html>
<body>

<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
?>

</body>
</html>
```

Example

Using comments to leave out parts of the code:

```
<!DOCTYPE html>
<html>
<body>

<?php
// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>

</body>
</html>
```

```
10
```

➢ PHP Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example

PHP PROGRAMMING LANGUAGE

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;

echo $txt;
echo "<br>";
echo $x;
echo "<br>";
echo $y;
?>

</body>
</html>
```

Hello world!
5
10.5

After the execution of the statements above, the variable **$txt** will hold the value Hello world! the variable **$x** will hold the value 5, and the variable **$y** will hold the value **10.5**.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

PHP PROGRAMMING LANGUAGE

Output Variables

The **PHP echo** statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt = "rose";
echo "I love $txt!";
?>

</body>
</html>
```

I love rose!

The following example will produce the same output as the example above:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt = "rose";
echo "I love " . $txt . "!";
?>

</body>
</html>
```

I love rose!

The following example will output the sum of two variables:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
$y = 4;
echo $x + $y;
?>

</body>
</html>
```

```
9
```

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In **PHP 7**, type declarations were added. This gives an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

PHP Variables Scope

In **PHP**, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

PHP PROGRAMMING LANGUAGE

A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

Variable with global scope:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5; // global scope

function myTest() {
  // using x inside this function will generate an error
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>

</body>
</html>
```

Variable x inside function is:

Variable x outside function is: 5

A variable declared within a function has a **LOCAL SCOPE** and can only be accessed within that function:

Example

Variable with local scope:

```
<!DOCTYPE html>
<html>
<body>

<?php
function myTest() {
  $x = 5; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>

</body>
</html>
```

Variable x inside function is: 5

Variable x outside function is:

PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
$y = 10;

function myTest() {
  global $x, $y;
  $y = $x + $y;
}

myTest();  // run function
echo $y; // output the new value for variable $y
?>

</body>
</html>
```

15

PHP also stores all global variables in an array called **$GLOBALS[index]**. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
$y = 10;

function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y;
?>

</body>
</html>
```

15

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}

myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
?>

</body>
</html>
```

```
0
1
2
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

➢ PHP echo and print Statements

With **PHP**, there are two basic ways to get output: **echo** and **print**.

PHP echo and print Statements

echo and **print** are more or less the same. They are both used to output data to the screen. The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

The PHP echo Statement

The **echo** statement can be used with or without parentheses: **echo** or **echo()**.

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>

</body>
</html>
```

PHP is Fun!

Hello world!
I'm about to learn PHP!
This string was made with multiple parameters.

Display Variables

The following example shows how to output text and variables with the echo statement:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>

</body>
</html>
```

Learn PHP

Study PHP at W3Schools.com
9

The PHP print Statement

The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

PHP PROGRAMMING LANGUAGE

```
<!DOCTYPE html>
<html>
<body>

<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>

</body>
</html>
```

PHP is Fun!

Hello world!

I'm about to learn PHP!

Display Variables

The following example shows how to output text and variables with the **print** statement:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>";
print $x + $y;
?>

</body>
</html>
```

Learn PHP

Study PHP at W3Schools.com

9

PHP PROGRAMMING LANGUAGE

➢ PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String

A **string** is a sequence of characters, like "Hello world!".

A **string** can be any text inside quotes. You can use single or double quotes:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "<br>";
echo $y;
?>

</body>
</html>
```

Hello world!
Hello world!

PHP PROGRAMMING LANGUAGE

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An **integer** must have at least one digit
- An **integer** must not have a decimal point
- An **integer** can be either positive or negative
- **Integers** can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP **var_dump()** function returns the data type and value:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5985;
var_dump($x);
?>

</body>
</html>
```

```
int(5985)
```

PHP Float

A **float** (floating point number) is a number with a decimal point or a number in exponential form.

In the following example **$x** is a float. The PHP **var_dump()** function returns the data type and value:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 10.365;
var_dump($x);
?>

</body>
</html>
```

```
float(10.365)
```

PHP Boolean

A **Boolean** represents two possible states: **TRUE** or **FALSE**.

```
$x = true;
$y = false;
```

PHP Array

An **array** stores multiple values in one single variable.

In the following example $cars is an array. The PHP **var_dump()** function returns the data type and value:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>

</body>
</html>
```

```
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
```

PHP Object

An **object** is a data type which stores data and information on how to process that data.

PHP PROGRAMMING LANGUAGE

In **PHP**, an object must be explicitly declared.

First, we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
class Car {
  function Car() {
    $this->model = "VW";
  }
}
// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>

</body>
</html>
```

VW

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

Tip: If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>

</body>
</html>
```

```
NULL
```

PHP Resource

The **special resource** type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.

A common example of using the resource data type is a database call.

➢ PHP Strings

A **string** is a sequence of characters, like "Hello world!".

strlen() - Return the Length of a String

The PHP **strlen()** function returns the length of a string.

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo strlen("Hello world!");
?>

</body>
</html>
```

```
12
```

str_word_count() - Count Words in a String

The PHP **str_word_count()** function counts the number of words in a string.

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo str_word_count("Hello world!");
?>

</body>
</html>
```

2

strrev() - Reverse a String

The PHP **strrev()** function reverses a string.

Example

Reverse the string "Hello world!":

```
<!DOCTYPE html>
<html>
<body>

<?php
echo strrev("Hello world!");
?>

</body>
</html>
```

!dlrow olleH

strpos() - Search For a Text Within a String

The PHP **strpos()** function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

Example

Search for the text "world" in the string "Hello world!":

```
<!DOCTYPE html>
<html>
<body>

<?php
echo strpos("Hello world!", "world");
?>

</body>
</html>
```

```
6
```

Tip: The first character position in a string is 0 (not 1).

str_replace() - Replace Text Within a String

The PHP **str_replace()** function replaces some characters with some other characters in a string.

Example

Replace the text "world" with "Dolly":

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

```
Hello Dolly!
```

➢ PHP Numbers

One thing to notice about **PHP** is that it provides automatic data type conversion.

So, if you assign an integer value to a **variable**, the type of that **variable** will automatically be an integer. Then, if you assign a string to the same variable, the type will change to a string.

This automatic conversion can sometimes break your code.

PHP PROGRAMMING LANGUAGE

PHP Integers

An integer is a number without any decimal part.

2, 256, -256, 10358, -179567 are all integers. While 7.56, 10.0, 150.67 are floats.

So, an integer data type is a non-decimal number between -2147483648 and 2147483647.

A value greater (or lower) than this, will be stored as float, because it exceeds the limit of an integer.

Another important thing to know is that even if 4 * 2.5 is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

PHP has the following functions to check if the type of a variable is integer:

- is_int()
- is_integer() - alias of is_int()
- is_long() - alias of is_int()

Example

Check if the type of a variable is integer:

```
<!DOCTYPE html>
<html>
<body>

<?php
// Check if the type of a variable is integer
$x = 5985;
var_dump(is_int($x));

echo "<br>";

// Check again...
$x = 59.85;
var_dump(is_int($x));
?>

</body>
</html>
```

```
bool(true)
bool(false)
```

PHP Floats

A **float** is a number with a decimal point or a number in exponential form.

2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.

The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits.

PHP has the following functions to check if the type of a variable is float:

- is_float()
- is_double() - alias of is_float()

Example

Check if the type of a variable is float:

```
<!DOCTYPE html>
<html>
<body>

<?php
// Check if the type of a variable is float
$x = 10.365;
var_dump(is_float($x));
?>

</body>
</html>
```

```
bool(true)
```

PHP Infinity

A **numeric** value that is larger than **PHP_FLOAT_MAX** is considered infinite.

PHP has the following functions to check if a numeric value is finite or infinite:

- is_finite()
- is_infinite()

However, the PHP **var_dump()** function returns the data type and value:

Example

Check if a numeric value is finite or infinite:

```
<!DOCTYPE html>
<html>
<body>

<?php
// Check if a numeric value is finite or infinite
$x = 1.9e411;
var_dump($x);
?>

</body>
</html>
```

```
float(INF)
```

PHP NaN

NaN stands for Not a Number.

NaN is used for impossible mathematical operations.

PHP has the following functions to check if a value is not a number:

- **is_nan()**

However, the PHP **var_dump()** function returns the data type and value:

Example

Invalid calculation will return a **NaN** value:

```
<!DOCTYPE html>
<html>
<body>

<?php
// Invalid calculation will return a NaN value
$x = acos(8);
var_dump($x);
?>

</body>
</html>
```

```
float(NAN)
```

PHP Numerical Strings

The PHP **is_numeric()** function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

Example

Check if the variable is numeric:

```
<!DOCTYPE html>
<html>
<body>

<?php
// Check if the variable is numeric
$x = 5985;
var_dump(is_numeric($x));

echo "<br>";

$x = "5985";
var_dump(is_numeric($x));

echo "<br>";

$x = "59.85" + 100;
var_dump(is_numeric($x));

echo "<br>";

$x = "Hello";
var_dump(is_numeric($x));
?>

</body>
</html>
```

```
bool(true)
bool(true)
bool(true)
bool(false)
```

PHP Casting Strings and Floats to Integers

Sometimes you need to cast a numerical value into another data type.

The (int), (integer), or **intval()** function are often used to convert a value to an integer.

Example

Cast float and string to integer:

```
<!DOCTYPE html>
<html>
<body>

<?php
// Cast float to int
$x = 23465.768;
$int_cast = (int)$x;
echo $int_cast;

echo "<br>";

// Cast string to int
$x = "23465.768";
$int_cast = (int)$x;
echo $int_cast;
?>

</body>
</html>
```

```
23465
23465
```

➢ PHP Math

PHP has a set of math functions that allows you to perform mathematical tasks on numbers.

PHP pi() Function

The **pi()** function returns the value of PI:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo(pi());
?>

</body>
</html>
```

```
3.1415926535898
```

PHP min() and max() Functions

The **min()** and **max()** functions can be used to find the lowest or highest value in a list of arguments:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo(min(0, 150, 30, 20, -8, -200) . "<br>");
echo(max(0, 150, 30, 20, -8, -200));
?>

</body>
</html>
```

```
-200
150
```

PHP abs() Function

The **abs()** function returns the absolute (positive) value of a number:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo(abs(-6.7));
?>

</body>
</html>
```

```
6.7
```

PHP sqrt() Function

The **sqrt()** function returns the square root of a number:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo(sqrt(64) . "<br>");
echo(sqrt(0) . "<br>");
echo(sqrt(1) . "<br>");
echo(sqrt(9));
?>

</body>
</html>
```

8

0

1

3

PHP round() Function

The **round()** function rounds a floating-point number to its nearest integer:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo(round(0.60) . "<br>");
echo(round(0.50) . "<br>");
echo(round(0.49) . "<br>");
echo(round(-4.40) . "<br>");
echo(round(-4.60));
?>

</body>
</html>
```

1

1

0

-4

-5

Random Numbers

The **rand()** function generates a random number:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo(rand());
?>

</body>
</html>
```

```
1381167960
```

To get more control over the random number, you can add the optional **min** and **max** parameters to specify the lowest integer and the highest integer to be returned.

For example, if you want a random integer between 10 and 100 (inclusive), use **rand(10, 100)**:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo(rand(10, 100));
?>

</body>
</html>
```

```
97
```

➢ PHP Constants

Constants are like variables except that once they are defined, they cannot be changed or undefined.

A **constant** is an identifier (name) for a simple value. The value cannot be changed during the script.

A **valid constant** name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

PHP PROGRAMMING LANGUAGE

Create a PHP Constant

To create a constant, use the **define()** function.

Syntax

```
define(name, value, case-insensitive)
```

Parameters:

- **name:** Specifies the name of the constant
- **value:** Specifies the value of the constant
- **case-insensitive:** Specifies whether the constant name should be case-insensitive. Default is false

Example

Create a constant with a case-sensitive name:

```
<!DOCTYPE html>
<html>
<body>

<?php
// case-sensitive constant name
define("GREETING", "Welcome to iraq!");
echo GREETING;
?>

</body>
</html>
```

Welcome to iraq!

Example

Create a constant with a case-insensitive name:

```
<!DOCTYPE html>
<html>
<body>

<?php
// case-insensitive constant name
define("GREETING", "Welcome to Iraq!", true);
echo greeting;
?>

</body>
</html>
```

Welcome to Iraq!

PHP Constant Arrays

In PHP7, you can create an Array constant using the **define()** function.

Example

Create an Array constant:

```
<!DOCTYPE html>
<html>
<body>

<?php
define("cars", [
  "Alfa Romeo",
  "BMW",
  "Toyota"
]);
echo cars[0];
?>

</body>
</html>
```

Alfa Romeo

Constants are Global

Constants are automatically global and can be used across the entire script.

Example

This example uses a constant inside a function, even if it is defined outside the function:

```
<!DOCTYPE html>
<html>
<body>

<?php
define("GREETING", "Welcome to Iraq");

function myTest() {
  echo GREETING;
}

myTest();
?>

</body>
</html>
```

Welcome to Iraq

➢ PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

PHP Arithmetic Operators

The **PHP arithmetic** operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

The **PHP assignment** operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

PHP Comparison Operators

The **PHP comparison** operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.

PHP Increment / Decrement Operators

The **PHP increment** operators are used to increment a variable's value.

The **PHP decrement** operators are used to decrement a variable's value.

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one

--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

The **PHP logical operators** are used to combine conditional statements.

Operator	Name	Example	Result
And	And	$x and $y	True if both $x and $y are true
Or	Or	$x or $y	True if either $x or $y is true
Xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Array Operators

The **PHP array operators** are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

PHP Conditional Assignment Operators

The **PHP conditional assignment operators** are used to set a value depending on conditions:

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

➢ PHP if...else...elseif Statements

Conditional statements are used to perform different actions based on different conditions.

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif...else** statement - executes different codes for more than two conditions
- **switch statement** - selects one of many blocks of code to be executed

PHP - The if Statement

The **if statement** executes some code if one condition is true.

Syntax

```
if (condition) {
  code to be executed if condition is true;
}
```

Example

Output "Have a good day!" if the current time (HOUR) is less than 20:

```
<!DOCTYPE html>
<html>
<body>

<?php
$t = date("H");

if ($t < "20") {
  echo "Have a good day!";
}
?>

</body>
</html>
```

Have a good day!

PHP - The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {
  code to be executed if condition is true;
} else {
  code to be executed if condition is false;
}
```

Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
<!DOCTYPE html>
<html>
<body>

<?php
$t = date("H");

if ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>

</body>
</html>
```

Have a good day!

PHP - The if...elseif...else Statement

The **if...elseif...else** statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
  code to be executed if this condition is true;
} elseif (condition) {
  code to be executed if first condition is false and this
condition is true;
} else {
  code to be executed if all conditions are false;
}
```

Example

Output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

```
<!DOCTYPE html>
<html>
<body>

<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following message:</p>";

if ($t < "10") {
  echo "Have a good morning!";
} elseif ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>

</body>
</html>
```

The hour (of the server) is 17, and will give the following message:

Have a good day!

➢ PHP switch Statement

The **switch statement** is used to perform different actions based on different conditions.

The PHP switch Statement

Use the **switch statement** to select one of many blocks of code to be executed.

Syntax

```
switch (n) {
  case label1:
    code to be executed if n=label1;
    break;
  case label2:
    code to be executed if n=label2;
    break;
  case label3:
    code to be executed if n=label3;
```

```
    break;
    ...
  default:
    code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The default statement is used if no match is found.

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$favcolor = "red";

switch ($favcolor) {
  case "red":
    echo "Your favorite color is red!";
    break;
  case "blue":
    echo "Your favorite color is blue!";
    break;
  case "green":
    echo "Your favorite color is green!";
    break;
  default:
    echo "Your favorite color is neither red, blue, nor green!";
}
?>

</body>
</html>
```

Your favorite color is red!

➢ PHP Loops

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

PHP while Loop

The **while loop** - Loops through a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
  code to be executed;
}
```

Example

The example below displays the numbers from 1 to 5:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 1;

while($x <= 5) {
  echo "The number is: $x <br>";
  $x++;
}
?>

</body>
</html>
```

The number is: 1

The number is: 2

The number is: 3

The number is: 4

The number is: 5

Example Explained

- $x = 1; - Initialize the loop counter ($x), and set the start value to 1
- $x <= 5 - Continue the loop as long as $x is less than or equal to 5
- $x++; - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 0;

while($x <= 100) {
  echo "The number is: $x <br>";
  $x+=10;
}
?>

</body>
</html>
```

```
The number is: 0
The number is: 10
The number is: 20
The number is: 30
The number is: 40
The number is: 50
The number is: 60
The number is: 70
The number is: 80
The number is: 90
The number is: 100
```

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 100 - Continue the loop as long as $x is less than or equal to 100
- $x+=10; - Increase the loop counter value by 10 for each iteration

PHP do while Loop

The **do...while loop** will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
  code to be executed;
} while (condition is true);
```

Example

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 1;

do {
  echo "The number is: $x <br>";
  $x++;
} while ($x <= 5);
?>

</body>
</html>
```

The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5

This example sets the $x variable to 6, then it runs the loop, and then the condition is checked:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 6;

do {
  echo "The number is: $x <br>";
  $x++;
} while ($x <= 5);
?>

</body>
</html>
```

The number is: 6

PHP for Loop

The **for loop** is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
  code to be executed for each iteration;
}
```

Parameters:

- init counter: Initialize the loop counter value
- test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- increment counter: Increases the loop counter value

Example

The example below displays the numbers from 0 to 10:

```
<!DOCTYPE html>
<html>
<body>

<?php
for ($x = 0; $x <= 10; $x++) {
  echo "The number is: $x <br>";
}
?>

</body>
</html>
```

The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
The number is: 10

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 10; - Continue the loop as long as $x is less than or equal to 10

- $x++ - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
for ($x = 0; $x <= 100; $x+=10) {
  echo "The number is: $x <br>";
}
?>

</body>
</html>
```

The number is: 0
The number is: 10
The number is: 20
The number is: 30
The number is: 40
The number is: 50
The number is: 60
The number is: 70
The number is: 80
The number is: 90
The number is: 100

Example Explained

- $x = 0; - Initialize the loop counter ($x), and set the start value to 0
- $x <= 100; - Continue the loop as long as $x is less than or equal to 100
- $x+=10 - Increase the loop counter value by 10 for each iteration

PHP foreach Loop

The **foreach loop** works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
  code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

Example

The following example will output the values of the given array ($colors):

```
<!DOCTYPE html>
<html>
<body>

<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
  echo "$value <br>";
}
?>

</body>
</html>
```

red
green
blue
yellow

The following example will output both the keys and the values of the given array ($age):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $val) {
  echo "$x = $val<br>";
}
?>

</body>
</html>
```

Peter = 35

Ben = 37

Joe = 43

➢ PHP Functions

PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

PHP User Defined Functions

Besides the **built-in PHP functions**, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

Create a User Defined Function in PHP

A **user-defined function** declaration starts with the word function:

Syntax

```
function functionName() {
  code to be executed;
}
```

Note: A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

Tip: Give the function a name that reflects what the function does!

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code, and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name followed by brackets ():

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
function writeMsg() {
  echo "Hello world!";
}

writeMsg();
?>

</body>
</html>
```

Hello world!

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
function familyName($fname) {
  echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>

</body>
</html>
```

Jani Refsnes.

Hege Refsnes.

Stale Refsnes.

Kai Jim Refsnes.

Borge Refsnes.

The following example has a function with two arguments ($fname and $year):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
function familyName($fname, $year) {
  echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege","1975");
familyName("Stale","1978");
familyName("Kai Jim","1983");
?>

</body>
</html>
```

Hege Refsnes. Born in 1975

Stale Refsnes. Born in 1978

Kai Jim Refsnes. Born in 1983

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatches.

In the following example we try to send both a number and a string to the function without using strict:

Example

```php
<?php
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is changed to int(5), and it will return 10
?>
```

```
10
```

To specify strict we need to set **declare (strict_types=1);**. This must be on the very first line of the PHP file.

In the following example we try to send both a number and a string to the function, but here we have added the strict declaration:

Example

```php
<?php declare(strict_types=1); // strict requirement

function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer, an error will be thrown
?>
```

```
PHP Fatal error: Uncaught TypeError: Argument 2 passed to addNumbers() must be of the type integer, string
given, called in /home/gn4Q8y/prog.php on line 6 and defined in /home/gn4Q8y/prog.php:3 Stack trace: #0
/home/gn4Q8y/prog.php(6): addNumbers(5, '5 days') #1 {main} thrown in /home/gn4Q8y/prog.php on line 3
```

The strict declaration forces things to be used in the intended way.

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function **setHeight()** without arguments it takes the default value as argument:

Example

```
<?php declare(strict_types=1); // strict requirement ?>
<!DOCTYPE html>
<html>
<body>

<?php
function setHeight(int $minheight = 50) {
  echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>

</body>
</html>
```

```
The height is : 350
The height is : 50
The height is : 135
The height is : 80
```

PHP Functions - Returning values

To let a function, return a value, use the return statement:

Example

```php
<?php declare(strict_types=1); // strict requirement ?>
<!DOCTYPE html>
<html>
<body>

<?php
function sum(int $x, int $y) {
  $z = $x + $y;
  return $z;
}

echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>";
echo "2 + 4 = " . sum(2,4);
?>

</body>
</html>
```

5 + 10 = 15

7 + 13 = 20

2 + 4 = 6

PHP Return Type Declarations

PHP 7 also supports Type Declarations for the return statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

To declare a type for the function return, add a colon (:) and the type right before the opening curly ({)bracket when declaring the function.

In the following example we specify the return type for the function:

Example

```php
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
  return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

6.4

You can specify a different return type, than the argument types, but make sure the return is the correct type:

Example

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : int {
  return (int)($a + $b);
}
echo addNumbers(1.2, 5.2);
?>
```

```
6
```

➢ PHP Arrays

An array stores multiple values in one single variable:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>

</body>
</html>
```

```
I like Volvo, BMW and Toyota.
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the **array()** function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- Indexed arrays - Arrays with a numeric index
- Associative arrays - Arrays with named keys
- Multidimensional arrays - Arrays containing one or more arrays

Get The Length of an Array - The count() Function

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

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>

</body>
</html>
```

```
3
```

➢ PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>

</body>
</html>
```

I like Volvo, BMW and Toyota.

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:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
  echo $cars[$x];
  echo "<br>";
}
?>

</body>
</html>
```

Volvo
BMW
Toyota

➢ PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

Peter is 35 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:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>

</body>
</html>
```

Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43

PHP Multidimensional Arrays

A **multidimensional** array is an array containing one or more arrays.

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.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13

Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

Now the **two-dimensional $cars** array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);

echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>

</body>
</html>
```

Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.

We can also put a for loop inside another for loop to get the elements of the $cars array (we still have to point to the two indices):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);

for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>

</body>
</html>
```

Row number 0

- Volvo
- 22
- 18

Row number 1

- BMW
- 15
- 13

Row number 2

- Saab
- 5
- 2

Row number 3

- Land Rover
- 17
- 15

➢ PHP Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- **sort()** - sort arrays in ascending order
- **rsort()** - sort arrays in descending order
- **asort()** - sort associative arrays in ascending order, according to the value
- **ksort()** - sort associative arrays in ascending order, according to the key
- **arsort()** - sort associative arrays in descending order, according to the value
- **krsort()** - sort associative arrays in descending order, according to the key

Sort Array in Ascending Order - sort()

The following example sorts the elements of the **$cars** array in ascending alphabetical order:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
  echo $cars[$x];
  echo "<br>";
}
?>

</body>
</html>
```

BMW
Toyota
Volvo

The following example sorts the elements of the $numbers array in ascending numerical order:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);

$arrlength = count($numbers);
for($x = 0; $x < $arrlength; $x++) {
  echo $numbers[$x];
  echo "<br>";
}
?>

</body>
</html>
```

```
2
4
6
11
22
```

Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
  echo $cars[$x];
  echo "<br>";
}
?>

</body>
</html>
```

Volvo
Toyota
BMW

The following example sorts the elements of the $numbers array in descending numerical order:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);

$arrlength = count($numbers);
for($x = 0; $x < $arrlength; $x++) {
  echo $numbers[$x];
  echo "<br>";
}
?>

</body>
</html>
```

22
11
6
4
2

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>

</body>
</html>
```

Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>

</body>
</html>
```

Key=Ben, Value=37
Key=Joe, Value=43
Key=Peter, Value=35

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>

</body>
</html>
```

Key=Joe, Value=43
Key=Ben, Value=37
Key=Peter, Value=35

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>

</body>
</html>
```

Key=Peter, Value=35
Key=Joe, Value=43
Key=Ben, Value=37

➢ PHP Global Variables – Superglobals

Some predefined variables in PHP are "**superglobals**", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP **superglobal** variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

PHP Superglobal - $GLOBALS

Super global variables are built-in variables that are always available in all scopes.

PHP $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called **$GLOBALS[index]**. The index holds the name of the variable.

The example below shows how to use the super global variable **$GLOBALS**:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 75;
$y = 25;

function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>

</body>
</html>
```

```
100
```

PHP Superglobal - $_SERVER

Super global variables are built-in variables that are always available in all scopes.

PHP $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in **$_SERVER**:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>

</body>
</html>
```

```
/demo/demo_global_server.php
35.194.26.41
35.194.26.41
https://tryphp.w3schools.com/showphp.php?filename=demo_global_server
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/81.0.4044.129 Safari/537.36
/demo/demo_global_server.php
```

The following table lists the most important elements that can go inside **$_SERVER**:

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request

$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)

$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP Superglobal - $_REQUEST

PHP **$_REQUEST** is a PHP super global variable which is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<!DOCTYPE html>
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = htmlspecialchars($_REQUEST['fname']);
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

Name: Submit

PHP Superglobal - $_POST

PHP **$_POST** is a PHP super global variable which is used to collect form data after submitting an HTML form with **method="post"**. **$_POST** is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable **$_POST** to collect the value of the input field:

Example

```
<!DOCTYPE html>
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

Name: Submit

➢ PHP Superglobal - $_GET

PHP $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".

$_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>
<a href="test_get.php?subject=PHP&web=W3schools.com">Test
$GET</a>
```

```
</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "**test_get.php**", and you can then access their values in "**test_get.php**" with **$_GET**. The example below shows the code in "**test_get.php**":

Example

```
<!DOCTYPE html>
<html>
<body>

<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>
</html>
```

Test $GET

PHP PROGRAMMING LANGUAGE

PHP Form Handling

The PHP superglobals **$_GET** and **$_POST** are used to collect form-data.

PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example

```
<!DOCTYPE HTML>
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

Name:
E-mail:
Submit

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "**welcome.php**". The form data is sent with the HTTP POST method.

To display the submitted data, you could simply echo all the variables. The "**welcome.php**" looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
```

```
</body>
</html>
```

The output could be something like this:

```
Welcome John
Your email address is john.doe@example.com
```

The same result could also be achieved using the **HTTP GET** method:

Example

```
<!DOCTYPE HTML>
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

Name:
E-mail:
Submit

and "**welcome_get.php**" looks like this:

```
<html>
<body>
Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
</body>
</html>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

GET vs. POST

Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both **GET** and **POST** are treated as **$_GET** and **$_POST**. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the **URL** parameters.

$_POST is an array of variables passed to the current script via the **HTTP POST** method.

When to use GET?

Information sent from a form with the **GET** method is visible to everyone (all variable names and values are displayed in the **URL**). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the **URL**, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the **POST** method is invisible to others (all names/values are embedded within the body of the **HTTP** request) and has no limits on the amount of information to send.

Moreover, **POST** supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

PHP PROGRAMMING LANGUAGE

➢ PHP Form Validation

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Submit

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

Text Fields

The **name**, **email**, and **website** fields are text input elements, and the comment field is a **textarea**. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
```

```
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
```

The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER
["PHP_SELF"]);?>">
```

When the form is submitted, the form data is sent with method="post".

What is the $_SERVER["PHP_SELF"] variable?

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars() function?

The **htmlspecialchars()** function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with **&lt**; and **>**. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!

If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (**XSS**) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "**test_form.php**":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following **URL** in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that any JavaScript code can be added inside the <script> tag! A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the **htmlspecialchars()** function.

The form code should look like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

The **htmlspecialchars()** function converts special characters to HTML entities. Now if the user tries to exploit the **PHP_SELF** variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's **htmlspecialchars()** function.

When we use the **htmlspecialchars()** function; then if a user tries to submit the following in a text field:

<script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this:

<script>location.href('http://www.hacked.com')</script>

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP **trim()** function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function **test_input()**.

Now, we can check each **$_POST** variable with the **test_input()** function, and the script looks like this:

PHP PROGRAMMING LANGUAGE

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>

<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

PHP Form Validation Example

Name:

E-mail:

Website:

Comment:

Gender: Female Male Other

Submit

Your Input:

Notice that at the start of the script, we check whether the form has been submitted using **$_SERVER["REQUEST_METHOD**"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed.

➢ PHP Forms - Required Fields

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

```
<?php
// define variables and set to empty values
```

PHP PROGRAMMING LANGUAGE

```
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }
  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }
  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
```

```
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

Example

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
```

```
    $name = test_input($_POST["name"]);
  }
  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }
  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
```

```
  $data = htmlspecialchars($data);
  return $data;
}
?>
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER
["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textare
a>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <input type="radio" name="gender" value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
```

PHP PROGRAMMING LANGUAGE

```
  <input type="submit" name="submit" value="Submit">
</form>
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
</body>
</html>
```

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Submit

Your Input:

➢ PHP Forms - Validate E-mail and URL

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's **filter_var()** function.

In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

PHP - Validate Name, E-mail, and URL

Now, the script looks like this:

Example

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }
  if (empty($_POST["email"])) {
```

```
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }
  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
```

```
    $gender = test_input($_POST["gender"]);
  }
}
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER
["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textare
a>
  <br><br>
```

PHP PROGRAMMING LANGUAGE

```
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <input type="radio" name="gender" value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
</body>
</html>
```

PHP PROGRAMMING LANGUAGE

PHP Form Validation Example

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Submit

Your Input:


Python Programming
Language

PYTHON PROGRAMMING LANGUAGE

Python Introduction

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

PYTHON PROGRAMMING LANGUAGE

- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

PYTHON PROGRAMMING LANGUAGE

Python Getting Started

Python Install

Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

```
C:\Users\Your Name>python --version
```

To check if you have python installed on a Linux or Mac, then on **linux** open the command line or on Mac open the Terminal and type:

```
python --version
```

If you find that you do not have python installed on your computer, then you can download it for free from the following website: **https://www.python.org/**

Python Quickstart

Python is an interpreted programming language; this means that as a developer you write Python (**.py**) files in a text editor and then put those files into the python interpreter to be executed.

The way to run a python file is like this on the command line:

```
C:\Users\Your Name>python helloworld.py
```

Where "**helloworld.py**" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

```
#!/bin/python3
print("Hello, World!")
```

```
Hello, World!
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

```
C:\Users\Your Name>python helloworld.py
```

The output should read:

```
Hello, World!
```

The Python Command Line

To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following on the **Windows**, **Mac** or **Linux** command line:

```
C:\Users\Your Name>python
```

Or, if the "**python**" command did not work, you can try "**py**":

```
C:\Users\Your Name>py
```

From there you can write any python, including our hello world example from earlier in the tutorial:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
```

Which will write "Hello, World!" in the command line:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
```

Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:

```
exit()
```

PYTHON PROGRAMMING LANGUAGE

Python Tutorial

➢ PHP Syntax

Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

```
Five is greater than two!
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

```
  File "demo_indentation_test.py", line 2
    print("Five is greater than two!")
        ^
IndentationError: expected an indented block
```

The number of spaces is up to you as a programmer, but it has to be at least one.

Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

```
Five is greater than two!
Five is greater than two!
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

```
  File "demo_indentation2_error.py", line 3
    print("Five is greater than two!")
    ^
IndentationError: unexpected indent
```

Python Variables

In Python, variables are created when you assign a value to it:

Example

Variables in Python:

```
x = 5
y = "Hello, World!"

print(x)
print(y)
```

```
5
Hello, World!
```

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

Example

```
#This is a comment.
print("Hello, World!")
```

```
Hello, World!
```

➢ Python Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

Creating a Comment

Comments starts with a #, and Python will ignore them:

Example

```
#This is a comment.
print("Hello, World!")
```

```
Hello, World!
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

```
print("Hello, World!") #This is a comment.
```

```
Hello, World!
```

Comments does not have to be text to explain the code, it can also be used to prevent Python from executing code:

Example

```
#print("Hello, World!")
print("Cheers, Mate!")
```

```
Cheers, Mate!
```

Multi Line Comments

Python does not really have a syntax for multi-line comments.

To add a multiline comment, you could insert a # for each line:

Example

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
Hello, World!
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
Hello, World!
```

➢ Python Variables

Creating Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example

```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

Variables do not need to be declared with any particular type and can even change type after they have been set.

Example

```
x = 4
x = "Sally"
print(x)
```

```
Sally
```

String variables can be declared either by using single or double quotes:

Example

```
x = "John"
print(x)
#double quotes are the same as single quotes:
x = 'John'
print(x)
```

```
John
John
```

Variable Names

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:
- A variable name must start with a letter or the underscore character

- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

```
  File "demo_variable_names.py", line 10
    2myvar = "John"
         ^
SyntaxError: invalid syntax
```

Assign Value to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Banana", "Cherry"

print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

And you can assign the same value to multiple variables in one line:

Example

```
x = y = z = "Orange"

print(x)
print(y)
print(z)
```

```
Orange
Orange
Orange
```

Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

Example

```
x = "awesome"
print("Python is " + x)
```

```
Python is awesome
```

You can also use the + character to add a variable to another variable:

Example

```
x = "Python is "
y = "awesome"
z = x + y
print(z)
```

```
Python is awesome
```

For numbers, the + character works as a mathematical operator:

Example

```
x = 5
y = 10
print(x + y)
```

```
15
```

If you try to combine a string and a number, Python will give you an error:

Example

```
x = 5
y = "John"
print(x + y)
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

```
Python is awesome
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

```
Python is fantastic
Python is awesome
```

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

```
Python is fantastic
```

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

```
Python is fantastic
```

➢ Python Data Types

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	Str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	Dict
Set Types:	set, frozenset
Boolean Type:	Bool
Binary Types:	bytes, bytearray, memoryview

Getting the Data Type

You can get the data type of any object by using the **type()** function:

Example

```
x = 5
print(type(x))
```

```
<class 'int'>
```

Setting the Data Type

In **Python**, the data type is set when you assign a value to a variable:

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple

x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool

x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

➢ Python Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

Example

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the **type()** function:

Example

```
x = 1
y = 2.8
z = 1j

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

Floats:

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods:

Example

Convert from one type to another:

```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(x)

print(x)
print(y)
print(z)
```

```
1.0
2
(1+0j)
<class 'float'> <class 'int'> <class 'complex'>
```

Note: You cannot convert complex numbers into another number type.

Random Number

Python does not have a **random()** function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Example

Import the random module, and display a random number between 1 and 9:

```
import random

print(random.randrange(1, 10))
```

```
4
```

➢ Python Casting

Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- **int()** - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- **float()** - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- **str()** - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example

Integers:

```
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

```
1
2
3
```

Example

Floats:

```
x = float(1)
y = float(2.8)
z = float("3")
w = float("4.2")
print(x)
print(y)
print(z)
print(w)
```

```
1.0
2.8
3.0
4.2
```

Example

Strings:

```
x = str("s1")
y = str(2)
z = str(3.0)
print(x)
print(y)
print(z)
```

```
s1
2
3.0
```

➢ Python Strings

String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the **print()** function:

Example

```
#You can use double or single quotes:

print("Hello")
print('Hello')
```

```
Hello
Hello
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

```
Hello
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Or three single quotes:

Example

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing **Unicode** characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

```
e
```

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

```
llo
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters from position 5 to position 1, starting the count from the end of the string:

```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

String Length

To get the length of a string, use the **len()** function.

Example

The **len()** function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

```
13
```

String Methods

Python has a set of built-in methods that you can use on strings.

Example

The **strip()** method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip())
```

```
Hello, World!
```

Example

The **lower()** method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())
```

```
hello, world!
```

Example

The **upper()** method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

```
HELLO, WORLD!
```

Example

The **replace()** method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
Jello, World!
```

Example

The **split()** method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
b = a.split(",")
print(b)
```

```
['Hello', ' World!']
```

Check String

To check if a certain phrase or character is present in a string, we can use the keywords in or not in.

Example

Check if the phrase "**ain**" is present in the following text:

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x)
```

```
True
```

Example

Check if the phrase "**ain**" is **NOT** present in the following text:

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" not in txt
print(x)
```

```
False
```

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable a with variable b into variable c:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

```
HelloWorld
```

Example

To add a space between them, add a " ":

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

```
Hello World
```

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

```
Traceback (most recent call last):
  File "demo_string_format_error.py", line 2, in <module>
    txt = "My name is John, I am " + age
TypeError: must be str, not int
```

But we can combine strings and numbers by using the format() method!

The **format()** method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Example

Use the **format()** method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

```
My name is John, and I am 36
```

The **format()** method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567
```

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."


#You will get an error if you use double quotes inside a string that are surrounded by
double quotes:
```

```
  File "demo_string_escape_error.py", line 1
    txt = "We are the so-called "Vikings" from the north."
                                ^
SyntaxError: invalid syntax
```

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```

```
We are the so-called "Vikings" from the north.
```

Other escape characters used in Python:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

String Methods

Python has a set of built-in methods that you can use on strings.

Note: All string methods returns new values. They do not change the original string.

Method	Description
capitalize()	Converts the first character to upper case

casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title

isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case

translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

➢ Python Booleans

Booleans represent one of two values: **True** or **False**.

Boolean Values

In programming you often need to know if an expression is **True** or **False**.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Example

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

When you run a condition in an if statement, Python returns **True** or **False**:

Example

Print a message based on whether the condition is True or False:

```
a = 200
b = 33

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
b is not greater than a
```

Evaluate Values and Variables

The **bool()** function allows you to evaluate any value, and give you True or False in return,

Example

Evaluate a string and a number:

```
print(bool("Hello"))
print(bool(15))
```

```
True
True
```

Example

Evaluate two variables:

```
x = "Hello"
y = 15

print(bool(x))
print(bool(y))
```

```
True
True
```

Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is **True**, except empty strings.

Any number is **True**, except **0**.

Any list, tuple, set, and dictionary are True, except empty ones.

Example

The following will return True:

```
print(bool("abc"))
print(bool(123))
print(bool(["apple", "cherry", "banana"]))
```

```
True
True
True
```

Some Values are False

In fact, there are not many values that evaluates to False, except empty values, such as **()**, **[]**, **{}**, "", the number 0, and the value None. And of course, the value False evaluates to False.

Example

The following will return False:

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

```
False
False
False
False
False
False
False
```

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a **__len__** function that returns 0 or False:

Example

```
class myclass():
  def __len__(self):
    return 0

myobj = myclass()
print(bool(myobj))
```

```
False
```

Functions can Return a Boolean

You can create functions that returns a Boolean Value:

Example

Print the answer of a function:

```
def myFunction() :
  return True

print(myFunction())
```

```
True
```

You can execute code based on the Boolean answer of a function:

Example

Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :
  return True

if myFunction():
  print("YES!")
else:
  print("NO!")
```

```
YES!
```

Python also has many built-in functions that returns a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:

Example

Check if an object is an integer or not:

```
x = 200
print(isinstance(x, int))
```

```
True
```

➢ Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5

+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
Or	Returns True if one of the statements is true	x < 5 or x < 4
Not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

➢ Python Lists

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- List is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- Set is a collection which is unordered and unindexed. No duplicate members.
- Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

List

A **list** is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

Access Items

You access the list items by referring to the index number:

Example

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
banana
```

Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

```
cherry
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])

#This will return the items from position 2 to 5.

#Remember that the first item is position 0,
#and note that the item in position 5 is NOT included
```

```
['cherry', 'orange', 'kiwi']
```

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to "orange":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])

#This will return the items from index 0 to index 4.

#Remember that index 0 is the first item, and index 4 is the fifth item
#Remember that the item in index 4 is NOT included
```

```
['apple', 'banana', 'cherry', 'orange']
```

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" and to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])

#This will return the items from index 2 to the end.

#Remember that index 0 is the first item, and index 2 is the third
```

```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])

#Negative indexing means starting from the end of the list.

#This example returns the items from index -4 (included) to index -1 (excluded)

#Remember that the last item has the index -1,
```

```
['orange', 'kiwi', 'melon']
```

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"

print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

Loop Through a List

You can loop through the list items by using a for loop:

Example

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

```
apple
banana
cherry
```

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

```
Yes, 'apple' is in the fruits list
```

List Length

To determine how many items a list has, use the **len()** function:

Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
3
```

Add Items

To add an item to the end of the list, use the **append()** method:

Example

Using the **append()** method to append an item:

```
thislist = ["apple", "banana", "cherry"]

thislist.append("orange")

print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

To add an item at the specified index, use the **insert()** method:

Example

Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

Remove Item

There are several methods to remove items from a list:

Example

The **remove()** method removes the specified item:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry']
```

Example

The **pop()** method removes the specified index, (or the last item if index is not specified):

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

```
['apple', 'banana']
```

Example

The **del** keyword removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

Example

The **del** keyword can also delete the list completely:

```
thislist = ["apple", "banana", "cherry"]
del thislist
print(thislist) #this will cause an error because you have succsesfully deleted
"thislist".
```

```
Traceback (most recent call last):
  File "demo_list_del2.py", line 3, in <module>
    print(thislist) #this will cause an error because you have succsesfully deleted "this
NameError: name 'thislist' is not defined
```

Example

The **clear()** method empties the list:

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

Example

Make a copy of a list with the **copy()** method:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

Another way to make a copy is to use the built-in method list().

Example

Make a copy of a list with the **list()** method:

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

```
['apple', 'banana', 'cherry']
```

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

Join two list:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Another way to join two lists are by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Or you can use the **extend()** method, which purpose is to add elements from one list to another list:

Example

Use the **extend()** method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

The list() Constructor

It is also possible to use the **list()** constructor to make a new list.

Example

Using the **list()** constructor to make a List:

```
thislist = list(("apple", "banana", "cherry"))
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

➢ Python Tuples

Tuple

A **tuple** is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

Example

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])

#This will return the items from position 2 to 5.

#Remember that the first item is position 0,
#and note that the item in position 5 is NOT included
```

```
('cherry', 'orange', 'kiwi')
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])

#Negative indexing means starting from the end of the tuple.

#This example returns the items from index -4 (included) to index -1 (excluded)

#Remember that the last item has the index -1,
```

```
('orange', 'kiwi', 'melon')
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
("apple", "kiwi", "cherry")
```

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
apple
banana
cherry
```

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

Example

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

Tuple Length

To determine how many items a tuple has, use the len() method:

Example

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

Add Items

Once a tuple is created, you cannot add items to it. Tuples are unchangeable.

Example

You cannot add items to a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

```
Traceback (most recent call last):
  File "demo_tuple_add.py", line 2, in <module>
    thistuple[3] = "orange" # This will raise an error
TypeError: 'tuple' object does not support item assignment
```

Create Tuple with One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example

One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

Remove Items

Note: You cannot remove items in a tuple.

Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely:

Example

The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

```
Traceback (most recent call last):
  File "demo_tuple_del.py", line 3, in <module>
    print(thistuple) #this will raise an error because the tuple no longer exists
NameError: name 'thistuple' is not defined
```

Join Two Tuples

To join two or more tuples, you can use the + operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

The tuple() Constructor

It is also possible to use the **tuple()** constructor to make a tuple.

Example

Using the **tuple()** method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry"))
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

➢ Python Sets

Set

A **set** is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)

# Note: the set list is unordered, meaning: the items will appear in a random order.

# Refresh this page to see the change in the result.
```

```
{'banana', 'apple', 'cherry'}
```

Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
cherry
apple
banana
```

Example

Check if "banana" is present in the set:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Add Items

To add one item to a set use the **add()** method.

To add more than one item to a set use the **update()** method.

Example

Add an item to a set, using the **add()** method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'banana', 'cherry', 'orange', 'apple'}
```

Example

Add multiple items to a set, using the **update()** method

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

```
{'mango', 'orange', 'cherry', 'grapes', 'apple', 'banana'}
```

Get the Length of a Set

To determine how many items a set has, use the **len()** method.

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

```
3
```

Remove Item

To remove an item in a set, use the **remove()**, or the discard() method.

Example

Remove "banana" by using the **remove()** method:

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

```
{'cherry', 'apple'}
```

Note: If the item to remove does not exist, **remove()** will raise an error.

Example

Remove "banana" by using the **discard()** method:

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

Note: If the item to remove does not exist, discard() will NOT raise an error.

You can also use the **pop()**, method to remove an item, but this method will remove the last item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the pop() method is the removed item.

Example

Remove the last item by using the **pop()** method:

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x) #removed item

print(thisset) #the set after removal
```

```
cherry
{'apple', 'banana'}
```

Note: Sets are unordered, so when using the pop() method, you will not know which item that gets removed.

Example

The **clear()** method empties the set:

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

```
set()
```

Example

The **del** keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset) #this will raise an error because the set no longer exists
```

```
Traceback (most recent call last):
  File "demo_set_del.py", line 5, in <module>
    print(thisset) #this will raise an error because the set no longer exists
NameError: name 'thisset' is not defined
```

Join Two Sets

There are several ways to join two or more sets in Python.

You can use the **union()** method that returns a new set containing all items from both sets, or the **update()** method that inserts all the items from one set into another:

Example

The **union()** method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{'b', 'c', 2, 'a', 3, 1}
```

Example

The **update()** method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{3, 'c', 2, 1, 'a', 'b'}
```

Note: Both union() and update() will exclude any duplicate items.

There are other methods that joins two sets and keeps ONLY the duplicates, or NEVER the duplicates, check the full list of set methods in the bottom of this page.

The set() Constructor

It is also possible to use the **set()** constructor to make a set.

Example

Using the **set()** constructor to make a set:

```
thisset = set(("apple", "banana", "cherry"))
print(thisset)
# Note: the set list is unordered, so the result will display the items in a random
order.
```

```
{'cherry', 'apple', 'banana'}
```

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets

difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

➢ Python Dictionaries

Dictionary

A **dictionary** is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

```
Mustang
```

There is also a method called **get()** that will give you the same result:

Example

Get the value of the "model" key:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

```
Mustang
```

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict["year"] = 2018

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Example

Print all key names in the dictionary, one by one:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

```
brand
model
year
```

Example

Print all values in the dictionary, one by one:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

```
Ford
Mustang
1964
```

Example

You can also use the **values()** method to return values of a dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

```
Ford
Mustang
1964
```

Example

Loop through both keys and values, by using the **items()** method:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

```
brand Ford
model Mustang
year 1964
```

Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword:

Example

Check if "model" is present in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

Dictionary Length

To determine how many items (key-value pairs) a dictionary has, use the **len()** function.

Example

Print the number of items in the dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

print(len(thisdict))
```

```
3
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Removing Items

There are several methods to remove items from a dictionary:

Example

The **pop()** method removes the item with the specified key name:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

Example

The **popitem()** method removes the last inserted item

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

Example

The **del** keyword removes the item with the specified key name:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

Example

The **del** keyword can also delete the dictionary completely:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thislist" no longer exists.
```

```
Traceback (most recent call last):
  File "demo_dictionary_del3.py", line 7, in <module>
    print(thisdict) #this will cause an error because "thisdict" no longer exists.
NameError: name 'thisdict' is not defined
```

Example

The **clear()** method empties the dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
{}
```

Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

Example

Make a copy of a dictionary with the **copy()** method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Another way to make a copy is to use the built-in function **dict()**.

Example

Make a copy of a dictionary with the **dict()** function:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Nested Dictionaries

A **dictionary** can also contain many dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily)
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007},
'child3': {'name': 'Linus', 'year': 2011}}
```

Or, if you want to nest three dictionaries that already exists as dictionaries:

Example

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

print(myfamily)
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007},
'child3': {'name': 'Linus', 'year': 2011}}
```

The dict() Constructor

It is also possible to use the **dict()** constructor to make a new dictionary:

Example

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

➢ Python If ... Else

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b

- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

Example

If statement:

```
a = 33
b = 200

if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200

if b > a:
print("b is greater than a")
```

```
  File "demo_if_error.py", line 4
    print("b is greater than a")
        ^
IndentationError: expected an indented block
```

Elif

The **elif** keyword is python's way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a and b are equal
```

In this example a is equal to b, so the first condition is not true, but the **elif** condition is true, so we print to screen that "a and b are equal".

Else

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

In this example a is greater than b, so the first condition is not true, also the **elif** condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the **elif**:

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
b is not greater than a
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
a = 200
b = 33

if a > b: print("a is greater than b")
```

```
"a is greater than b"
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2
b = 330

print("A") if a > b else print("B")
```

```
B
```

This technique is known as Ternary Operators, or Conditional Expressions.

You can also have multiple else statements on the same line:

Example

One line if else statement, with 3 conditions:

```
a = 330
b = 330

print("A") if a > b else print("=") if a == b else print("B")
```

```
=
```

And

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

Nested If

You can have if statements inside if statements, this is called nested if statements.

Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
Above ten,
and also above 20!
```

The pass Statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

Example

```
a = 33
b = 200

if b > a:
  pass

# having an empty if statement like this, would raise an error without the pass
statement
```

➢ Python While Loops

Python Loops

Python has two primitive loop commands:

- while loops
- for loops

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

Example

Print **i** as long as **i** is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

Note: remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, **i**, which we set to 1.

The break Statement

With the break statement we can stop the loop even if the while condition is true:

Example

Exit the loop when **i** is 3:

```
i = 1
while i < 6:
  print(i)
  if (i == 3):
    break
  i += 1
```

```
1
2
3
```

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)

# Note that number 3 is missing in the result
```

```
1
2
4
5
6
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

➢ Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
apple
banana
cherry
```

The for loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are **iterable** objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

Example

Exit the loop when x is "banana", but this time the break comes before the print:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

```
apple
```

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example

Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

```
apple
cherry
```

The range() Function

To loop through a set of code a specified number of times, we can use the range() function, The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the **range()** function:

```
for x in range(6):
  print(x)
```

```
0
1
2
3
4
5
```

Note that **range(6)** is not the values of 0 to 6, but the values 0 to 5.

The **range()** function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Example

Using the start parameter:

```
for x in range(2, 6):
  print(x)
```

```
2
3
4
5
```

The **range()** function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

Example

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

Example

```
for x in [0, 1, 2]:
  pass

# having an empty for loop like this, would raise an error without the pass statement
```

➢ Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

Example

```
def my_function():
  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Arguments are often shortened to args in Python documentations.

Parameters or Arguments?

The terms **parameter** and **argument** can be used for the same thing: information that are passed into a function.

From a function's perspective:

A **parameter** is the variable listed inside the parentheses in the function definition.

An **argument** is the value that is sent to the function when it is called.

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Example

This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```
Emil Refsnes
```

If you try to call the function with 1 or 3 arguments, you will get an error:

Example

This function expects 2 arguments, but gets only 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

```
Traceback (most recent call last):
  File "demo_function_args_error.py", line 4, in <module>
    my_function("Emil")
TypeError: my_function() missing 1 required positional argument: 'lname'
```

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

Example

If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

Arbitrary Arguments are often shortened to *args in Python documentations.

Keyword Arguments

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

Example

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

```
The youngest child is Linus
```

The phrase Keyword Arguments are often shortened to kwargs in Python documentations.

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

Example

If the number of keyword arguments is unknown, add a double ** before the parameter name:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

```
His last name is Refsnes
```

Arbitrary **Kword** Arguments are often shortened to ****kwargs** in Python documentations.

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

Example

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

Example

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
apple
banana
cherry
```

Return Values

To let a function return a value, use the return statement:

Example

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Example

```
def myfunction():
  pass

# having an empty function definition like this, would raise an error without the pass statement
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result. The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or

processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, **tri_recursion()** is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

Example

Recursion Example

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

```
Recursion Example Results
1
3
6
10
15
21
```

➢ Python Lambda

A **lambda** function is a small anonymous function.

A **lambda** function can take any number of arguments, but can only have one expression.

Syntax

lambda ***arguments*** **:** ***expression***

The expression is executed and the result is returned:

Example

A **lambda** function that adds 10 to the number passed in as an argument, and print the result:

```
x = lambda a: a + 10
print(x(5))
```

```
15
```

Lambda functions can take any number of arguments:

Example

A **lambda** function that multiplies argument a with argument b and print the result:

```
x = lambda a, b: a * b
print(x(5, 6))
```

```
30
```

Example

A **lambda** function that sums argument a, b, and c and print the result:

```
x = lambda a, b, c: a + b + c
print(x(5, 6, 2))
```

```
13
```

Why Use Lambda Functions?

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):
  return lambda a : a * n
```

Use that function definition to make a function that always doubles the number you send in:

Example

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
22
```

Or, use the same function definition to make a function that always triples the number you send in:

Example

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

```
33
```

Or, use the same function definition to make both functions, in the same program:

Example

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

```
22
33
```

➢ Python Arrays

What is an Array?

An **array** is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

You refer to an array element by referring to the index number.

Example

Get the value of the first array item:

```
cars = ["Ford", "Volvo", "BMW"]

x = cars[0]

print(x)
```

```
Ford
```

Example

Modify the value of the first array item:

```
cars = ["Ford", "Volvo", "BMW"]

cars[0] = "Toyota"

print(cars)
```

```
['Toyota', 'Volvo', 'BMW']
```

The Length of an Array

Use the **len()** method to return the length of an array (the number of elements in an array).

Example

Return the number of elements in the cars array:

```
cars = ["Ford", "Volvo", "BMW"]

x = len(cars)

print(x)
```

```
3
```

Looping Array Elements

You can use the for in loop to loop through all the elements of an array.

Example

Print each item in the cars array:

```
cars = ["Ford", "Volvo", "BMW"]

for x in cars:
  print(x)
```

```
Ford
Volvo
BMW
```

Adding Array Elements

You can use the **append()** method to add an element to an array.

Example

Add one more element to the cars array:

```
cars = ["Ford", "Volvo", "BMW"]
cars.append("Honda")
print(cars)
```

```
['Ford', 'Volvo', 'BMW', 'Honda']
```

Removing Array Elements

You can use the **pop()** method to remove an element from the array.

Example

Delete the second element of the cars array:

```
cars = ["Ford", "Volvo", "BMW"]
cars.pop(1)
print(cars)
```

```
['Ford', 'BMW']
```

You can also use the **remove()** method to remove an element from the array.

Example

Delete the element that has the value "Volvo":

```
cars = ["Ford", "Volvo", "BMW"]
cars.remove("Volvo")
print(cars)
```

```
['Ford', 'BMW']
```

Note: The list's **remove()** method only removes the first occurrence of the specified value.

Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

➢ Python Classes and Objects

Python Classes/Objects

Python is an **object oriented** programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class:

Example

Create a class named **MyClass,** with a property named x:

```
class MyClass:
  x = 5

print(MyClass)
```

```
<class '__main__.MyClass'>
```

Create Object

Now we can use the class named **MyClass** to create objects:

Example

Create an object named p1, and print the value of x:

```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
```

```
5
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called **__init__()**, which is always executed when the class is being initiated.

Use the **__init__()** function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the **__init__()** function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

```
John
36
```

Note: The **__init__()** function is called automatically every time the class is being used to create a new object.

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

Note: The self-parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self-Parameter

The self-parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words mysillyobject and abc instead of self:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

Modify Object Properties

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.age = 40

print(p1.age)
```

```
40
```

Delete Object Properties

You can delete properties on objects by using the del keyword:

Example

Delete the age property from the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1.age

print(p1.age)
```

```
Traceback (most recent call last):
  File "demo_class7.py", line 13, in <module>
    print(p1.age)
AttributeError: 'Person' object has no attribute 'age'
```

Delete Objects

You can delete objects by using the del keyword:

Example

Delete the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1

print(p1)
```

```
Traceback (most recent call last):
  File "demo_class8.py", line 13, in <module>
    print(p1)
NameError: 'p1' is not defined
```

The pass Statement

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

Example

```
class Person:
  pass

# having an empty class definition like this, would raise an error without the pass
statement
```

➢ Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class:

Example

Create a class named Person, with **firstname** and **lastname** properties, and a **printname** method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

```
John Doe
```

Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Example

Create a class named Student, which will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

Now the Student class has the same properties and methods as the Person class.

Example

Use the Student class to create an object, and then execute the printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Add the __init__() Function

So far we have created a child class that inherits the properties and methods from its parent.

We want to add the **__init__()** function to the child class (instead of the pass keyword).

Note: The **__init__()** function is called automatically every time the class is being used to create a new object.

Example

Add the **__init__()** function to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

When you add the **__init__()** function, the child class will no longer inherit the parent's **__init__()** function.

Note: The child's **__init__()** function overrides the inheritance of the parent's **__init__()** function.

To keep the inheritance of the parent's **__init__()** function, add a call to the parent's **__init__()** function:

Example

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Now we have successfully added the __init__() function, and kept the inheritance of the parent class, and we are ready to add functionality in the __init__() function.

Use the super() Function

Python also has a **super()** function that will make the child class inherit all the methods and properties from its parent:

Example

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

By using the **super()** function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

Add Properties

Example

Add a property called graduationyear to the Student class:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019

x = Student("Mike", "Olsen")
print(x.graduationyear)
```

```
2019
```

In the example below, the year 2019 should be a variable, and passed into the Student class when creating student objects. To do so, add another parameter in the **__init__()** function:

Example

Add a year parameter, and pass the correct year when creating objects:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
print(x.graduationyear)
```

```
2019
```

Add Methods

Example

Add a method called welcome to the Student class:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

```
Welcome Mike Olsen to the class of 2019
```

➢ Python Iterators

An **iterator** is an object that contains a countable number of values.

An **iterator** is an object that can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods **__iter__()** and **__next__()**.

Iterator vs Iterable

Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from.

All these objects have a **iter()** method which is used to get an iterator:

Example

Return an iterator from a tuple, and print each value:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

```
apple
banana
cherry
```

Even strings are iterable objects, and can return an iterator:

Example

Strings are also **iterable** objects, containing a sequence of characters:

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

```
b
a
n
a
n
a
```

Looping Through an Iterator

We can also use a for loop to iterate through an iterable object:

Example

Iterate the values of a tuple:

```
mytuple = ("apple", "banana", "cherry")

for x in mytuple:
  print(x)
```

```
apple
banana
cherry
```

Example

Iterate the characters of a string:

```
mystr = "banana"

for x in mystr:
  print(x)
```

```
b
a
n
a
n
a
```

The for loop actually creates an iterator object and executes the next() method for each loop.

Create an Iterator

To create an object/class as an iterator you have to implement the methods **__iter__()** and **__next__()** to your object.

As you have learned in the Python Classes/Objects chapter, all classes have a function called **__init__()**, which allows you to do some initializing when the object is being created.

The **__iter__()** method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself.

The **__next__()** method also allows you to do operations, and must return the next item in the sequence.

Example

Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc.):

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

```
1
2
3
4
5
```

StopIteration

The example above would continue forever if you had enough next() statements, or if it was used in a for loop.

To prevent the iteration to go on forever, we can use the **StopIteration** statement.

In the **__next__()** method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

Example

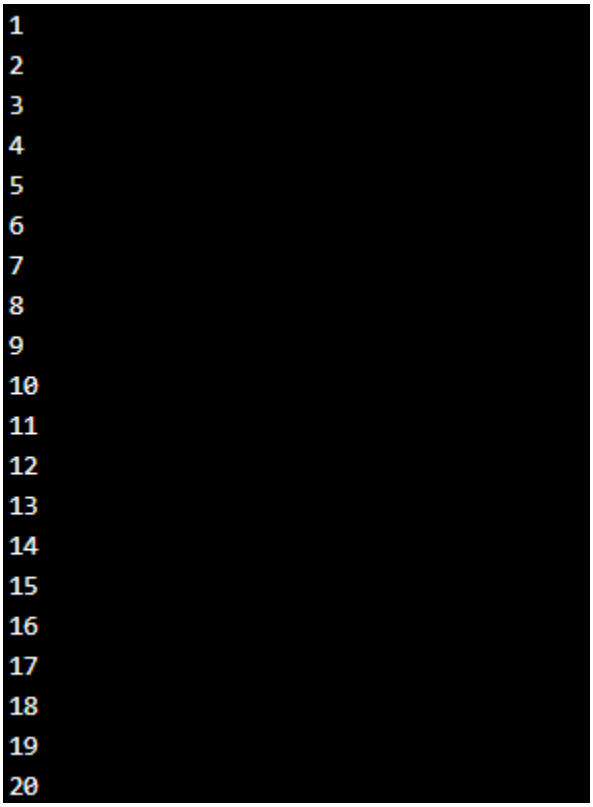
Stop after 20 iterations:

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```



➢ Python Scope

A **variable** is only available from inside the region it is created. This is called scope.

Local Scope

A **variable** created inside a function belongs to the local scope of that function, and can only be used inside that function.

Example

A **variable** created inside a function is available inside that function:

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

```
300
```

Function Inside Function

As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function:

Example

The **local variable** can be accessed from a function within the function:

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

```
300
```

Global Scope

A **variable** created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

Example

A variable created outside of a function is global and can be used by anyone:

```
x = 300

def myfunc():
  print(x)

myfunc()

print(x)
```

```
300
300
```

Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

Example

The function will print the local x, and then the code will print the global x:

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```

```
200
300
```

Global Keyword

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

The global keyword makes the variable global.

Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = 300

myfunc()

print(x)
```

```
300
```

Also, use the global keyword if you want to make a change to a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = 300

def myfunc():
  global x
  x = 200

myfunc()

print(x)
```

```
200
```

➢ Python Modules

What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example

Save this code in a file named **mymodule.py**

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named **mymodule,** and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

```
C:\Users\My Name>python demo_module1.py
Hello, Jonathan
```

Note: When using a function from a module, use the syntax:

module_name.function_name.

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example

Save this code in the file **mymodule.py**

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

```
C:\Users\My Name>python demo_module2.py
36
```

Naming a Module

You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

Example

Create an alias for **mymodule** called mx:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

```
C:\Users\My Name>python demo_module3.py
36
```

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Example

Import and use the platform module:

```
import platform

x = platform.system()
print(x)
```

```
Windows
```

Using the dir() Function

There is a **built-in function** to list all the function names (or variable names) in a module.

The **dir()** function:

Example

List all the defined names belonging to the platform module:

```
import platform

x = dir(platform)
print(x)
```

Note: The **dir()** function can be used on all modules, also the ones you create yourself.

Import From Module

You can choose to import only parts from a module, by using the from keyword.

Example

The module named **mymodule** has one function and one dictionary:

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import only the person1 dictionary from the module:

```
from mymodule import person1

print(person1["age"])
```

```
C:\Users\My Name>python demo_module6.py
36
```

Note: When importing using the from keyword, do not use the module name when referring to elements in the module.

➢ Python Datetime

Python Dates

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Example

Import the datetime module and display the current date

```
import datetime

x = datetime.datetime.now()

print(x)
```

```
2020-05-28 22:17:32.465616
```

Date Output

When we execute the code from the example above the result will be:

2020-05-28 22:15:04.448185

The date contains **year**, **month**, **day**, **hour**, **minute**, **second**, and **microsecond**.

The datetime module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

Example

Return the year and name of weekday:

```
import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

```
2020
Thursday
```

Creating Date Objects

To create a date, we can use the **datetime()** class (constructor) of the datetime module.

The **datetime()** class requires three parameters to create a date: year, month, day.

Example

Create a date object:

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

```
2020-05-17 00:00:00
```

The **datetime()** class also takes parameters for time and **timezone** (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

The strftime() Method

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

Example

Display the name of the month:

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

```
June
```

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday

%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52

%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

➢ Python Math

Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

Built-in Math Functions

The **min()** and **max()** functions can be used to find the lowest or highest value in an **iterable**:

Example

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

```
5
25
```

The **abs()** function returns the absolute (positive) value of the specified number:

Example

```
x = abs(-7.25)

print(x)
```

```
7.25
```

The **pow(x, y)** function returns the value of x to the power of **y (xy)**.

Example

Return the value of 4 to the power of 3 (same as 4 * 4 * 4):

```
x = pow(4, 3)

print(x)
```

```
64
```

The Math Module

Python has also a built-in module called math, which extends the list of mathematical functions.

To use it, you must import the math module:

```
import math
```

When you have imported the math module, you can start using methods and constants of the module.

The **math.sqrt()** method for example, returns the square root of a number:

Example

```
import math

x = math.sqrt(64)

print(x)
```

```
8.0
```

The **math.ceil()** method rounds a number upwards to its nearest integer, and the **math.floor()** method rounds a number downwards to its nearest integer, and returns the result:

Example

```
#Import math library
import math

#Round a number upward to its nearest integer
x = math.ceil(1.4)

#Round a number downward to its nearest integer
y = math.floor(1.4)

print(x)
print(y)
```

```
2
1
```

The **math.pi** constant, returns the value of PI (3.14...):

Example

```
import math

x = math.pi

print(x)
```

```
3.141592653589793
```

➢ Python Try Except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

Example

The try block will generate an exception, because x is not defined:

```
#The try block will generate an error, because x is not defined:

try:
  print(x)
except:
  print("An exception occurred")
```

```
An exception occurred
```

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

Example

This statement will raise an error, because x is not defined:

```
#This will raise an exception, because x is not defined:

print(x)
```

```
Traceback (most recent call last):
  File "demo_try_except_error.py", line 3, in <module>
    print(x)
NameError: name 'x' is not defined
```

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a **NameError** and another for other errors:

```
#The try block will generate a NameError, because x is not defined:

try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

```
Variable x is not defined
```

Else

You can use the else keyword to define a block of code to be executed if no errors were raised:

Example

In this example, the try block does not generate any error:

```
#The try block does not raise any errors, so the else block is executed:

try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

```
Hello
Nothing went wrong
```

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

```
#The finally block gets executed no matter if the try block raises any errors or not:

try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

```
Something went wrong
The 'try except' is finished
```

This can be useful to close objects and clean up resources:

Example

Try to open and write to a file that is not writable:

```
#The try block will raise an error when trying to write to a read-only file:

try:
  f = open("demofile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()

#The program can continue, without leaving the file object open
```

```
Something went wrong when writing to the file
```

The program can continue, without leaving the file object open.

Raise an exception

As a **Python** developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example

Raise an error and stop the program if x is lower than 0:

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

```
Traceback (most recent call last):
  File "demo_ref_keyword_raise.py", line 4, in <module>
    raise Exception("Sorry, no numbers below zero")
Exception: Sorry, no numbers below zero
```

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example

Raise a **TypeError** if x is not an integer:

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

```
Traceback (most recent call last):
  File "demo_ref_keyword_raise2.py", line 4, in <module>
    raise TypeError("Only integers are allowed")
TypeError: Only integers are allowed
```

➢ Python User Input

Python allows for user input.

That means we are able to ask the user for input.

The method is a bit different in Python 3.6 than Python 2.7.

Python 3.6 uses the **input()** method.

Python 2.7 uses the **raw_input()** method.

The following example asks for the username, and when you entered the username, it gets printed on the screen:

Python 3.6

```
username = input("Enter username:")
print("Username is: " + username)
```

```
C:\Users\My Name>python demo_user_input3.py
Enter username:elaf
Username is: elaf
```

Python 2.7

```
username = raw_input("Enter username:")
print("Username is: " + username)
```

```
C:\Users\My Name>python demo_user_input2.py
Enter username:elaf
Username is: elaf
```

Python stops executing when it comes to the input() function, and continues when the user has given some input.

➢ Python String Formatting

To make sure a string will display as expected, we can format the result with the format() method.

String format()

The format() method allows you to format selected parts of a string.

Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?

To control such values, add placeholders (curly brackets {}) in the text, and run the values through the **format()** method:

Example

Add a placeholder where you want to display the price:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

```
The price is 49 dollars
```

You can add parameters inside the curly brackets to specify how to convert the value:

Example

Format the price to be displayed as a number with two decimals:

```
price = 49
txt = "The price is {:.2f} dollars"
print(txt.format(price))
```

```
The price is 49.00 dollars
```

Multiple Values

If you want to use more values, just add more values to the **format()** method:

```
print(txt.format(price, itemno, count))
```

And add more placeholders:

Example

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item number 567 for 49.00 dollars.
```

Index Numbers

You can use index numbers (a number inside the curly brackets {0}) to be sure the values are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item number 567 for 49.00 dollars.
```

Also, if you want to refer to the same value more than once, use the index number:

Example

```
age = 36
name = "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
```

```
His name is John. John is 36 years old.
```

Named Indexes

You can also use named indexes by entering a name inside the curly brackets {carname}, but then you must use names when you pass the parameter values **txt.format(carname = "Ford")**:

Example

PYTHON PROGRAMMING LANGUAGE

```
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname = "Ford", model = "Mustang"))
```

```
I have a Ford, it is a Mustang.
```

REFERENCE

1- Nixon, Robin. Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. " O'Reilly Media, Inc.", 2014.

2- Davis, Michele E., and Jon A. Phillips. Learning PHP & MySQL: Step-by-Step Guide to Creating Database-Driven Web Sites. " O'Reilly Media, Inc.", 2007.

3- Sklar, David. Learning php 5. " O'Reilly Media, Inc.", 2004.

4- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

5- Lutz, Mark. Learning python: Powerful object-oriented programming. " O'Reilly Media, Inc.", 2013.

6- Bergstra, James, et al. "Theano: Deep learning on gpus with python." NIPS 2011, BigLearning Workshop, Granada, Spain. Vol. 3. Microtome Publishing., 2011.

7- https://www.w3schools.com/