OBJECT ORIENTATION PROGRAMMING:

Java uses **O**bject-**O**riented **P**rogramming (OOP), a programming style that is intended to make thinking about programming closer to thinking about the real world.

In OOP, each object is an independent unit with a **unique identity**, just as objects in the real world are.

Objects also have **characteristics**, which are used to describe them.

For example, a car can be red or blue, a mug can be full or empty, and so on. These characteristics are also called **attributes**. An attribute describes the current state of an object.

In the real world, each object behaves in its own way. The car moves, the phone rings, and so on.

The same applies to objects: **behavior**is specific to the object's type.

In summary, in object oriented programming, each object has three dimensions: **identity, attributes**, and **behavior**.

Attributes describe the object's current state, and what the object is capable of doing is demonstrated through the object's behavior.

**Getters & Setters**

**Getters**and **Setters**are used to effectively protect your data, particularly when creating classes. For each variable, the **get**method returns its value, while the **set**method sets the value.

**Getters**start with **get**, followed by the variable name, with the first letter of the variable name capitalized.

**Setters**start with **set**, followed by the variable name, with the first letter of the variable name capitalized.

Java is a pass-by-value language, which means when you pass a variable to a method, a copy of its value is created and passed to the method. Modifying the copy inside the method does not affect the original variable outside the method.

# Encapsulation

There are 4 core concepts in OOP: **encapsulation**, **inheritance**, **polymorphism**, and **abstraction**.

The idea behind **encapsulation**is to ensure that implementation details are not visible to users. The variables of one class will be hidden from the other classes, accessible only through the methods of the current class. This is called **data hiding**.

To achieve encapsulation in Java, declare the class' variables as **private**and provide public **setter**and **getter**methods to modify and view the variables' values.

In summary, **encapsulation**provides the following benefits:

- Control of the way data is accessed or modified

- More flexible and easily changed code

- Ability to change one part of the code without affecting other parts

# Inheritance

**Inheritance**is the process that enables one class to acquire the properties (methods and variables) of another. With inheritance, the information is placed in a more manageable, hierarchical order.

The class inheriting the properties of another is the **subclass**(also called derived class, or child class); the class whose properties are inherited is the **superclass**(base class, or parent class).

To inherit from a class, use the **extends**keyword.

Constructors are not member methods, and so are not inherited by subclasses.

However, the constructor of the superclass is called when the subclass is instantiated.

Example:

class A {

public A() {

System.out.println("New A");

}

}

class B extends A {

public B() {

System.out.println("New B");

}

}

class Program {

public static void main(String[ ] args) {

B obj = new B();

}

}

Output:

New A

New B

# Method Overriding

A subclass can implement a parent class method based on its requirement.

This feature is known as method **overriding.**

**3**

16 Comments

**Method Overriding**

As we saw in the previous lesson, a subclass can define a behavior that's specific to the subclass type, meaning that a subclass can implement a parent class method based on its requirement.

This feature is known as method **overriding.**

**Example:**

**CODE PLAYGROUNDJAVA**

class Animal {

public void makeSound() {

System.out.println("Grr...");

}

}

class Cat extends Animal {

public void makeSound() {

System.out.println("Meow");

}

}

Click to run

In the code above, the Cat class overrides the **makeSound()** method of its superclass Animal.

**Rules for Method Overriding:**

- Should have the **same**return type and arguments

- The **access level** cannot be more restrictive than the overridden method's access level (Example: If the superclass method is declared public, the overriding method in the sub class can be neither private nor protected)

- A method declared **final**or **static**cannot be overridden

- If a method cannot be inherited, it cannot be overridden

- Constructors cannot be overridden

Method overriding is also known as **runtime polymorphism.**

**NOTE:**

Covariant Method overriding means that when overriding a method in the child class, the return type may vary. Before java 5 it was not allowed to override any function if the return type is changed in the child class. But now it is possible only return type is subtype class.

**Method Overloading**

When methods have the same name, but different parameters, it is known as method **overloading**.

This can be very useful when you need the same method functionality for different types of parameters.

The following example illustrates a method that returns the maximum of its two parameters.

int max(int a, int b) {

if(a > b) {

return a;

}

else {

return b;

}

}

**JAVA**Copy

The method shown above will only work for parameters of type **integer**.

However, we might want to use it for **doubles**, as well. For that, you need to overload the **max**method:

**CODE PLAYGROUNDJAVA**

double max(double a, double b) {

if(a > b) {

return a;

}

else {

return b;

}

}

Click to run

Now, our **max**method will also work with **doubles**.

An overloaded method **must**have a different argument list; the parameters should differ in their type, number, or both.

Another name for method overloading is **compile-time polymorphism.**

**Abstraction**

Data **abstraction**provides the outside world with only essential information, in a process of representing essential features without including implementation details.

A good real-world example is a book. When you hear the term book, you don't know the exact specifics, such as the page count, the color, or the size, but you understand the idea, or abstraction, of a book.

The concept of **abstraction**is that we focus on essential qualities, rather than the specific characteristics of one particular example.

In Java, **abstraction**is achieved using **abstract classes** and **interfaces**.

An abstract class is defined using the **abstract**keyword.

- If a class is declared abstract it cannot be instantiated (you cannot create objects of that type).

- To use an abstract class, you have to inherit it from another class.

- Any class that contains an abstract method should be defined as abstract.

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon): a**bstract void walk();**

**Interfaces**

An **interface**is a completely abstract class that contains only abstract methods.

Some specifications for interfaces:

- Defined using the **interface**keyword.

- May contain only static final variables.

- Cannot contain a constructor because interfaces cannot be instantiated.

- Interfaces can extend other interfaces.

- A class can implement any number of interfaces.

**An example of a simple interface:**

interface Animal {

public void eat();

public void makeSound();

}opy

Interfaces have the following properties:

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.

- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.

- Methods in an interface are implicitly public.

A class can inherit from just **one**superclass, but can implement **multiple**interfaces!

**Inner Classes**

Java supports **nesting**classes; a class can be a member of another class.

Creating an inner class is quite simple. Just write a class within a class. Unlike a class, an inner class can be private. Once you declare an inner class private, it cannot be accessed from an object outside the class.

**Example:**

**CODE PLAYGROUNDJAVA**

class Robot {

int id;

Robot(int i) {

id = i;

Brain b = new Brain();

b.think();

}

private class Brain {

public void think() {

System.out.println(id + " is thinking");

}

}

}

Click to run

The class **Robot**has an inner class **Brain**. The inner class can access all of the member variables and methods of its outer class, but it cannot be accessed from any outside class.

**Enums**

An Enum is a special type used to define collections of constants.

**Here is a simple Enum example:**

enum Rank {

SOLDIER,

SERGEANT,

CAPTAIN

}

**JAVA**Copy

Note that the values are **comma-separated.**

You can refer to the constants in the enum above with the **dot**syntax.

Rank a = Rank.SOLDIER;

**JAVA**Copy

Basically, Enums define variables that represent members of a fixed set.

**Java API**

The Java API is a collection of classes and interfaces that have been written for you to use.

The Java API Documentation with all of the available APIs can be located on the Oracle website at

**http://docs.oracle.com/javase/7/docs/api/**

Once you locate the package you want to use, you need to import it into your code.

The package can be imported using the import keyword.

**For example:**

import java.awt.\*;

**JAVA**Copy

The **awt**package contains all of the classes for creating user interfaces and for painting graphics and images.

The wildcard character (\*) is used to import all of the classes in the package.

**Exceptions**

An **exception**is a problem that occurs during program execution. Exceptions cause abnormal termination of the program.

**Exception handling** is a powerful mechanism that handles runtime errors to maintain normal application flow.

An exception can occur for many different reasons. Some examples:

- A user has entered invalid data.

- A file that needs to be opened cannot be found.

- A network connection has been lost in the middle of communications.

- Insufficient memory and other issues related to physical resources.

As you can see, exceptions are caused by user error, programmer error, or physical resource issues. However, a well-written program should handle all possible exceptions.

**throw**

The **throw**keyword allows you to manually generate exceptions from your methods. Some of the numerous available exception types include the IndexOutOfBoundsException, IllegalArgumentException, ArithmeticException, and so on.

For example, we can throw an ArithmeticException in our method when the parameter is 0.

**CODE PLAYGROUNDJAVA**

int div(int a, int b) throws ArithmeticException {

if(b == 0) {

throw new ArithmeticException("Division by Zero");

} else {

return a / b;

}

}

Click to run

The **throws**statement in the method definition defines the type of Exception(s) the method can throw.

Next, the **throw**keyword throws the corresponding exception, along with a custom message.

If we call the **div**method with the second parameter equal to 0, it will throw an ArithmeticException with the message "Division by Zero".

Multiple exceptions can be defined in the throws statement using a **comma-separated** list.

**ArrayList**

The Java API provides special classes to store and manipulate groups of objects.

One such class is the **ArrayList**. Standard Java arrays are of a fixed length, which means that after they are created, they cannot expand or shrink.

On the other hand, **ArrayLists**are created with an initial size, but when this size is exceeded, the collection is automatically enlarged.

When objects are removed, the ArrayList may shrink in size. Note that the ArrayList class is in the **java.util** package, so it's necessary to import it before using it.

Create an ArrayList as you would any object.

import java.util.ArrayList;

//...

ArrayList colors = new ArrayList();

**JAVA**Copy

You can optionally specify a **capacity**and **type**of objects the ArrayList will hold:

ArrayList<String> colors = new ArrayList<String>(10);

**JAVA**Copy

The code above defines an ArrayList of Strings with 10 as its initial size.

ArrayLists store objects. Thus, the type specified must be a class type. You cannot pass, for example, **int**as the objects' type. Instead, use the special **class types** that correspond to the desired value type, such as **Integer**for int, **Double**for double, and so on.

**ArrayList**

The **ArrayList**class provides a number of useful methods for manipulating its objects.

The **add()** method adds new objects to the ArrayList. Conversely, the remove() method removes objects from the ArrayList.

**Example:**

**CODE PLAYGROUNDJAVA**

import java.util.ArrayList;

public class MyClass {

public static void main(String[ ] args) {

ArrayList<String> colors = new ArrayList<String>();

colors.add("Red");

colors.add("Blue");

colors.add("Green");

colors.add("Orange");

colors.remove("Green");

System.out.println(colors);

}

}

Click to run

Other useful methods include the following:

- **contains()**: Returns true if the list contains the specified element

- **get(int index)**: Returns the element at the specified position in the list

- **size()**: Returns the number of elements in the list

- **clear()**: Removes all of the elements from the list

Note: As with arrays, the indexing starts with 0.

**LinkedList**

The **LinkedList**is very similar in syntax to the **ArrayList**.

You can easily change an ArrayList to a LinkedList by changing the object type.

import java.util.LinkedList;

public class MyClass {

public static void main(String[ ] args) {

LinkedList<String> c = new LinkedList<String>();

c.add("Red");

c.add("Blue");

c.add("Green");

c.add("Orange");

c.remove("Green");

System.out.println(c);

}

}

Click to run

You cannot specify an initial capacity for the **LinkedList**.

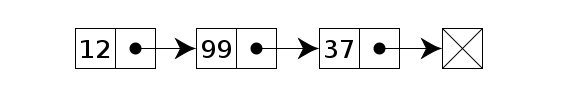
**LinkedList vs. ArrayList**

The most notable difference between the **LinkedList**and the **ArrayList**is in the way they store objects.

The ArrayList is better for **storing**and **accessing**data, as it is very similar to a normal array.

The LinkedList is better for **manipulating**data, such as making numerous inserts and deletes.

In addition to storing the object, the LinkedList stores the memory address (or link) of the element that follows it. It's called a LinkedList because each element contains a link to the neighboring element.



You can use the enhanced for loop to iterate over its elements.

LinkedList<String> c = new LinkedList<String>();

c.add("Red");

c.add("Blue");

c.add("Green");

c.add("Orange");

c.remove("Green");

for(String s: c) {

System.out.println(s);

}

Click to run

Summary:

- Use an **ArrayList**when you need rapid access to your data.

- Use a **LinkedList**when you need to make a large number of inserts and/or deletes.

**HashMap**

Arrays and Lists store elements as ordered collections, with each element given an integer index.

**HashMap**is used for storing data collections as key and value pairs. One object is used as a key (index) to another object (the value).

The **put**, **remove**, and **get**methods are used to add, delete, and access values in the HashMap.

**Example:**

import java.util.HashMap;

public class MyClass {

public static void main(String[ ] args) {

HashMap<String, Integer> points = new HashMap<String, Integer>();

points.put("Amy", 154);

points.put("Dave", 42);

points.put("Rob", 733);

System.out.println(points.get("Dave"));

}

}

Click to run

We have created a HashMap with Strings as its keys and Integers as its values.

Use the **get** method and the corresponding key to access the HashMap elements.

A **Set**is a collection that cannot contain duplicate elements. It models the mathematical set abstraction.

One of the implementations of the Set is the **HashSet**class.

**Example:**

import java.util.HashSet;

public class MyClass {

public static void main(String[ ] args) {

HashSet<String> set = new HashSet<String>();

set.add("A");

set.add("B");

set.add("C");

System.out.println(set);

}

}

Click to run

You can use the **size()** method to get the number of elements in the HashSet.

**LinkedHashSet**

The HashSet class does not automatically retain the order of the elements as they're added. To order the elements, use a **LinkedHashSet**, which maintains a linked list of the set's elements in the order in which they were inserted.

**What is hashing?**

A hash table stores information through a mechanism called hashing, in which a key's informational content is used to determine a unique value called a hash code.

So, basically, each element in the HashSet is associated with its unique hash code.

You've learned about the various collection types that are available in Java, including **Lists**, **Maps**, and **Sets**. The choice of which one to use is specific to the data you need to store and manipulate.

**Sorting Lists**

For the manipulation of data in different collection types, the Java API provides a **Collections**class, which is included in the java.util package.

One of the most popular **Collections**class methods is **sort()**, which sorts the elements of your collection type. The methods in the Collections class are **static**, so you don't need a Collections object to call them.

**Example:**

public class MyClass {

public static void main(String[ ] args) {

ArrayList<String> animals = new ArrayList<String>();

animals.add("tiger");

animals.add("cat");

animals.add("snake");

animals.add("dog");

Collections.sort(animals);

System.out.println(animals);

}

}

Click to run

As you can see, the elements have been sorted alphabetically.

**Working with Files**

The **java.io package** includes a **File**class that allows you to work with files.

To start, create a **File**object and specify the path of the file in the constructor.

import java.io.File;

...

File file = new File("C:\\data\\input-file.txt");

**JAVA**Copy

With the **exists()** method, you can determine whether a file exists.

import java.io.File;

public class MyClass {

public static void main(String[ ] args) {

File x = new File("C:\\sololearn\\test.txt");

if(x.exists()) {

System.out.println(x.getName() + "exists!");

}

else {

System.out.println("The file does not exist");

}

}

}py

The code above prints a message stating whether or not the file exists at the specified path.

The **getName()** method returns the name of the file.

Note that we used double backslashes in the path, as one backslash should be escaped in the path String.

**Reading a File**

Files are useful for storing and retrieving data, and there are a number of ways to read from a file.

 One of the simplest ways is to use the **Scanner**class from the **java.util**package.

The constructor of the **Scanner**class can take a **File**object as input.

To read the contents of a text file at the path "C:\\sololearn\\test.txt", we would need to create a File object with the corresponding path and pass it to the Scanner object.

try {

File x = new File("C:\\sololearn\\test.txt");

Scanner sc = new Scanner(x);

}

catch (FileNotFoundException e) {

}

We surrounded the code with a try/catch block, because there's a chance that the file may not exist.

**Reading a File**

The **Scanner**class inherits from the **Iterator**, so it behaves like one.

We can use the Scanner object's **next()** method to read the file's contents.

try {

File x = new File("C:\\sololearn\\test.txt");

Scanner sc = new Scanner(x);

while(sc.hasNext()) {

System.out.println(sc.next());

}

sc.close();

} catch (FileNotFoundException e) {

System.out.println("Error");

}Copy

The file's contents are output word by word, because the **next()** method returns each word separately.

It is always good practice to close a file when finished working with it. One way to do this is to use the Scanner's **close()** method.

**Formatter**, another useful class in the **java.util** package, is used to create content and write it to files.

**Example:**

import java.util.Formatter;

public class MyClass {

public static void main(String[ ] args) {

try {

Formatter f = new Formatter("C:\\sololearn\\test.txt");

} catch (Exception e) {

System.out.println("Error");

}

}

}

opy

This creates an empty file at the specified path. If the file already exists, this will overwrite it.

**Writing to Files**

Once the file is created, you can write content to it using the same Formatter object's **format()** method.

**Example:**

import java.util.Formatter;

public class MyClass {

public static void main(String[ ] args) {

try {

Formatter f = new Formatter("C:\\sololearn\\test.txt");

f.format("%s %s %s", "1","John", "Smith \r\n");

f.format("%s %s %s", "2","Amy", "Brown");

f.close();

} catch (Exception e) {

System.out.println("Error");

}

}

}

Click to run

The **format()** method formats its parameters according to its first parameter.

%s mean a string and get's replaced by the first parameter after the format. The second %s get's replaced by the next one, and so on. So, the format **%s %s %s** denotes three strings that are separated with spaces.

Note: **\r\n** is the newline symbol in Windows.

The code above creates a file with the following content:

1 John Smith

2 Amy BrownCopy

Don't forget to **close**the file once you're finished writing to it!

# HTML:

## **What is HTML?**

* HTML stands for Hyper Text Markup Language
* HTML is the standard markup language for creating Web pages
* HTML describes the structure of a Web page
* HTML consists of a series of elements
* HTML elements tell the browser how to display the content
* HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

## **Web Browsers**

The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.

A browser does not display the HTML tags, but uses them to determine how to display the document:

The <!DOCTYPE> declaration represents the document type, and helps browsers to display web pages correctly.

The <!DOCTYPE> declaration is not case sensitive.

HTML headings are defined with the <h1> to <h6> tags.

HTML paragraphs are defined with the <p> tag:

HTML links are defined with the <a> tag:

<a href="https://www.w3schools.com">This is a link</a>

HTML images are defined with the <img> tag.

The source file (src), alternative text (alt), width, and height are provided as attributes:

### **Example**

<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">

The <body> element defines the document's body.

It has a start tag <body> and an end tag </body>.

## **HTML Attributes**

* All HTML elements can have **attributes**
* The href attribute of <a> specifies the URL of the page the link goes to
* The src attribute of <img> specifies the path to the image to be displayed
* The width and height attributes of <img> provide size information for images
* The alt attribute of <img> provides an alternate text for an image
* The style attribute is used to add styles to an element, such as color, font, size, and more
* The lang attribute of the <html> tag declares the language of the Web page
* The title attribute defines some extra information about an element

## **The src Attribute**

The <img> tag is used to embed an image in an HTML page. The src attribute specifies the path to the image to be displayed:

### **Example**

<img src="img\_girl.jpg">

There are two ways to specify the URL in the src attribute:

**1. Absolute URL** - Links to an external image that is hosted on another website. Example: src="https://www.w3schools.com/images/img\_girl.jpg".

**Notes:** External images might be under copyright. If you do not get permission to use it, you may be in violation of copyright laws. In addition, you cannot control external images; it can suddenly be removed or changed.

**2. Relative URL** - Links to an image that is hosted within the website. Here, the URL does not include the domain name. If the URL begins without a slash, it will be relative to the current page. Example: src="img\_girl.jpg". If the URL begins with a slash, it will be relative to the domain. Example: src="/images/img\_girl.jpg".

**Tip:** It is almost always best to use relative URLs. They will not break if you change domain.

## **The style Attribute**

The style attribute is used to add styles to an element, such as color, font, size, and more.

### **Example**

<p style="color:red;">This is a red paragraph.</p>

## **The alt Attribute**

The required alt attribute for the <img> tag specifies an alternate text for an image, if the image for some reason cannot be displayed. This can be due to a slow connection, or an error in the src attribute, or if the user uses a screen reader.

### **Example**

<img src="img\_girl.jpg" alt="Girl with a jacket">

## **The title Attribute**

The title attribute defines some extra information about an element.

The value of the title attribute will be displayed as a tooltip when you mouse over the element:

### **Example**

<p title="I'm a tooltip">This is a paragraph.</p>

## **We Suggest: Always Quote Attribute Values**

The HTML standard does not require quotes around attribute values.

However, we recommends quotes in HTML, and **demands** quotes for stricter document types like XHTML.

Sometimes you have to use quotes. This example will not display the title attribute correctly, because it contains a space:

### **Example**

<p title = About W3Schools>

# HTML Paragraphs

tag reference contains additional information about HTML elements and their attributes.

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<p>](https://www.w3schools.com/tags/tag_p.asp) | Defines a paragraph |
| [<hr>](https://www.w3schools.com/tags/tag_hr.asp) | Defines a thematic change in the content |
| [<br>](https://www.w3schools.com/tags/tag_br.asp) | Inserts a single line break |
| [<pre>](https://www.w3schools.com/tags/tag_pre.asp) | Defines pre-formatted text |

The HTML <p> element defines a paragraph.

A paragraph always starts on a new line, and browsers automatically add some white space (a margin) before and after a paragraph.

### **Example**

<p>This is a paragraph.</p>  
<p>This is another paragraph.</p>

The <hr> element is used to separate content (or define a change) in an HTML page:

### **Example**

<h1>This is heading 1</h1>  
<p>This is some text.</p>  
<hr>  
<h2>This is heading 2</h2>  
<p>This is some other text.</p>  
<hr>

The <hr> tag is an empty tag, which means that it has no end tag.

## **HTML Line Breaks**

The HTML <br> element defines a line break.

Use <br> if you want a line break (a new line) without starting a new paragraph:

### **Example**

<p>This is<br>a paragraph<br>with line breaks.</p>

The <br> tag is an empty tag, which means that it has no end tag.

HTML Style:

The HTML style attribute is used to add styles to an element, such as color, font, size, and more.

* Use the style attribute for styling HTML elements

<tagname style="property:value;">

* Use background-color for background color

<body style="background-color:powderblue;">

* Use color for text colors

<h1 style="color:blue;">This is a heading</h1>  
<p style="color:red;">This is a paragraph.</p>

* Use font-family for text fonts

<h1 style="font-family:verdana;">This is a heading</h1>  
<p style="font-family:courier;">This is a paragraph.</p>

* Use font-size for text sizes

<h1 style="font-size:300%;">This is a heading</h1>  
<p style="font-size:160%;">This is a paragraph.</p>

* Use text-align for text alignment

<h1 style="text-align:center;">Centered Heading</h1>  
<p style="text-align:center;">Centered paragraph.</p>

## **HTML Formatting Elements**

Formatting elements were designed to display special types of text:

* <b> - Bold text
* <strong> - Important text
* <i> - Italic text
* <em> - Emphasized text
* <mark> - Marked text
* <small> - Smaller text
* <del> - Deleted text
* <ins> - Inserted text
* <sub> - Subscript text

## <sup> - Superscript text

## **What is CSS?**

Cascading Style Sheets (CSS) is used to format the layout of a webpage.

With CSS, you can control the color, font, the size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more!

* Use the HTML style attribute for inline styling
* Use the HTML <style> element to define internal CSS
* Use the HTML <link> element to refer to an external CSS file
* Use the HTML <head> element to store <style> and <link> elements
* Use the CSS color property for text colors
* Use the CSS font-family property for text fonts
* Use the CSS font-size property for text sizes
* Use the CSS border property for borders
* p {  
    border: 2px solid powderblue;  
  }
* Use the CSS padding property for space inside the border
* p {  
    border: 2px solid powderblue;  
    padding: 30px;  
  }
* Use the CSS margin property for space outside the border

HTML Favicon

A favicon is a small image displayed next to the page title in the browser tab.

<link rel="icon" type="image/x-icon" href="/images/favicon.ico">

## **HTML Table Tags**

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<table>](https://www.w3schools.com/tags/tag_table.asp) | Defines a table |
| [<th>](https://www.w3schools.com/tags/tag_th.asp) | Defines a header cell in a table |
| [<tr>](https://www.w3schools.com/tags/tag_tr.asp) | Defines a row in a table |
| [<td>](https://www.w3schools.com/tags/tag_td.asp) | Defines a cell in a table |
| [<caption>](https://www.w3schools.com/tags/tag_caption.asp) | Defines a table caption |
| [<colgroup>](https://www.w3schools.com/tags/tag_colgroup.asp) | Specifies a group of one or more columns in a table for formatting |
| [<col>](https://www.w3schools.com/tags/tag_col.asp) | Specifies column properties for each column within a <colgroup> element |
| [<thead>](https://www.w3schools.com/tags/tag_thead.asp) | Groups the header content in a table |
| [<tbody>](https://www.w3schools.com/tags/tag_tbody.asp) | Groups the body content in a table |
| [<tfoot>](https://www.w3schools.com/tags/tag_tfoot.asp) | Groups the footer content in a table |

For a complete list of all available HTML tags, visit this [HTML Tag Reference](https://www.w3schools.com/tags/default.asp).

# HTML Table Borders

The following values are allowed:

* dotted
* dashed
* solid
* double
* groove
* ridge
* inset
* outset
* none
* hidden

## **Dotted Table Borders**

With the border-style property, you can set the appearance of the border.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

### **Example**

 th, td {  
  border-color: #96D4D4;  
}

## **Border Color**

With the border-color property, you can set the color of the border.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

## **HTML List Tags**

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<ul>](https://www.w3schools.com/tags/tag_ul.asp) | Defines an unordered list |
| [<ol>](https://www.w3schools.com/tags/tag_ol.asp) | Defines an ordered list |
| [<li>](https://www.w3schools.com/tags/tag_li.asp) | Defines a list item |
| [<dl>](https://www.w3schools.com/tags/tag_dl.asp) | Defines a description list |
| [<dt>](https://www.w3schools.com/tags/tag_dt.asp) | Defines a term in a description list |
| [<dd>](https://www.w3schools.com/tags/tag_dd.asp) | Describes the term in a description list |

## **Block-level Elements**

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are: <p> and <div>.

Here are the block-level elements in HTML:

[<address>](https://www.w3schools.com/tags/tag_address.asp)[<article>](https://www.w3schools.com/tags/tag_article.asp)[<aside>](https://www.w3schools.com/tags/tag_aside.asp)[<blockquote>](https://www.w3schools.com/tags/tag_blockquote.asp)[<canvas>](https://www.w3schools.com/tags/tag_canvas.asp)[<dd>](https://www.w3schools.com/tags/tag_dd.asp)[<footer>](https://www.w3schools.com/tags/tag_footer.asp)

[<form>](https://www.w3schools.com/tags/tag_form.asp)[<h1>-<h6>](https://www.w3schools.com/tags/tag_hn.asp)[<header>](https://www.w3schools.com/tags/tag_header.asp)[<hr>](https://www.w3schools.com/tags/tag_hr.asp)[<li>](https://www.w3schools.com/tags/tag_li.asp)[<main>](https://www.w3schools.com/tags/tag_main.asp)[<noscript>](https://www.w3schools.com/tags/tag_noscript.asp)[<nav>](https://www.w3schools.com/tags/tag_nav.asp)

[<ol>](https://www.w3schools.com/tags/tag_ol.asp)[<p>](https://www.w3schools.com/tags/tag_p.asp)[<pre>](https://www.w3schools.com/tags/tag_pre.asp)[<section>](https://www.w3schools.com/tags/tag_section.asp)[<table>](https://www.w3schools.com/tags/tag_table.asp)[<tfoot>](https://www.w3schools.com/tags/tag_tfoot.asp)[<ul>](https://www.w3schools.com/tags/tag_ul.asp)[<video>](https://www.w3schools.com/tags/tag_video.asp)

## **Inline Elements**

An inline element does not start on a new line.

An inline element only takes up as much width as necessary.

This is a <span> element inside a paragraph.

Here are the inline elements in HTML:

[<a>](https://www.w3schools.com/tags/tag_a.asp)

[<em>](https://www.w3schools.com/tags/tag_em.asp)

[<dfn>](https://www.w3schools.com/tags/tag_dfn.asp)

[<i>](https://www.w3schools.com/tags/tag_i.asp)

[<img>](https://www.w3schools.com/tags/tag_img.asp)

[<input>](https://www.w3schools.com/tags/tag_input.asp)

[<kbd>](https://www.w3schools.com/tags/tag_kbd.asp)

[<label>](https://www.w3schools.com/tags/tag_label.asp)

[<map>](https://www.w3schools.com/tags/tag_map.asp)

[<object>](https://www.w3schools.com/tags/tag_object.asp)

[<output>](https://www.w3schools.com/tags/tag_output.asp)

[<q>](https://www.w3schools.com/tags/tag_q.asp)

[<abbr>](https://www.w3schools.com/tags/tag_abbr.asp)

[<acronym>](https://www.w3schools.com/tags/tag_acronym.asp)

[<b>](https://www.w3schools.com/tags/tag_b.asp)

[<bdo>](https://www.w3schools.com/tags/tag_bdo.asp)

[<big>](https://www.w3schools.com/tags/tag_big.asp)

[<br>](https://www.w3schools.com/tags/tag_br.asp)

[<button>](https://www.w3schools.com/tags/tag_button.asp)

[<cite>](https://www.w3schools.com/tags/tag_cite.asp)

[<code>](https://www.w3schools.com/tags/tag_code.asp)

[<samp>](https://www.w3schools.com/tags/tag_samp.asp)

[<script>](https://www.w3schools.com/tags/tag_script.asp)

[<select>](https://www.w3schools.com/tags/tag_select.asp)

[<small>](https://www.w3schools.com/tags/tag_small.asp)

[<span>](https://www.w3schools.com/tags/tag_span.asp)

[<strong>](https://www.w3schools.com/tags/tag_strong.asp)

[<sub>](https://www.w3schools.com/tags/tag_sub.asp)

[<sup>](https://www.w3schools.com/tags/tag_sup.asp)

[<textarea>](https://www.w3schools.com/tags/tag_textarea.asp)

[<time>](https://www.w3schools.com/tags/tag_time.asp)

[<tt>](https://www.w3schools.com/tags/tag_tt.asp)

[<var>](https://www.w3schools.com/tags/tag_var.asp)

# HTML class Attribute

The HTML class attribute is used to specify a class for an HTML element.

Multiple HTML elements can share the same class.

## **Using The class Attribute**

The class attribute is often used to point to a class name in a style sheet. It can also be used by a JavaScript to access and manipulate elements with the specific class name.

In the following example we have three <div> elements with a class attribute with the value of "city". All of the three <div> elements will be styled equally according to the .city style definition in the head section:

* The HTML class attribute specifies one or more class names for an element
* Classes are used by CSS and JavaScript to select and access specific elements
* The class attribute can be used on any HTML element
* The class name is case sensitive
* Different HTML elements can point to the same class name
* JavaScript can access elements with a specific class name with the getElementsByClassName() method

# HTML id Attribute

The HTML id attribute is used to specify a unique id for an HTML element.

You cannot have more than one element with the same id in an HTML document.

## **Using The id Attribute**

The id attribute specifies a unique id for an HTML element. The value of the id attribute must be unique within the HTML document.

The id attribute is used to point to a specific style declaration in a style sheet. It is also used by JavaScript to access and manipulate the element with the specific id.

The syntax for id is: write a hash character (#), followed by an id name. Then, define the CSS properties within curly braces {}.

In the following example we have an <h1> element that points to the id name "myHeader". This <h1> element will be styled according to the #myHeader style definition in the head section:

## **Difference Between Class and ID**

A class name can be used by multiple HTML elements, while an id name must only be used by one HTML element within the page:

HTML Iframes

An HTML iframe is used to display a web page within a web page.

### **Syntax**

<iframe src="*url*" title="description"></iframe>

### **Example**

<iframe src="demo\_iframe.htm" height="200" width="300" title="Iframe Example"></iframe>

you can add the style attribute and use the CSS height and width properties:

<iframe src="demo\_iframe.htm" style="height:200px;width:300px;" title="Iframe Example"></iframe>

## **Iframe - Remove the Border**

By default, an iframe has a border around it.

To remove the border, add the style attribute and use the CSS border property:

### **Example**

<iframe src="demo\_iframe.htm" style="border:none;" title="Iframe Example"></iframe>

With CSS, you can also change the size, style and color of the iframe's border:

### **Example**

<iframe src="demo\_iframe.htm" style="border:2px solid red;" title="Iframe Example"></iframe>

## **Iframe - Target for a Link**

An iframe can be used as the target frame for a link.

The target attribute of the link must refer to the name attribute of the iframe:

### **Example**

<iframe src="demo\_iframe.htm" name="iframe\_a" title="Iframe Example"></iframe>  
  
<p><a href="https://www.w3schools.com" target="iframe\_a">W3Schools.com</a></p>

## **Chapter Summary**

* The HTML <iframe> tag specifies an inline frame
* The src attribute defines the URL of the page to embed
* Always include a title attribute (for screen readers)
* The height and width attributes specify the size of the iframe
* Use border:none; to remove the border around the iframe

## **The HTML <script> Tag**

The HTML <script> tag is used to define a client-side script (JavaScript).

The <script> element either contains script statements, or it points to an external script file through the src attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

To select an HTML element, JavaScript most often uses the document.getElementById() method.

This JavaScript example writes "Hello JavaScript!" into an HTML element with id="demo":

### **Example**

<script>  
document.getElementById("demo").innerHTML = "Hello JavaScript!";  
</script>

## **HTML Script Tags**

|  |  |
| --- | --- |
| **Tag** | **Description** |
| [<script>](https://www.w3schools.com/tags/tag_script.asp) | Defines a client-side script |
| [<noscript>](https://www.w3schools.com/tags/tag_noscript.asp) | Defines an alternate content for users that do not support client-side scripts |

HTML - The Head Element

The HTML <head> element is a container for the following elements: <title>, <style>, <meta>, <link>, <script>, and <base>.

HTML Responsive Web Design

Responsive web design is about creating web pages that look good on all devices!

A responsive web design will automatically adjust for different screen sizes and viewports.

## **What is Responsive Web Design?**

Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones):

## **Setting The Viewport**

To create a responsive website, add the following <meta> tag to all your web pages:

### **Example**

<meta name="viewport" content="width=device-width, initial-scale=1.0">

This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

Here is an example of a web page without the viewport meta tag, and the same web page with the viewport meta tag:

Without the viewport meta tag:  
[](https://www.w3schools.com/html/example_withoutviewport.htm)

With the viewport meta tag:  
[](https://www.w3schools.com/html/example_withviewport.htm)

## **Responsive Web Page - Full Example**

A responsive web page should look good on large desktop screens and on small mobile phones.

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<style>

\* {

box-sizing: border-box;

}

.menu {

float: left;

width: 20%;

text-align: center;

}

.menu a {

background-color: #e5e5e5;

padding: 8px;

margin-top: 7px;

display: block;

width: 100%;

color: black;

}

.main {

float: left;

width: 60%;

padding: 0 20px;

}

.right {

background-color: #e5e5e5;

float: left;

width: 20%;

padding: 15px;

margin-top: 7px;

text-align: center;

}

@media only screen and (max-width: 620px) {

/\* For mobile phones: \*/

.menu, .main, .right {

width: 100%;

}

}

</style>

</head>

<body style="font-family:Verdana;color:#aaaaaa;">

<div style="background-color:#e5e5e5;padding:15px;text-align:center;">

<h1>Hello World</h1>

</div>

<div style="overflow:auto">

<div class="menu">

<a href="#">Link 1</a>

<a href="#">Link 2</a>

<a href="#">Link 3</a>

<a href="#">Link 4</a>

</div>

<div class="main">

<h2>Lorum Ipsum</h2>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>

</div>

<div class="right">

<h2>About</h2>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>

</div>

</div>

<div style="background-color:#e5e5e5;text-align:center;padding:10px;margin-top:7px;">© copyright w3schools.com</div>

</body>

</html>

HTML Forms

An HTML form is used to collect user input. The user input is most often sent to a server for processing.

## **The <input> Element**

The HTML <input> element is the most used form element.

An <input> element can be displayed in many ways, depending on the type attribute.

Here are some examples:

|  |  |
| --- | --- |
| **Type** | **Description** |
| <input type="text"> | Displays a single-line text input field |
| <input type="radio"> | Displays a radio button (for selecting one of many choices) |
| <input type="checkbox"> | Displays a checkbox (for selecting zero or more of many choices) |
| <input type="submit"> | Displays a submit button (for submitting the form) |
| <input type="button"> | Displays a clickable button |

⭐ The **<video>** tag embeds video in web pages

⭐ The **<source>** tag lets you add multiple media files

⭐ The **controls** attribute adds default video controls

⭐ The <audio> tag embeds sound into a web page

⭐ The autoplay, muted and loop attributes control how the multimedia behaves

⭐ **<article>** represents a self-contained, independent piece of content

⭐ **<section>** is used to divide content into parts

⭐ **<aside>** is used for secondary, additional or somehow related content

The **style** attribute to customize your web pages’...

⭐ colors

⭐ font sizes

⭐ alignments

⭐ borders

⭐ and more!

⭐ There are two types of HTML element: inline and block

⭐ **Block-level** elements always start on (and end with) a new line

⭐ **Inline** elements don’t start on a new line

⭐ <div> is a container that groups elements

⭐ <div> is a block-level element

⭐ <div> is used to apply the same style to grouped elements

⭐ Add tables to web pages with the <table> container tag

⭐ Add rows with the <tr> container tag

⭐ Add data cells with the <td> container tag

⭐ Add header cells with the <th> container tag

⭐**JavaScript**makes your web pages interactive

⭐ The **onclick** attribute triggers actions when a user clicks or taps on an element

⭐ The **alert()** function generates a message with an OK button