

## 2 : Fundamentals of Asynchronous JavaScript and XML (AJAX)

IT3406 – Web Application Development II

Level II - Semester 3

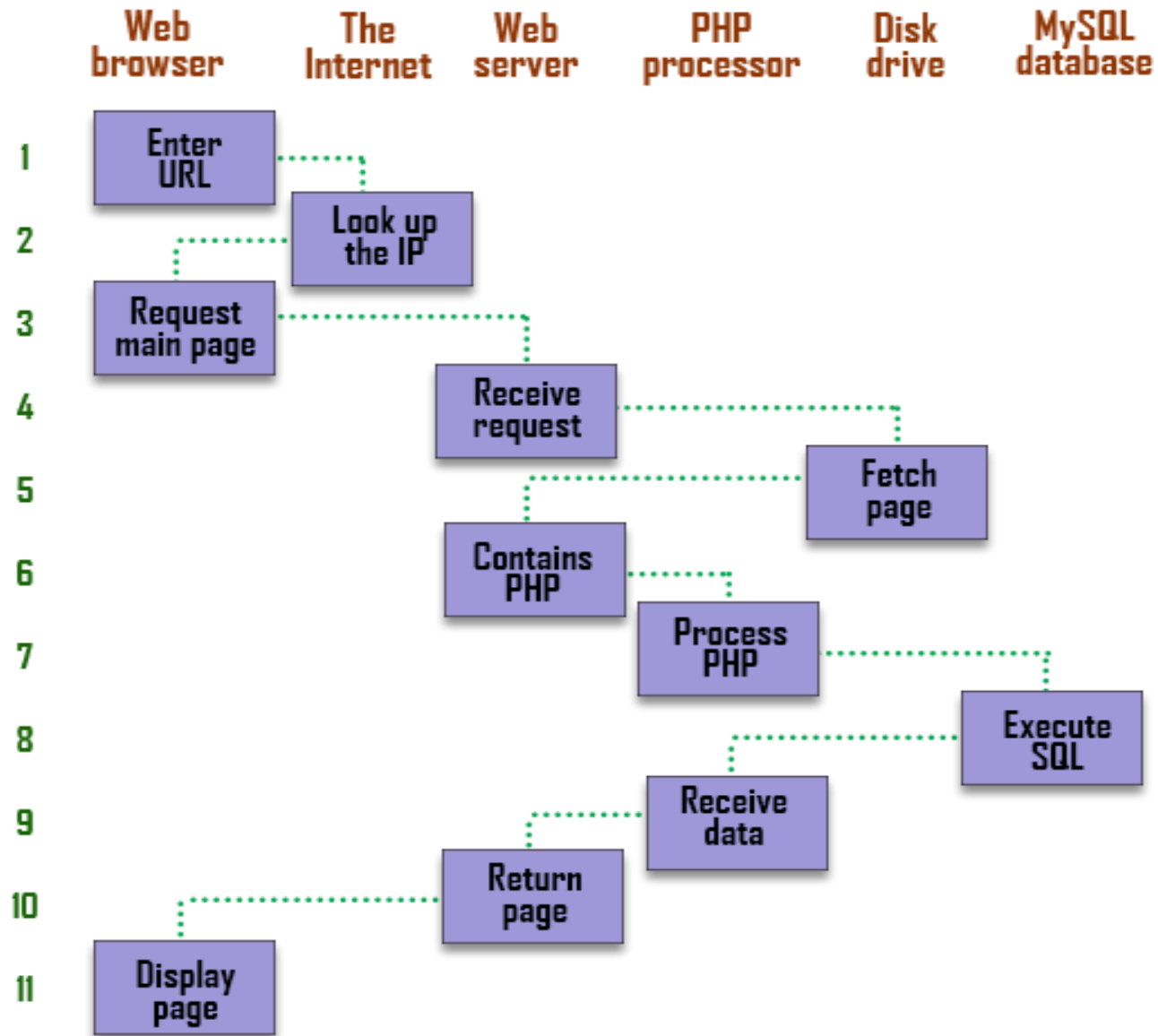
# **Traditional technologies used in web development**

# Internet in the past...

- Served static content.
- A very dry experience.
- Basic Request/Response procedure,
  1. Web browser asking the web server to send it a web page.
  2. Server sending back the page.
  3. Browser then takes care of displaying the page.

# Dynamic web pages

- Dynamic web pages altered the overall internet experience.
  - Ex: Shopping carts, Social media, Search engines etc..
- Popularity of tools and technologies to support dynamic web content.
  - Ex: PHP, MySQL, Perl etc..



An example dynamic client/server request/response sequence

# PHP, MySQL, JavaScript, and CSS

- Many web development technologies.
- PHP, MySQL, JavaScript, and CSS?
  - The Symbiotic nature of PHP and MYSQL;
  - JavaScript became an essential part in manipulating CSS;
  - It was possible to build dynamic and interactive when using PHP, MySQL, JavaScript, and CSS together.
  - Resulted in wider popularity and adoption.

# PHP

- A scripting language,
  - Speedy and seamlessly integrating with HTML code.

```
index.php
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <?php
5      echo "Hello World. Today is ".date("l").". ";
6  ?>
7  How are you?
8  </body>
9  </html>
```

- Modify HTML on the fly | Process a credit card | Add user details to a database | Fetch information from a third-party website | etc.....

# Activity

- What are the alternatives for PHP for server side scripting?
  - Compare and contrast advantages and disadvantages of the alternatives and PHP.



# MySQL

- In early days,
  - Developers used 'flat' text files to store data:
    - usernames, passwords etc..
  - Unsecure | Can become unwieldy/too large | Difficulties in merging files | Difficulties in performing complex searches.
- Relational databases became essential.
  - Structured Query Language.

# MySQL contd.

- Popularity of MySQL...

- Free to use and installed on vast numbers of Internet web servers.
- Robust and exceptionally fast database management system that uses English-like commands. Example:

```
SELECT surname,firstname FROM users WHERE email='jsmith@mysite.com';
```

- When used alongside PHP and other server-side scripting languages, you can make calls directly to MySQL without having to run the MySQL program yourself or using its command-line interface.

# Activity

- What are the alternatives for MySQL?
  - Compare and contrast advantages and disadvantages of the alternatives and MySQL.

# JavaScript

- Created to enable scripting access to all the elements of an HTML document.
  - Dynamic user interactions: validations | prompts | animations
  - Combined with CSS, JavaScript is the power behind dynamic web pages.

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <script type="text/javascript">
5  |   document.write("Hello World. Today is " + Date() );
6  </script>
7  How are you?
8  </body>
9  </html>
```

Hello World. Today is Thu Dec 31 2020 16:24:57 GMT+0530 (India Standard Time) How are you?

## JavaScript contd.

- Originally developed to offer dynamic control over the various elements within an HTML document.
- Now JavaScript is not only limited to client-side scripting.
  - You can develop both the client-side and server-side using JavaScript for highly dynamic and interactive web applications.



# JavaScript contd.

- Used for asynchronous communication.
  - Allows web pages to begin resembling stand-alone programs;
    - Because they don't have to be reloaded in their entirety to display new content.
  - An asynchronous call can pull in and update a single element on a web page
    - EX: Changing your photograph on a social networking site.

# Activity

- What are the alternatives for JavaScript?
  - Compare and contrast advantages and disadvantages of the alternatives and JavaScript.

# CSS

- CSS – Cascading Style Sheets
- Current version – CSS3
  - CSS now offers a level of dynamic interactivity previously supported only by JavaScript.
  - Example code:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <style>
5              p
6              {
7                  color:red;
8                  font-family:Helvetica;
9                  font-size: 20px;
10             }
11         </style>
12     </head>
13     <body>
14         <p>Hello world!</p>
15     </body>
16 </html>
```



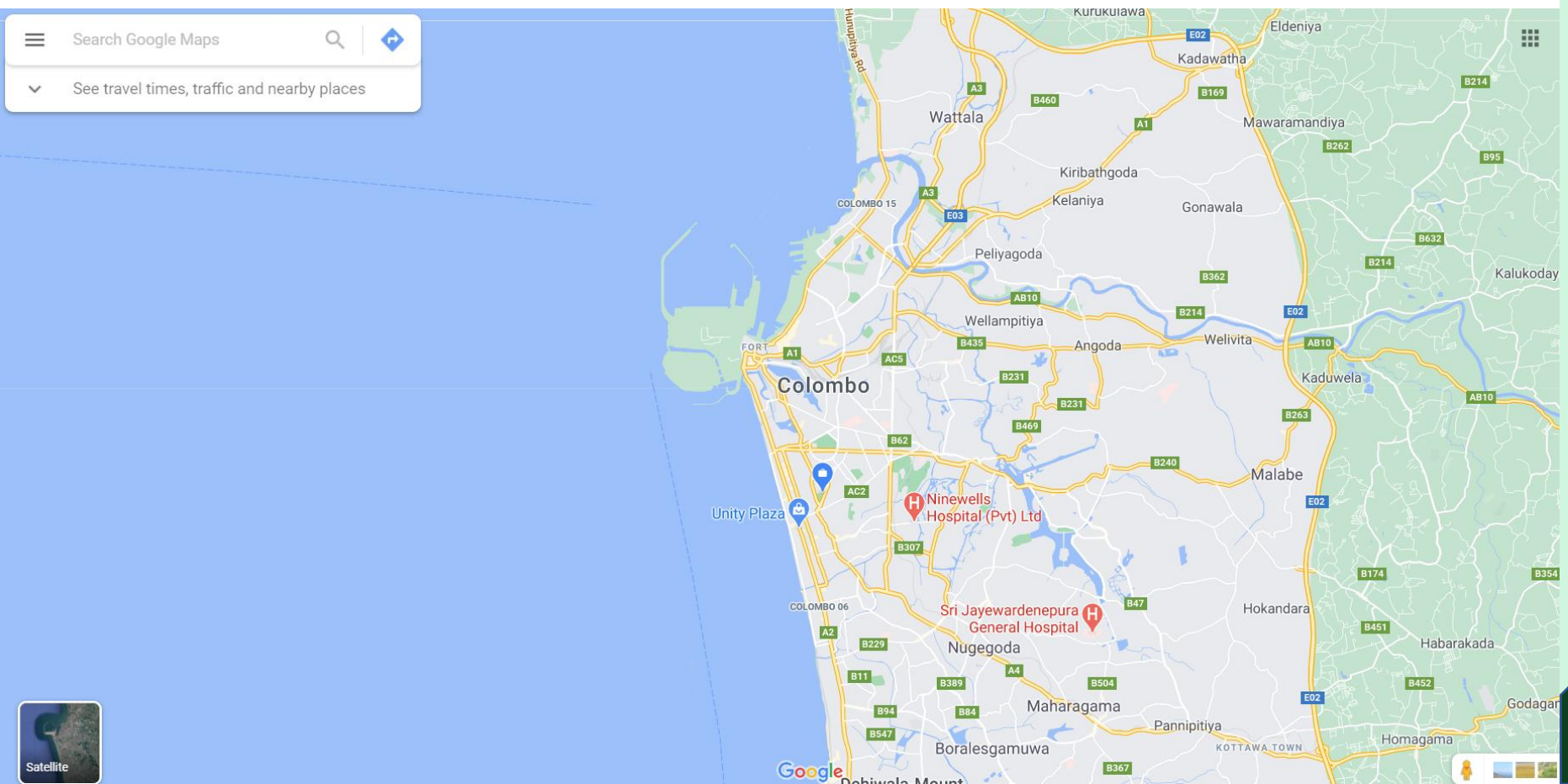
# PHP + MySQL + JavaScript + CSS ?

- PHP, MySQL, JavaScript, and CSS together?
  - PHP handles all the main work on the web server.
  - MySQL manages all the data.
  - The combination of CSS and JavaScript looks after web page presentation.
  - JavaScript can also interact with PHP code on the web server whenever it needs to update something.

# **Asynchronous JavaScript and XML (AJAX)**

# Asynchronous communication?

- Google maps: an example for asynchronous communication in web applications.
- New sections of a map are downloaded from the server when needed, without requiring a page refresh.



# Asynchronous communication?

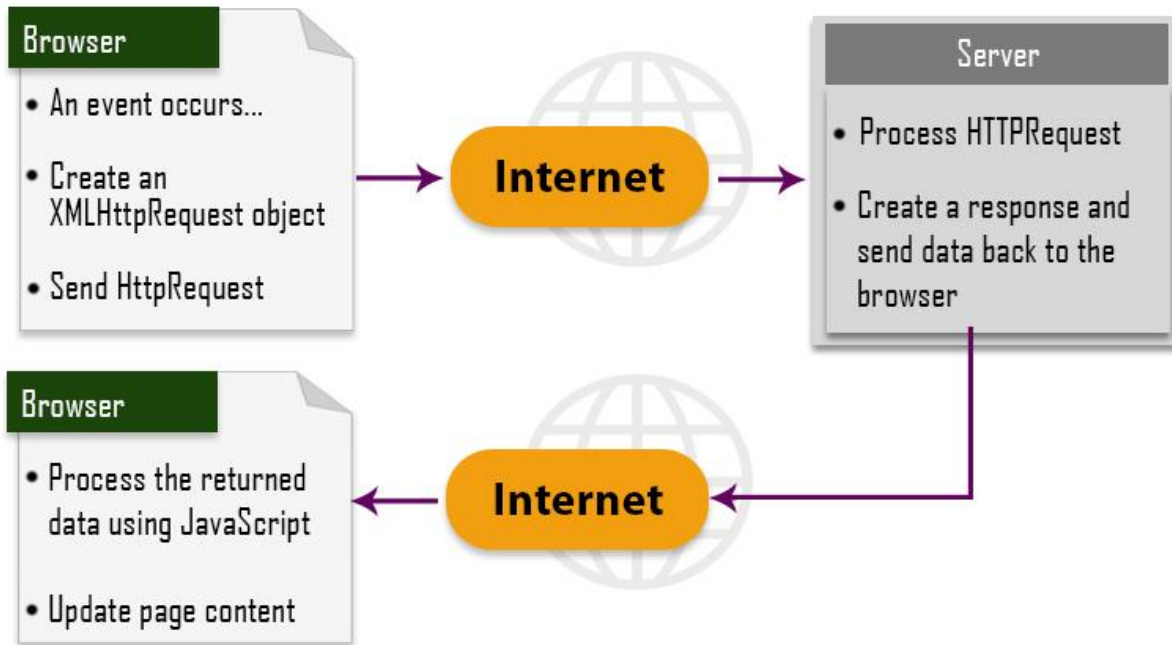
- Asynchronous communication: reduces the amount of data that must be sent back and forth.
- Makes web pages seamlessly dynamic,
  - More like self-contained applications rather than a set of static pages.
- Results in,
  - Improved user interfaces | Better responsiveness
- Examples;
  - Updating content of a web page without reloading the page.
  - Form auto-completion and in-line validation.
  - Social networks – reacts, ratings, voting, polls, etc..
  - Chat rooms and instant messaging

# AJAX introduction

- Asynchronous JavaScript and XML
- Misleading name?
  - AJAX applications may use XML to transport data
  - Equally common to transport data as plain text or JSON text.
- A set of methods that are built into JavaScript.
- Allow web pages to be updated asynchronously by transferring data between the browser and a server in the background.
- Update parts of a web page, without reloading the whole page.

# AJAX contd.

- AJAX uses a combination of:
  - A browser built-in XMLHttpRequest object to request data from a web server.
  - JavaScript and HTML DOM to display or use the data.



Source: [https://www.w3schools.com/xml/ajax\\_intro.asp](https://www.w3schools.com/xml/ajax_intro.asp)

# XMLHttpRequest object

- Syntax for creating an XMLHttpRequest object,

```
var xmlhttp = new XMLHttpRequest();
```

- For older versions of Internet Explorer,

```
IE 5: request = new ActiveXObject("Microsoft.XMLHTTP")
```

```
IE 6+: request = new ActiveXObject("Msxml2.XMLHTTP")
```

# XMLHttpRequest object methods

Method	Description
<code>abort()</code>	Aborts the current request
<code>getAllResponseHeaders()</code>	Returns all headers as a string
<code>getResponseHeader(<i>param</i>)</code>	Returns the value of <i>param</i> as a string
<code>open('method', 'url', 'async')</code>	Specifies the HTTP method to use (GET or POST), the target URL, and whether the request should be handled asynchronously ( <code>true</code> or <code>false</code> )
<code>send(<i>data</i>)</code>	Sends <i>data</i> to the target server using the specified HTTP method
<code>setRequestHeader('param', 'value')</code>	Sets a header with a parameter/value pair



# XMLHttpRequest object properties

Property	Description
onreadystatechange	Specifies an event-handling function to be called whenever the readyState property of an object changes.
readyState	An integer property that reports on the status of a request. It can have any of these values: 0 = Uninitialized, 1 = Loading, 2 = Loaded, 3 = Interactive, or 4 = Completed.
responseText	The data returned by the server in text format.
responseXML	The data returned by the server in XML format.
status	The HTTP status code returned by the server.
statusText	The HTTP status text returned by the server.

# Example: Asynchronous program

```
1  <!DOCTYPE html>
2  <html>
3      <body>
4          <h1>The XMLHttpRequest Object</h1>
5          <p id="info">This sentence will be changed</p>
6          <button type="button" onclick="asyncRequest()">Change Content</button>
7
8          <script>
9              function asyncRequest() {
10                 var xmlhttp = new XMLHttpRequest();
11                 xmlhttp.onreadystatechange = function() {
12                     if (this.readyState == 4 && this.status == 200) {
13                         document.getElementById("info").innerHTML = this.responseText;
14                     }
15                 };
16                 xmlhttp.open("GET", "infor.txt", true);
17                 xmlhttp.send();
18             }
19         </script>
20
21     </body>
22 </html>
```

## Example: Asynchronous program contd.

### The XMLHttpRequest Object

This sentence will be changed

Change Content

Click event



### The XMLHttpRequest Object

**Hello World!**

Change Content

## Example: Asynchronous program contd.

- Sending a request to a server.

```
xmlhttp.open("GET", "infor.txt", true);  
xmlhttp.send();
```

- open(method, url, async)
  - method: the type of request: GET or POST
  - url: the server (file) location
  - async: true (asynchronous) or false (synchronous)
- send()
  - Sends the request to the server

## Example: Asynchronous program contd.

- GET or POST?
  - GET is simpler and faster than POST.
  - Some browsers may cache GET requests, whereas POST requests will never be cached.
    - When cached, the browser would just redisplay what it got previously rather than displaying fresh input.
  - Solution: adds a random parameter to each request, ensuring that each URL requested is unique.

## Activity

Change the given sample code so that you can resolve the cached result issue in GET requests. Add a unique ID to the URL.

# Activity

Change the given sample code so that you are using a POST request instead of a GET request.

## Example: Asynchronous program contd.

- The url
  - Can be a file on the server.
  - .txt, .xml, .php etc...
- Asynchronous
  - True or False?
  - If set true, server requests would be sent asynchronously.

```
open(method, url, async)
```



## Example: Asynchronous program contd.

```
xmlhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
        document.getElementById("info").innerHTML = this.responseText;  
    }  
};
```

- *onreadystatechange* property
  - Defines a function to be called when the *readyState* property changes.
- *readyState* property
  - Holds the status of the XMLHttpRequest.
    - 0: request not initialized
    - 1: server connection established
    - 2: request received
    - 3: processing request
    - 4: request finished and response is ready

## Example: Asynchronous program contd.

```
xmlhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
        document.getElementById("info").innerHTML = this.responseText;  
    }  
};
```

- *status* property
  - Holds the status of the XMLHttpRequest object
  - 200: "OK" | 404: "Page not found" etc..
- The *onreadystatechange* function is called every time the *readyState* changes.
- When *readyState* is 4 and *status* is 200, the response is ready.

# Activity

Match the following HTTP status codes with relevant indications.

- 100
- 200
- 202
- 301
- 400
- 403
- 404
- 500
- 501
- 503

- Service Unavailable
- Bad request
- Forbidden
- Ok
- Not Implemented
- Continue
- Accepted
- Not Found
- Moved Permanently
- Internal Server Error

## Example: Asynchronous program contd.

```
document.getElementById("info").innerHTML = this.responseText;
```

- The element info is referenced via the getElementById method.

```
<p id="info">This sentence will be changed</p>
```

- Then its innerHTML property is assigned the value that was returned by the call.
- Element of the web page changes, while everything else remains the same.
- responseText - get the response data as a string.

# XML

- eXtensible Markup Language.
- Very similar to HTML.
- To store and transport data.
- Designed to be both human- and machine-readable.

```
test.xml
1  <CATALOG>
2  ∨ <GAMES>
3      <TITLE>Witcher 3 Wild Hunt</TITLE>
4      <GENRE>RPG</GENRE>
5      <COMPANY>CD Project Red</COMPANY>
6      <PRICE>30.99</PRICE>
7      <YEAR>2015</YEAR>
8  </GAMES>
9  </CATALOG>
```

# Activity

- Amali wants to use XML to store and transport some data on notes created through her system. She wants to store following details as elements in the XML file,
  - Time (24H)
  - Date (yyyy:mm:dd)
  - To
  - From
  - Heading
  - Body of the note
- Write the XML file content (note.xml)

# XML vs HTML

- XML: designed to transport data.
- HTML: designed to display data.
- XML does not have predefined tags unlike HTML.
  - Tags in XML are invented by the author.
- XML is extensible.
  - Most XML applications will work as expected even if new data is added or removed.

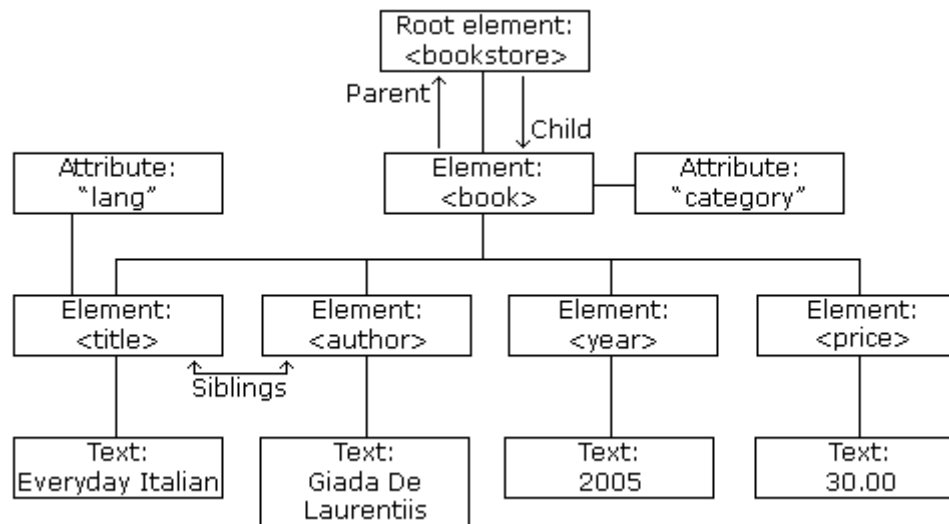
# Activity

- How can we use attributes in XML elements like in HTML?
- Amali wants to assign an ID as an attribute to her notes. Demonstrate how this can be done in the XML code you wrote in the last activity.



# XML DOM

- XML DOM defines a standard way for accessing and manipulating XML documents.
- It presents an XML document as a tree-structure.
- Example:



Source: [https://www.w3schools.com/xml/xml\\_dom.asp](https://www.w3schools.com/xml/xml_dom.asp)

# XML DOM contd.

- Sample XML file:

```
test.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CATALOG>
3      <GAME>
4          <TITLE>Witcher 3 Wild Hunt</TITLE>
5          <GENRE>RPG</GENRE>
6          <COMPANY>CD Project Red</COMPANY>
7          <PRICE>30.99</PRICE>
8          <YEAR>2015</YEAR>
9      </GAME>
10     <GAME>
11         <TITLE>Dragon Age Inquisition</TITLE>
12         <GENRE>RPG</GENRE>
13         <COMPANY>BioWare</COMPANY>
14         <PRICE>60.99</PRICE>
15         <YEAR>2014</YEAR>
16     </GAME>
17 </CATALOG>
```

## XML DOM contd.

- Following code retrieves the text value of the first <title> element in the sample XML document.

```
txt = xmlDoc.getElementsByTagName("TITLE")[0].childNodes[0].nodeValue;
```

```
1  <!DOCTYPE html>
2  <html>
3      <body>
4          <h1>The XMLHttpRequest Object</h1>
5          <p id="info">This sentence will be changed</p>
6          <button type="button" onclick="asyncRequest()">Change Content</button>
7
8          <script>
9              function asyncRequest() {
10                  var xmlhttp = new XMLHttpRequest();
11                  xmlhttp.onreadystatechange = function() {
12                      if (this.readyState == 4 && this.status == 200) {
13                          myFunction(this);
14                      }
15                  }
16
17                  xmlhttp.open("GET", "test.xml", true);
18                  xmlhttp.send();
19                  function myFunction(xml){
20                      var xmlDoc = xml.responseXML;
21                      var x = xmlDoc.getElementsByTagName("TITLE")[0].childNodes[0].nodeValue;
22                      var y = xmlDoc.getElementsByTagName("COMPANY")[0].childNodes[0].nodeValue;
23                      var z = xmlDoc.getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
24                      document.getElementById("info").innerHTML = x + " by " + y + " in " + z
25                  }
26              }
27          </script>
28
29      </body>
30  </html>
```

# The XMLHttpRequest Object

This sentence will be changed

Change Content



# The XMLHttpRequest Object

Witcher 3 Wild Hunt by CD Project Red in 2015

Change Content

# Activity

- Write the code to display the XML file you wrote in the previous assignment in a web page.
- Each element needs to be displayed.

🐘 index.php

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <style>
5      table,th,td {
6          border : 1px solid black;
7          border-collapse: collapse;
8      }
9      th,td {
10         padding: 5px;
11     }
12 </style>
13 </head>
14 <body>
15     <h1>The XMLHttpRequest Object</h1>
16     <table id="info"></table>
17     <button type="button" onclick="asyncRequest()">Change Content</button>
18

```

```

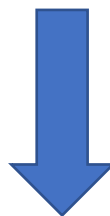
19     <script>
20     function asyncRequest() {
21         var xmlhttp = new XMLHttpRequest();
22         xmlhttp.onreadystatechange = function() {
23             if (this.readyState == 4 && this.status == 200) {
24                 myFunction(this);
25             }
26         }
27         xmlhttp.open("GET", "test.xml", true);
28         xmlhttp.send();
29         function myFunction(xml){
30             var i;
31             var xmlDoc = xml.responseXML;
32             var table="<tr><th>Title</th><th>Price</th></tr>";
33             var x = xmlDoc.getElementsByTagName("GAME");
34             for (i = 0; i < x.length; i++) {
35                 table += "<tr><td>" +
36                     x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
37                     "</td><td>" +
38                     x[i].getElementsByTagName("PRICE")[0].childNodes[0].nodeValue +
39                     "</td></tr>";
40             }
41             document.getElementById("info").innerHTML = table;
42         }
43     }
44     </script>
45 </body>
46 </html>

```



# The XMLHttpRequest Object

Change Content



# The XMLHttpRequest Object

Title	Price
Witcher 3 Wild Hunt	30.99
Dragon Age Inquisition	60.99

Change Content

# Activity

- Change the previous activity code to display multiple notes in a tabular format.

# JSON

- JavaScript **O**bject **N**otation.
- A syntax for storing and exchanging data.
- Syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

```
{ } test.json > ...
1  {
2      "catalog": "games",
3      "genre": "rpg",
4      "titles": [
5          { "name": "Witcher 3 Wild Hunt", "year": 2015 },
6          { "name": "Dragon Age Inquisition", "year": 2014 },
7          { "name": "Elder Scrolls V Skyrim", "year": 2011 }
8      ]
9  }
```

# Activity

- Amali wants to use JSON instead of XML in her application.
- Convert the note.xml file in previous activities to JSON.

# JSON vs XML

- Both are,
  - Human readable
  - Hierarchical
  - Can be parsed and used by lots of programming languages
  - Can be fetched with an XMLHttpRequest
- Different because,
  - JSON does not have end tags
  - Shorter
  - Quicker to read and write
  - Can use arrays
  - JSON is much easier to parse than XML

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4      </head>
5      <body>
6          <h1>The XMLHttpRequest Object</h1>
7          <table id="info"></table>
8          <button type="button" onclick="asyncRequest()">Change Content</button>
9
10         <script>
11             function asyncRequest() {
12                 var xmlhttp = new XMLHttpRequest();
13                 xmlhttp.onreadystatechange = function() {
14                     if (this.readyState == 4 && this.status == 200) {
15                         var myObj = JSON.parse(this.responseText);
16                         document.getElementById("info").innerHTML = myObj.catalog;
17                     }
18                 };
19                 xmlhttp.open("GET", "test.json", true);
20                 xmlhttp.send();
21             }
22         </script>
23     </body>
24 </html>
```

# The XMLHttpRequest Object

Change Content



# The XMLHttpRequest Object

games

Change Content

## Activity

- Display element in Amali's new JSON file on notes using AJAX.



## JSON contd.

- `JSON.parse()`
  - When receiving data from a web server, the data is always a string.
  - Parse the data with `JSON.parse()`, and the data becomes a JavaScript object.
- `JSON.stringify()`
  - When sending data to a web server, the data has to be a string.
  - Convert a JavaScript object into a string with `JSON.stringify()`

# AJAX and PHP

Start typing a name in the input field below:

First name:



Start typing a name in the input field below:

First name:

Hello Luke Skywalker. The time is: 21:01:01:07:50:34am

hello.php

```
1  <?php
2      echo date("y:m:d:h:i:sa");
3  ?>
```

index.php

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4      </head>
5      <body>
6          <p><b>Start typing a name in the input field below:</b></p>
7          <form>
8              First name: <input type="text" onkeyup="asyncRequest(this.value)">
9          </form>
10         <p id="info"></p>
11         <script>
12             function asyncRequest(name) {
13                 var xmlhttp = new XMLHttpRequest();
14                 xmlhttp.onreadystatechange = function() {
15                     if (this.readyState == 4 && this.status == 200 && name != "") {
16                         document.getElementById("info").innerHTML = "Hello " + name + ". The time is: " + this.responseText;
17                     }
18                 }
19                 xmlhttp.open("GET", "hello.php", true);
20                 xmlhttp.send();
21             }
22         </script>
23     </body>
24 </html>
```

# **Version controlling**

# Version controlling

- A system used to capture and record a visual history of changes in files within your project.
- Gives you the ability to go back and see
  - Who made the changes
  - What they changed - both files and the changed contents
  - When they made the change
  - Why it was changed – through reading the commit message
- Provides you with a mechanism to segregate changes in your code, called branches.

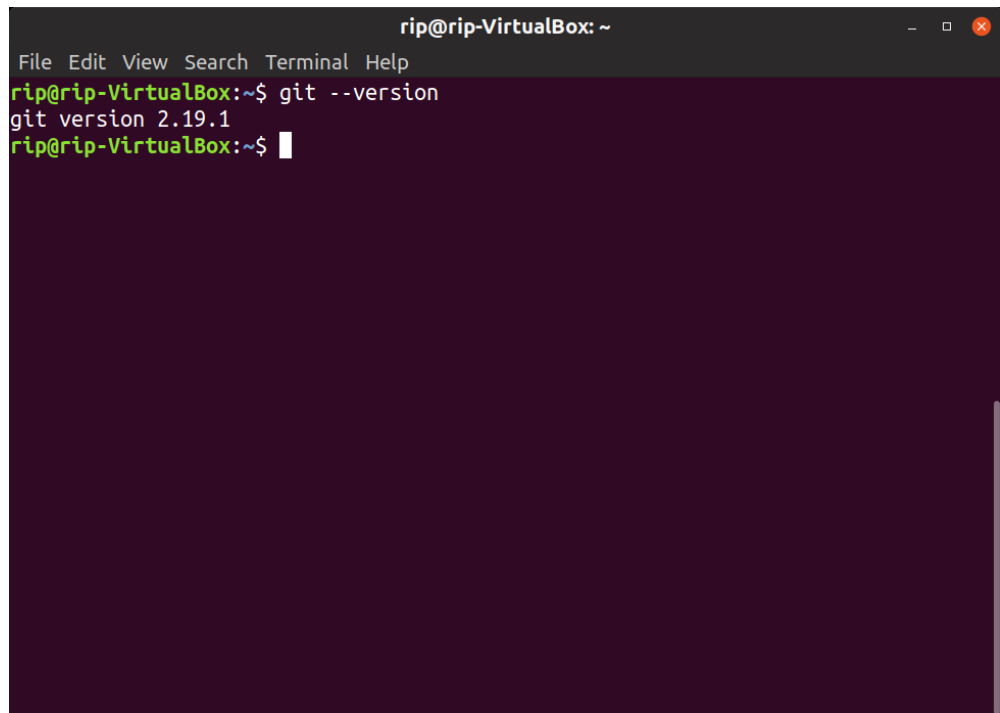
# Git

- A distributed version control system (DVCS).
- Allows you to distribute many copies (mirrors) of your repository to other members of your team.
- Able to track changes.
- Each person with a clone of the repository has an entire working copy of the system at the time of the clone.
- Simple, fast, and fully distributable.

# Setting up Git

- Download the appropriate binary install for your OS from: <http://git-scm.com>
- On Ubuntu,
- Depending on the OS, there may be multiple ways to install Git.

```
$ sudo dnf install git-all
```

A terminal window titled 'rip@rip-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'git --version' being executed, resulting in the output 'git version 2.19.1'. The prompt 'rip@rip-VirtualBox:~\$' is visible at the end of the line.

```
rip@rip-VirtualBox: ~  
File Edit View Search Terminal Help  
rip@rip-VirtualBox:~$ git --version  
git version 2.19.1  
rip@rip-VirtualBox:~$
```

# Activity

- Check the version of your Git installation.



# Configuring Git

- Setting up identity

- The first thing you should do when you install Git is to set your user name and email address.
- This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating.

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johnndoe@example.com
```

- To verify your configurations,

```
$ git config user.name
```

```
$ git config user.email
```

# Initializing a repository

- To create your first Git repository , you will simply use the *git init* command.
- This will initialize an empty Git repository in your source code directory.

```
C:\git>git init  
Initialized empty Git repository in C:/git/.git/
```

# Initial commit

- Let us add a very basic README file to the empty repository and then perform our initial commit.

```
$ echo "This is our README." > README.md
```

- You can look at the current status of the repository using,

```
$ git status
```

```
C:\git>echo "This is our README." > README.md

C:\git>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

nothing added to commit but untracked files present (use "git add" to track)
C:\git>
```

## Initial commit contd.

- The readme file is still an untracked file, meaning it isn't yet added to the repository or being tracked for changes by Git.
- To add the file to the repository,  
\$ git add README.md
- Check the repository status again.

```
C:\git>git add README.md
```

```
C:\git>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md

C:\git>
```

## Initial commit contd.

- Lastly, we'll commit our change, and our new README will now be in our repository and be tracked for changes going forward.

```
C:\git>git commit -m "Initial commit. Added our README"
[master (root-commit) 984ccdb] Initial commit. Added our README
1 file changed, 1 insertion(+)
create mode 100644 README.md
```

```
C:\git>git status
On branch master
nothing to commit, working tree clean
```

# Staging changes

- Let us change the README first and check the git status.

```
C:\git>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

- It shows that our README was modified, but that it's not staged for commit yet. We do this by using the git add command.

```
C:\git>git add README.md

C:\git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md

C:\git>git commit -m "Updated README"
[master 42a11e3] Updated README
1 file changed, 1 insertion(+), 1 deletion(-)
```

# Viewing history

- We have already made a few changes.
- Let us check the change history using

\$ git log

```
$ git log
commit ca476b6c41721cb74181085fd24a40e48ed991ab
Author: Chad Russell <chad@intuitivereason.com>
Date:   Tue Mar 31 12:25:36 2015 -0400

    Updated README

commit dc56de647ea8edb80037a2fc5e522eec32eca626
Author: Chad Russell <chad@intuitivereason.com>
Date:   Tue Mar 31 10:52:23 2015 -0400

    Initial commit. Added our README
```

# Ignoring specific files

- There will often be a number of files and directories within your project that you do not want Git to track.
  - Files that are private and not useful to other collaborators. Ex:
    - Personal IDE config files, such as .idea/workspace.xml
    - Compiled code, such as .o, .pyc, and .class files
    - Security and API keys
  - Example .gitignore file:

```
# Binaries for programs and plugins
*.exe
*.exe~
*.dll
*.so
*.dylib
```



# Activity

- What are Git ignore patterns?
- Name 5 patterns and explain them using examples.

# Removing files

- Completely removing a file from both the repository and the local working copy: *git rm*
- Delete the file from local copy using operating system or IDE: it will show up as a deleted file that needs to be committed.

```
C:\git>git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        deleteme.txt
        deleteme2.txt

nothing added to commit but untracked files present (use "git add" to track)
C:\git>git add deleteme.txt
C:\git>git add deleteme2.txt
C:\git>git commit -m "Adding files that we plan on deleting"
[master 6f02339] Adding files that we plan on deleting
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 deleteme.txt
 create mode 100644 deleteme2.txt
```

# Removing files

- First use *git rm* to delete a file.

```
C:\git>git rm deleteme.txt
rm 'deleteme.txt'

C:\git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    deleteme.txt

C:\git>git commit -m "Removed our temporary file"
[master 5da05de] Removed our temporary file
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 deleteme.txt

C:\git>git status
On branch master
nothing to commit, working tree clean
```

# Removing files

- First use *git rm* to delete a file. Use *git rm* and then commit change.

```
C:\git>git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    deleteme2.txt

no changes added to commit (use "git add" and/or "git commit -a")

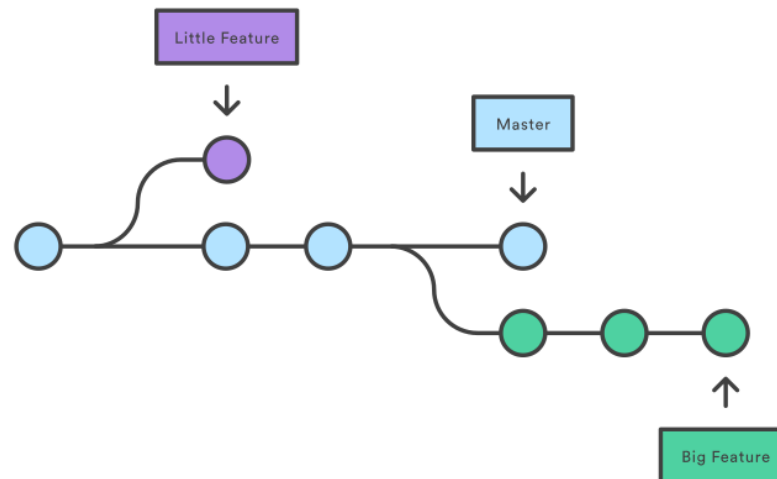
C:\git>git rm deleteme2.txt
rm 'deleteme2.txt'

C:\git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    deleteme2.txt

C:\git>git commit -m "Removed our temporary file"
[master 37db5a9] Removed our temporary file
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 deleteme2.txt
```

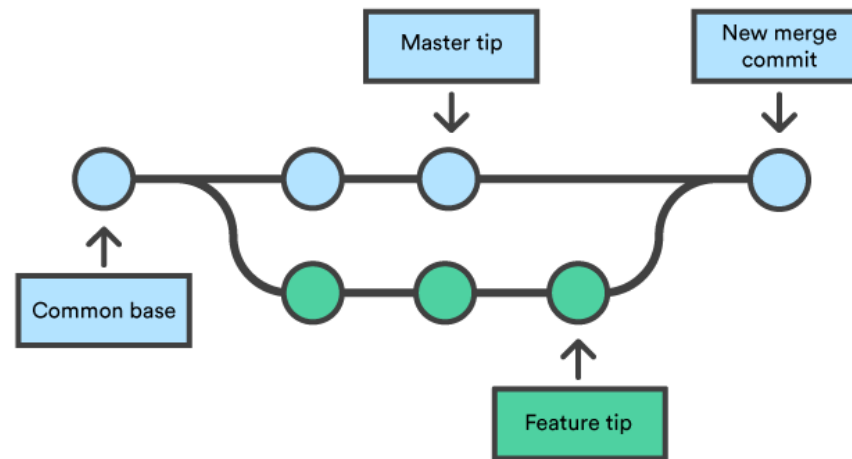
# Branching & merging

- Allows you to separate various segments of changes to your code into sub-repositories.
- Branching
  - When you want to add a new feature or fix a bug, you may spawn a new branch to encapsulate your changes.
  - Makes it harder for unstable code to get merged into the main codebase.



# Branching & merging contd.

- Merging
  - Lets you take the independent lines of development created by git branch and integrate them into a single branch.
  - Ex: we have a new branch feature that is based off the master branch. We now want to merge this feature branch into master.



## Branching & merging contd.

- Create a new branch and then switch to that branch.

```
$ git branch branch-example
```

```
$ git checkout branch-example
```

```
C:\git>git branch branch-example  
C:\git>git checkout branch-example  
Switched to branch 'branch-example'
```

- Add and commit new files to the branch.

```
C:\git>touch test.php  
Touching test.php  
  
C:\git>git add test.php  
  
C:\git>git commit -m "adding new file to the example branch"  
[branch-example 3898d29] adding new file to the example branch  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 test.php
```

## Branching & merging contd.

- Switch to the master branch and check the directory.
- The newly added test.php file is missing in the master branch.

```
Directory of C:\git
01/01/2021  06:16 PM    <DIR>        .
01/01/2021  06:16 PM    <DIR>        ..
01/01/2021  03:32 PM                35 README.md
01/01/2021  06:16 PM                0 test.php
                2 File(s)                35 bytes
                2 Dir(s)  65,362,948,096 bytes free
```

```
C:\git>git checkout master
Switched to branch 'master'

C:\git>dir
Volume in drive C has no label.
Volume Serial Number is 3878-1B64

Directory of C:\git

01/01/2021  06:19 PM    <DIR>        .
01/01/2021  06:19 PM    <DIR>        ..
01/01/2021  03:32 PM                35 README.md
                1 File(s)                35 bytes
                2 Dir(s)  65,362,305,024 bytes free
```



## Branching & merging contd.

- When performing a merge, Git will compare changes in both branches and will attempt to automatically merge the changes together.
- Collision of changes?
  - Same lines of code were changed in both branches.
  - Need manual intervention to resolve conflicts.

```
C:\git>git merge branch-example
Updating 37db5a9..3898d29
Fast-forward
 test.php | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.php
```

- To delete a branch: 

```
C:\git>git branch -d branch-example
Deleted branch branch-example (was 3898d29).
```

## Other features in Git

- Stashing Changes
- Tagging
  - Lightweight tags
  - Annotated tags
- Undoing changes
  - Amend
  - Un-stage
  - File Reset
  - Soft Reset
  - Mixed Reset
  - Hard reset

# Activity

- What is *Stashing Changes* in Git?
- Explain how \$ git stash can be used.

# Activity

- What is the purpose of *Tagging* in Git?
- Explain and demonstrate the difference between lightweight tags and annotated tags.

# Activity

- Provide examples when we need to undo changes in Git.
- Demonstrate and explain different ways of undoing changes in Git.

# Version control in cloud

- Having a remote-hosted repository is common practice when using Git other than for just your own personal projects.
- Popular services
  - Bitbucket (<https://bitbucket.org/>)
  - GitHub (<https://github.com/>)
- Both free and paid plans.

# **PHP Coding Standards**

# Introduction to coding standards

- A set definitions on how to structure your code in any given project.
- Applies to everything from,
  - naming conventions
  - Spaces
  - Variable names
  - Opening and closing bracket placementsEtc..
- PHP-FIG
  - PHP Framework Interoperability Group
  - created a standards body for PHP frameworks.



# Basic coding standards - Files

- PHP Tags
  - PHP code must use `<?php` tags or the short echo tag in `<?=</code> format.`
  - No other tag is acceptable, even if you have short tags enabled in your PHP configuration.
- Character Encoding PHP
  - PHP code must use only UTF-8 without the byte-order mark (BOM).

# Basic coding standards - Files

- Side Effects
  - PHP file should either declare new symbols (classes, functions, constants, etc.) or execute logic with side effects, but never both.
  - Side effects - logic executed without directly relating to declaring a class, functions or methods, constants, etc.
  - Unlike the following example, a file shouldn't both declare a function AND execute that function.

```
<?php

// Execution of code
myFunction();

// Declaration of function
function myFunction() {
    // Do something here
}
```

# Basic coding standards – Name spaces and class names

- A class is always in a file by itself (there are not multiple classes declared in a single file), and it includes at least a namespace of one level.
- Class names must be declared using StudlyCaps.

```
<?php

namespace Apress\PhpDevTools;

class MyClass
{
    // methods here
}
```

# Basic coding standards – Class constants, Properties, and Methods

- Constants
  - Class constants must be declared in all uppercase using underscores as separators.
- Properties
  - Property names: \$StudlyCaps , \$camelCase , or \$under\_score
  - Can mix them if they are outside of the scope of each other.
  - Be consistent within that given scope.
  - Better to stick to one throughout all of your code.
    - More uniformity and readability.

# Basic coding standards – Class constants, Properties, and Methods

- Methods
  - Must be declared using camelCase()

```
<?php

namespace Apress\PhpDevTools;

class MyClass
{
    const VERSION = '1.0';

    public $myProperty;

    public function myMethod()
    {
        $this->myProperty = true;
    }
}
```

# Coding style – General

- Files
  - All PHP files must use the Unix linefeed line ending, must end with a single blank line, and must omit the close `?>` tag if the file only contains PHP.
- Lines
  - There must not be a hard limit on the length of a line.
  - There must be a soft limit of 120 characters.
  - Lines should not be longer than 80 characters, and should be split into multiple lines if they go over 80 characters.
  - There must not be trailing whitespace at the end of non-blank lines.
  - Blank lines may be added to improve readability and to indicate related blocks of code.
  - You can only have one statement per line.

# Coding style – General

- Indentation
  - You must use four spaces and never use tabs.
  - (Most IDEs can map spaces to tab key.)
- Keywords and true, false, and null
  - All keywords must be in lowercase as must the constants true , false , and null.

# Coding style – Namespace and Use declarations

- There must be one blank line after the namespace is declared.
- Any use declarations must go after the namespace declaration.
- Only one use keyword per declaration.
- One blank line after the use block.



# Coding style – Classes, Properties, and Methods

- Classes
  - The extends and implements keywords must be declared on the same line as the class name.
  - The opening brace for the class must go on its own line, and the closing brace must appear on the next line after the body of your class.
  - Lists of implements may be split across multiple lines, where each subsequent line is indented once.
    - When doing this, the first item in the list must appear on the next line, and there must only be one interface per line.

# Coding style – Classes, Properties, and Methods

- Properties
  - Visibility (public, private, or protected) must be declared on all properties in your classes.
  - The var keyword must not be used to declare a property.
  - There must not be more than one property declared per statement.
  - Property names should not be prefixed with a single underscore to indicate protected or private visibility.

# Coding style – Classes, Properties, and Methods

- Method arguments
  - In your method argument list, there must not be a space before each comma, but there must be one space after each comma.
  - Method arguments with default values must go at the end of the argument list.
  - Can split method argument lists across multiple lines, where each subsequent line is indented once.
    - When using this approach, the first item in the list must be on the next line, and there must be only one argument per line
  - If the split argument list is used, the closing parenthesis and the opening brace must be placed together on their own line with one space between them.

# Coding style – Classes, Properties, and Methods

- Abstract, Final, and Static
  - When present, the abstract and final declarations must precede the visibility declaration.
  - When present, the static declaration must come after the visibility declaration.
- Method and Function Calls
  - When you make a method or function call, there must not be a space between the method or function name and the opening parenthesis.
  - There must not be a space after the opening parenthesis or before the closing parenthesis.
  - In the argument list, there must not be a space before each comma, but there must be one space after each comma.
  - The argument list may also be split across multiple lines, where each subsequent line is indented once.

## Coding style – Control structures

- There must be one space after the control structure keyword.
- There must not be a space after the opening parenthesis or before the closing parenthesis.
- There must be one space between the closing parenthesis and the opening brace, and the closing brace must be on the next line after the body.
- The structure body must be indented once.
- The body of each structure must be enclosed by braces.

# Coding style – Control structures

- if, elseif, else
  - A control structure should place else and elseif on the same line as the closing brace from the previous body. Also,
  - Always use elseif instead of else if so the keywords are all single words.

```
<?php  
  
if ($a === true) {  
} elseif ($b === true {  
} else {  
}
```

# Coding style – Control structures

- switch, case
  - The case statement must be indented once from the switch keyword, and the break keyword or other terminating keyword (return, exit, die , etc.) must be indented at the same level as the case body.
  - There must be a comment such as // no break when fall-through is intentional in a non-empty case body.

```
<?php
switch ($a) {
    case 1:
        echo "Here we are.";
        break;
    case 2:
        echo "This is a fall through";
        // no break
    case 3:
        echo "Using a different terminating keyword";
        return 1;
    default:
        // our default case
        break;
}
```

?>

# Coding style – Control structures

- while, do while
  - These structures place the braces and spaces similarly to those in the if and switch structures

```
<?php  
  
while ($a < 10) {  
    // do something  
}  
  
do {  
    // something  
} while ($a < 10);
```



# Coding style – Control structures

- for
  - Standards comply to the following examples.

```
<?php  
  
for ($i = 0; $i < 10; $i++) {  
    // do something  
}  
  
for ($j=0; $j<10; $i++) {  
    // do something  
}
```

# Coding style – Control structures

- foreach
  - Unlike in the for statement, the space is required if you are separating the key and value pairs using the => assignment.

```
<?php

foreach ($array as $a) {
    // do something
}

foreach ($array as $key => $value) {
    // do something
}
```

# Coding style – Control structures

- try, catch (and finally)
  - Standards comply to the following example.

```
<?php

try {
    // try something
} catch (ExceptionType $e) {
    // catch exception
} finally {
    // added a finally block
}
```

# PHP CodeSniffer

- To check coding standards.
  - A code validator that everyone can easily run.
  - Could be incorporated into an automated process.
  - Ensure all code is compliant with PHP-FIG(PSR-1 and PSR-2) standards, or even with another coding standard that you choose.
- PHP CodeSniffer is a set of two PHP scripts
  - phpcs: tokenizes PHP files (as well as JavaScript and CSS) to detect violations of a defined coding standard.
  - phpcbf: used to automatically correct coding standard violations.
- Can use it either via command line or directly in some IDEs, such as PHP Storm or NetBeans.

# PHP CodeSniffer

```
$ phpcs --standard=PSR1,PSR2 invalid.php
```

```
FILE: /Apress/source/invalid.php
```

```
-----  
FOUND 10 ERRORS AFFECTING 5 LINES  
-----
```

```
 3 | ERROR | [ ] Each class must be in a namespace of at least one  
   |       |     level (a top-level vendor name)  
 3 | ERROR | [x] Opening brace of a class must be on the line after  
   |       |     the definition  
 4 | ERROR | [ ] Class constants must be uppercase; expected VERSION  
   |       |     but found version  
 6 | ERROR | [ ] The var keyword must not be used to declare a  
   |       |     property  
 6 | ERROR | [ ] Visibility must be declared on property "$Property"  
 8 | ERROR | [ ] Method name "ExampleClass::ExampleMethod" is not in  
   |       |     camel caps format  
 8 | ERROR | [ ] Expected "function abc(...)"; found "function abc  
   |       |     (...)"
```

An example for errors that were detected when validating against the PSR-1 and PSR-2 standards using PHP CodeSniffer

# phpDocumentor

- Not all coding standards provide rules regarding code comments.
- Capable of automatically generating code documentation for the entire project and provide an easy reference for all developers to follow.
- Mainly used for generating real documentation from the source in a variety of different formats .

# **Frameworks for Web Development**

# Why use frameworks?

- A defined structure across all your applications.
- Community support.
- A pre-developed set of functionality that you don't have to reinvent with each application.
- Modules, libraries, and plugins available to add additional functionality.
- Better testability.
- Pre-established use of design patterns in your application.
- Reusable and maintainable code.



# Example PHP frameworks?

- Laravel 5
- Symfony 2
- Zend Framework 2
- CodeIgniter

**END**