DATABASE SYSTEMS

The very basics of Web programming

Database Systems Course

TEL AVIV UNIVERSITY

BEFORE WE START…

- **This lecture is an overview of the very complicated world of web programming.**
- **If you are a (very) experienced web developer - take a 2.5hr break and come back for the last 20 minutes of the third period.**
- **The goal of this talk is to introduce you to this world and give you some tools to explore it by yourself.**
- **You can do this,**
- **even if you have no experience at all.**
- **You may hate it now, and send me thank-you emails after graduation. ^_^**

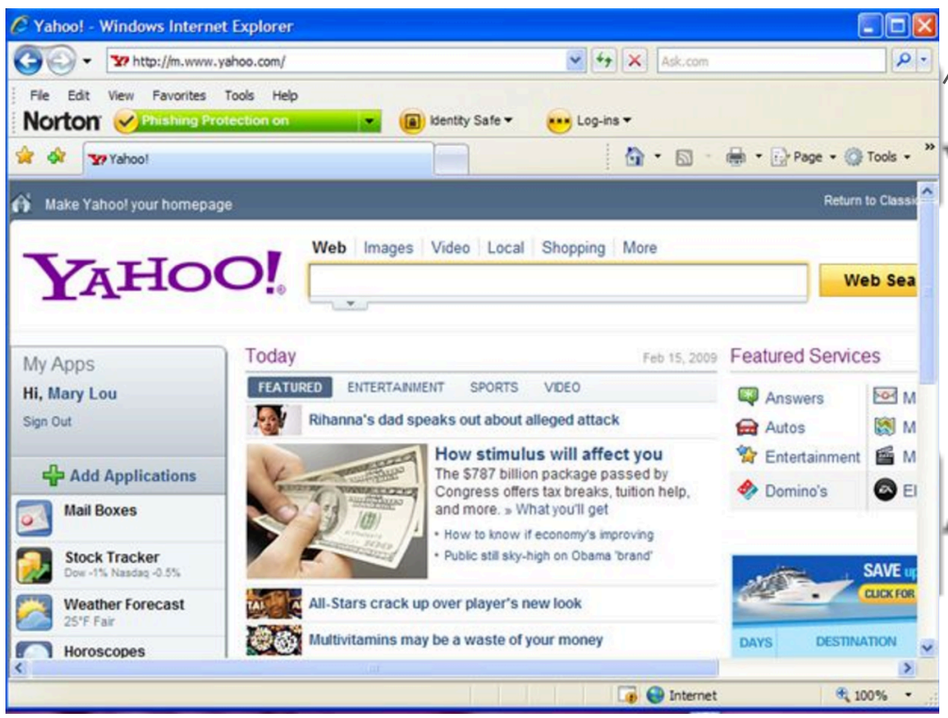
CLIENT VS SERVER

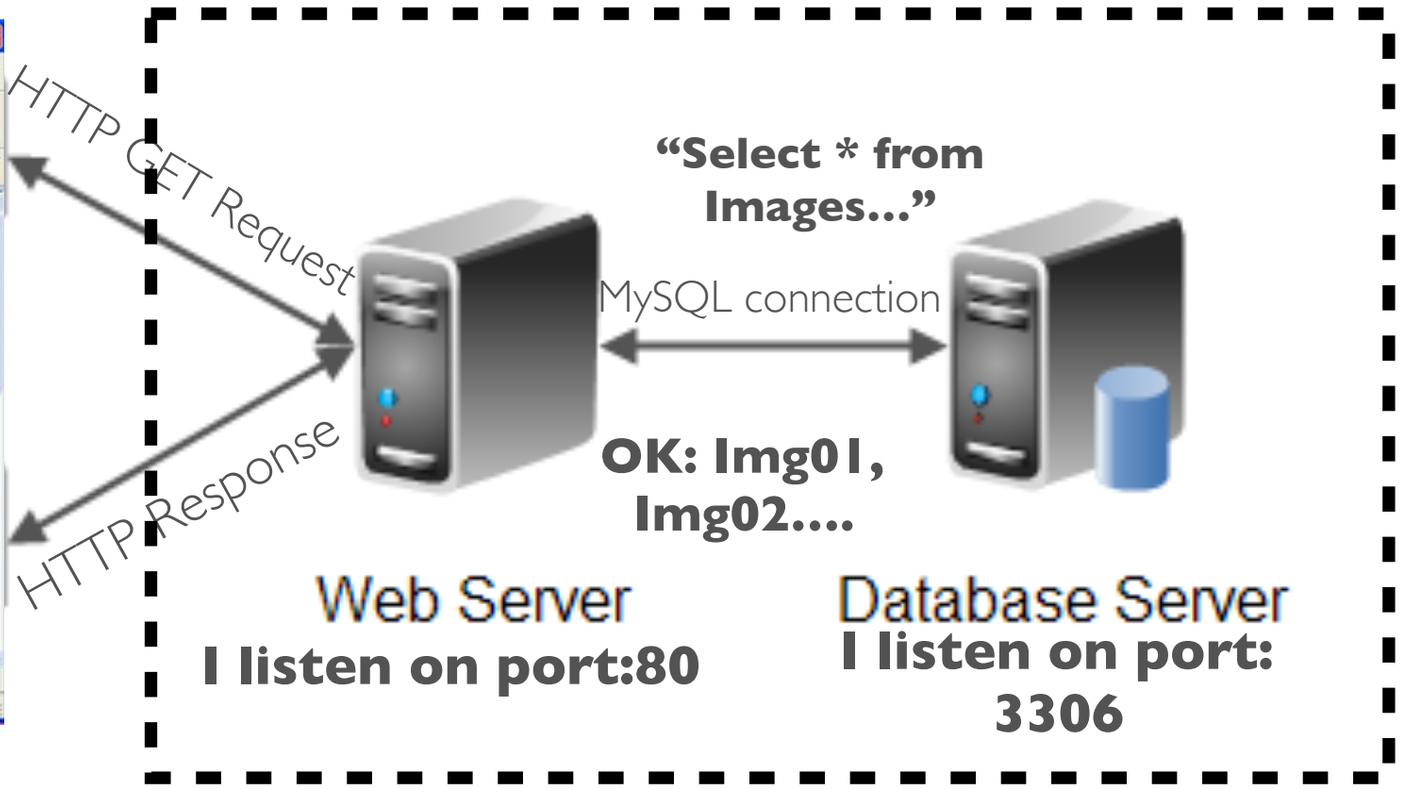
The client:

1. Send and receive HTTP
2. Read and parse HTML (or Json or XML if it is API)
3. I interact with the user

The server(s):

1. Listen and Wait for HTTP requests
2. Process the request
3. Send a response
4. I'm stateless





AGENDA FOR TODAY

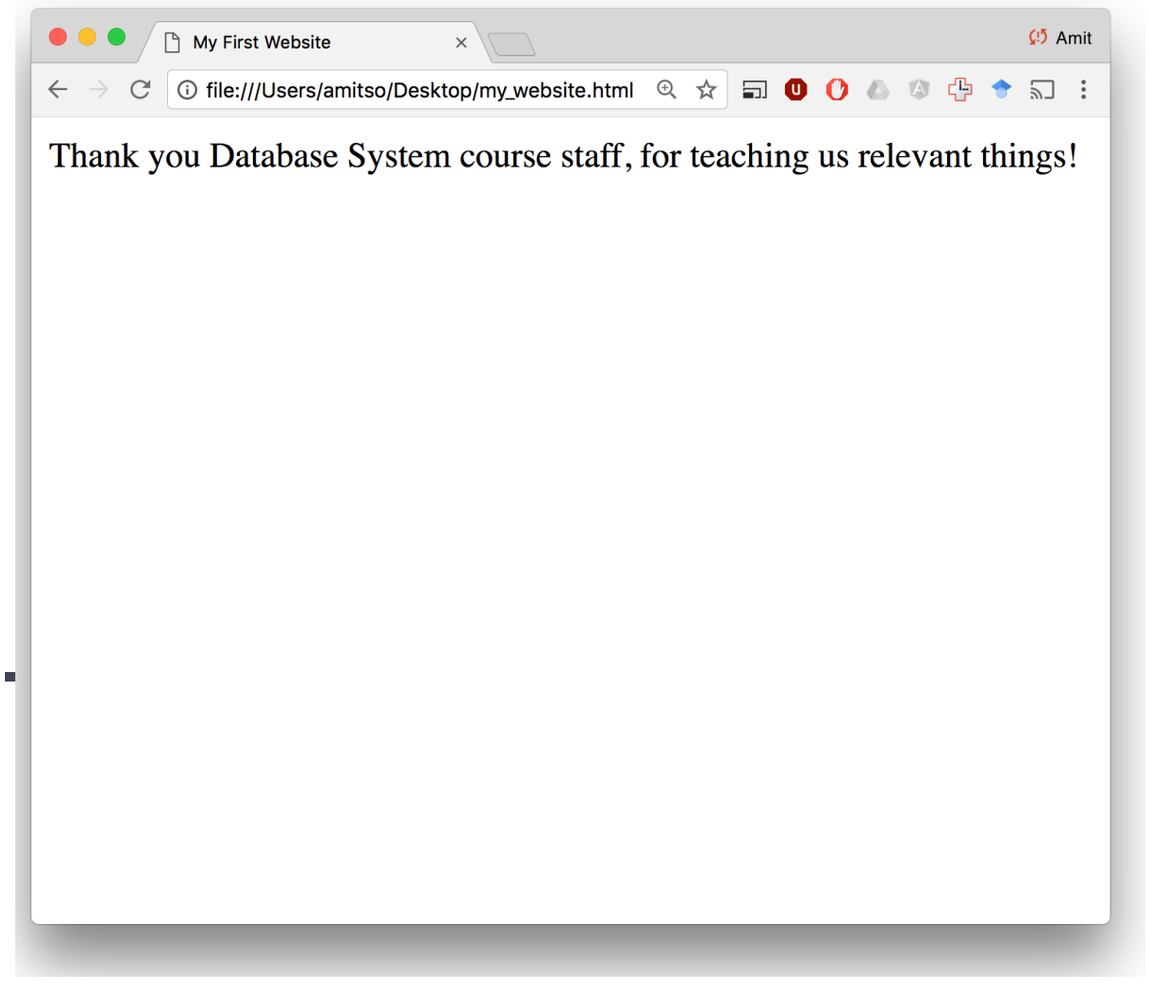
- **Part #1: Client side programming:**
  - HTML, CSS Javascript
  - Responsive design
  - Front-end frameworks
- **Part #2: Server side programming**
  - Python (Flask): Install, hello world, requests, cookies, sessions.
- Using the university web servers.
- **Part #3: Using the university servers (python/ php)**

YOUR FIRST WEBSITE!

- **Open a text editor**
- **Type this:**

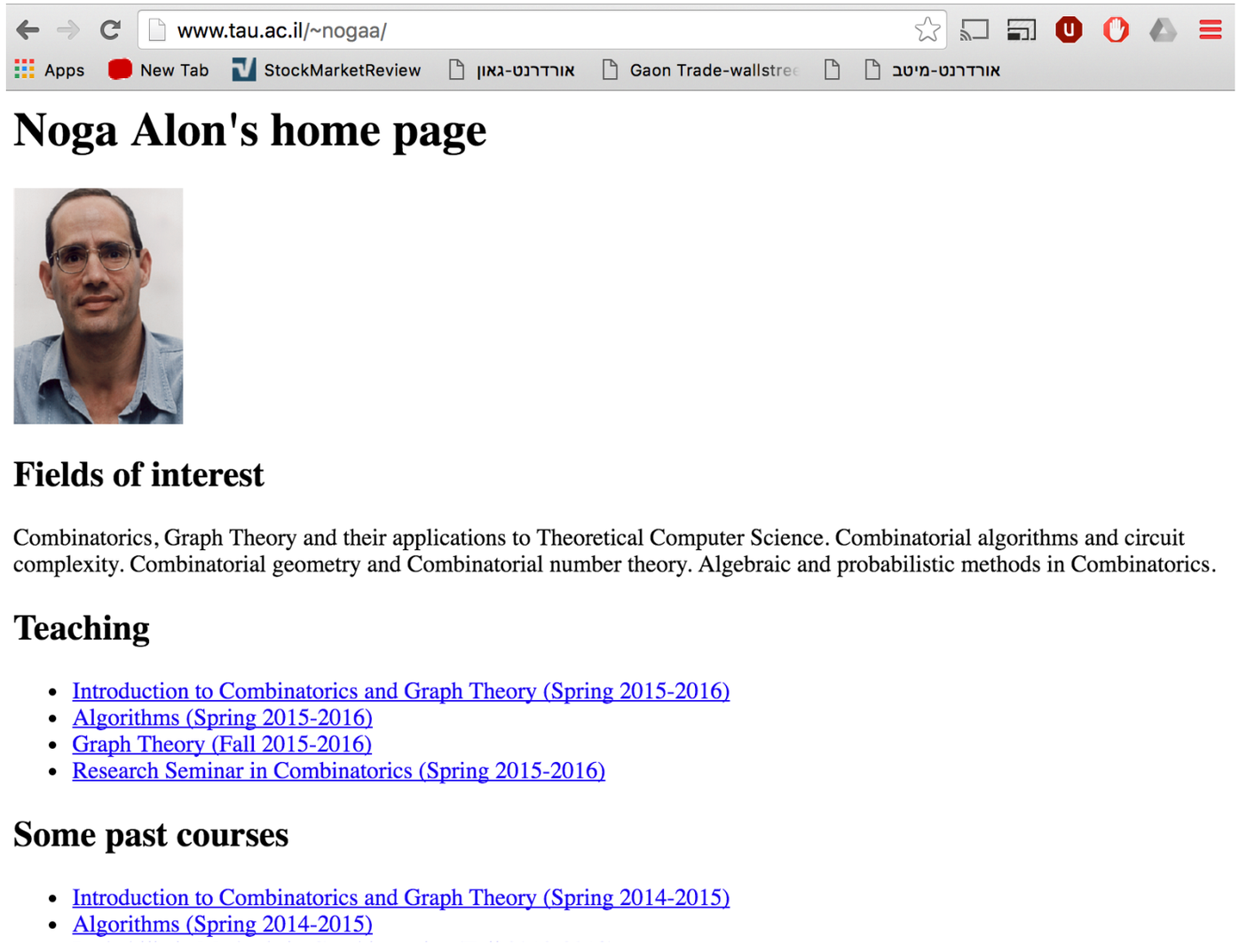
```
1 <html>
2   <title> My First Website </title>
3   <body>
4   Thank you Database System course staff,
5   for teaching us relevant things!
6   </body>
7 </html>
```

- **Save the file as my_website.html**
- **Open in your favorite browser**
- **Voila !**



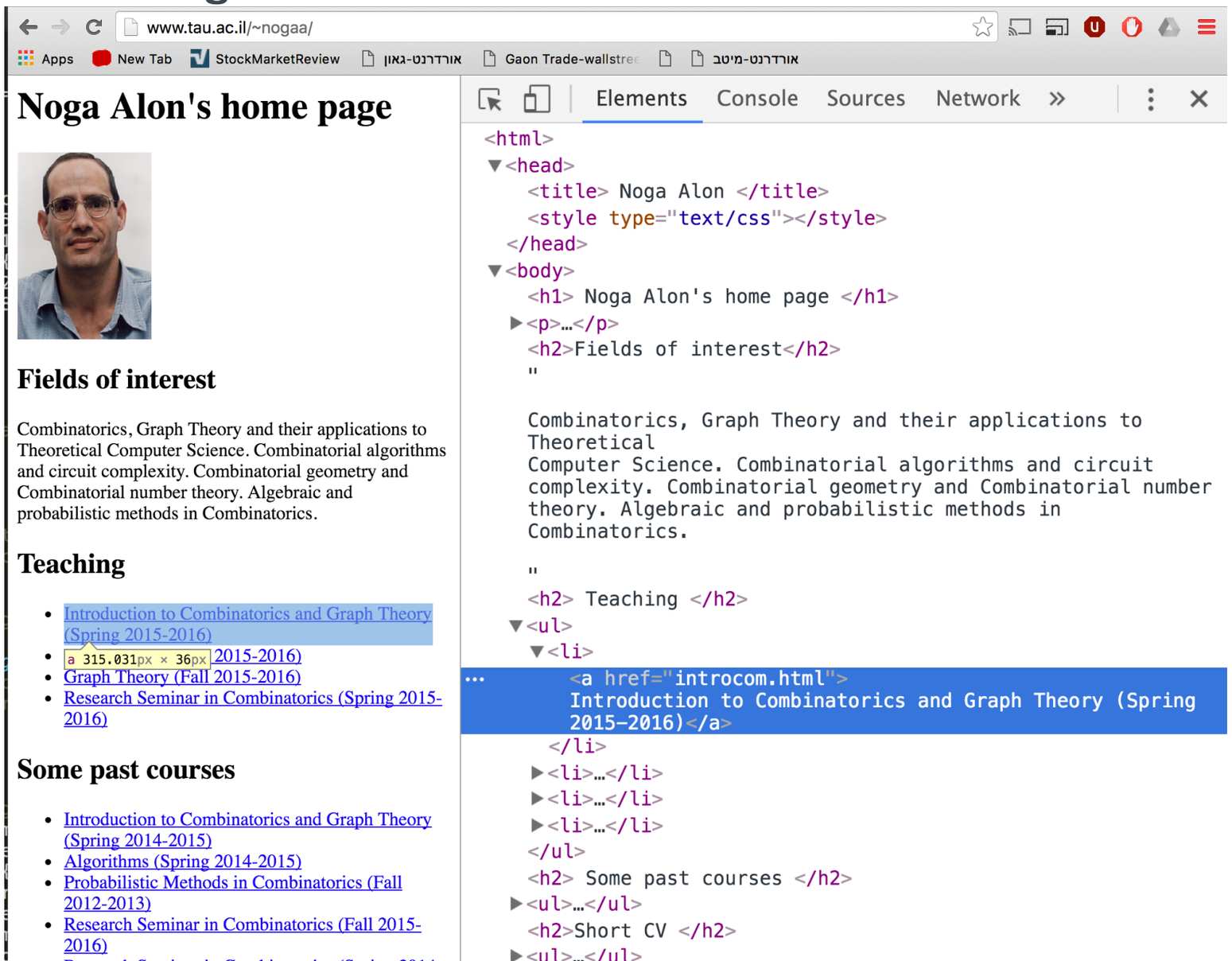
A STATIC WEB PAGE

- **Content is identical, regardless (i.e. non-interactive)**
- **To perform changes in content, the programmer has to change the HTML file.**
- **For example:**



A STATIC WEB PAGE

- **To view the HTML source code, we can right click and select “view source”**
- **Or use the browser’s *developer tools*. e.g.**



A STATIC WEB PAGE

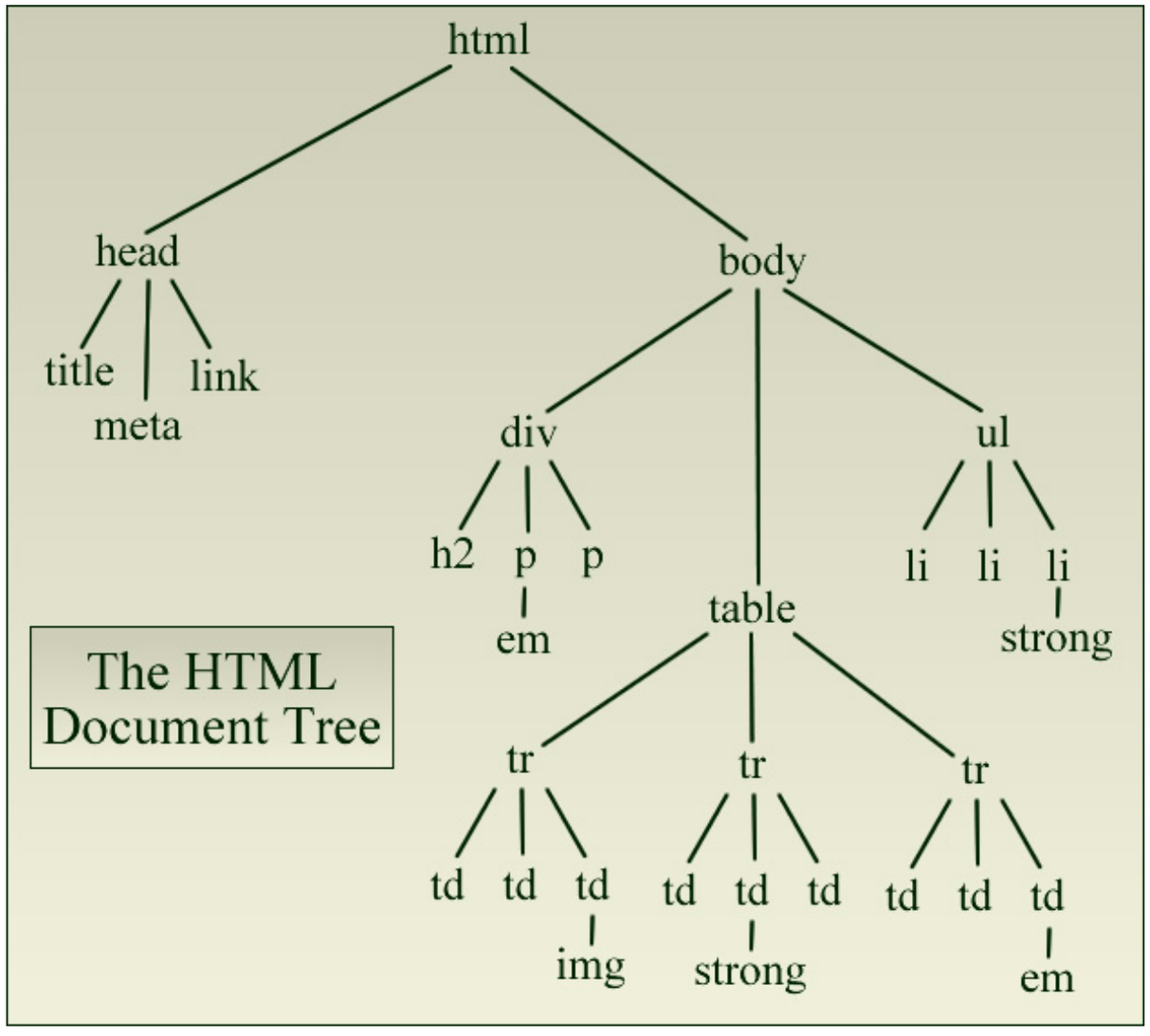
- **HTML web page is a document, orgnizied in a tree structure, according to the Document Object Model (DOM).**
- **The most important nodes:**

<html> the root of every web page

<head> containing meta-data and external sources

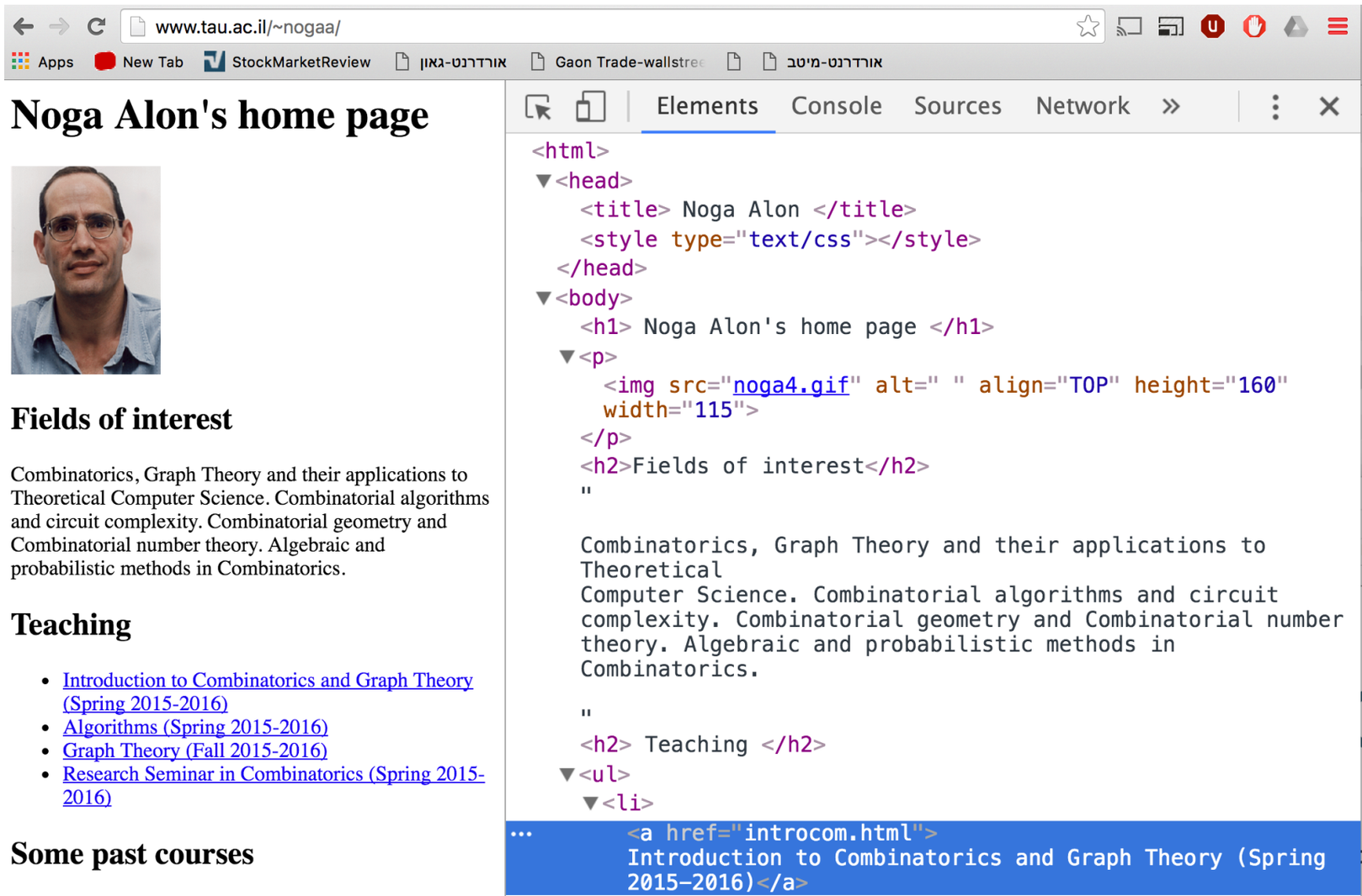
<body> holds the content of the webpage

** ** is the basic content container.



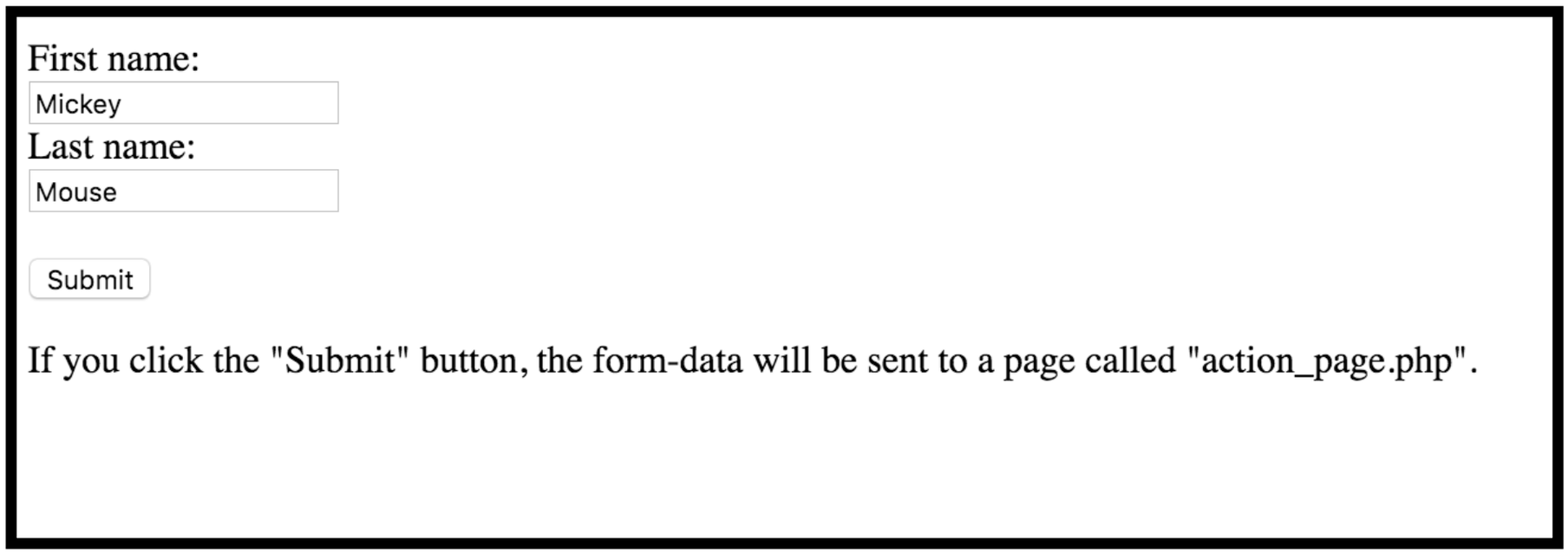
A STATIC WEB PAGE

- **Each node is an *element***
- **Each element beings and ends with a tag e.g. : <title> Noga Alon </title>**
- **Each *element* has *a set of attributes***
  - **structure:** *attr = val*
  - *<img src="noga4.gif" alt=" ">*



BASIC "INTERACTIONS"

- **Web forms:**
  - Used to collect **input** from the user and **submit** it to the server
  - The values are sent to the **web server** via **HTTP GET/POST** requests:
    - **GET:** most web requests you will encounter, parameters are passed in the URL
    - **POST:** used to send files, large size parameters, and sensitive parameters (passwords)

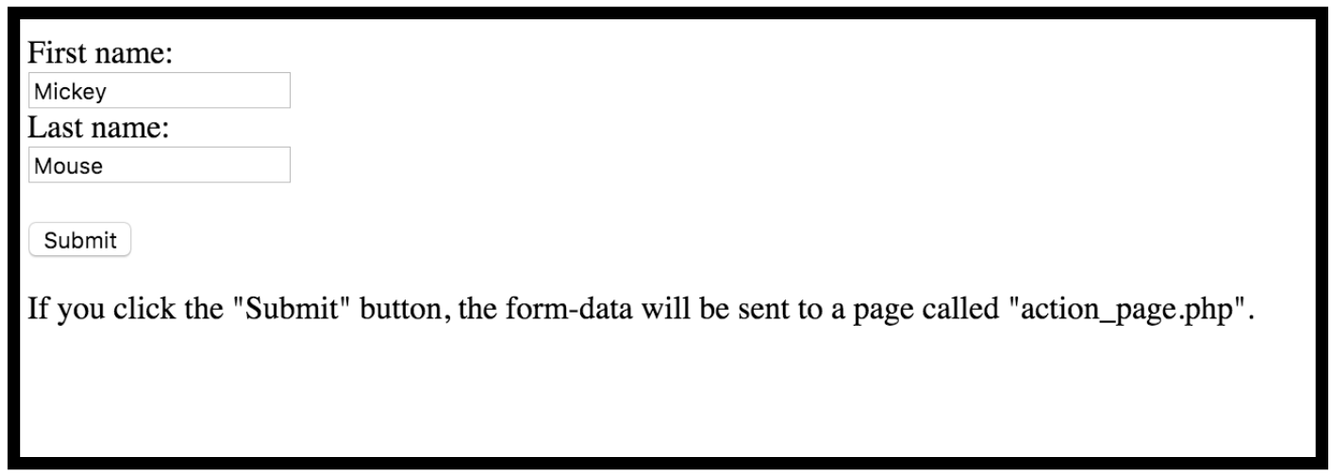


BASIC "INTERACTIONS"

Web forms:

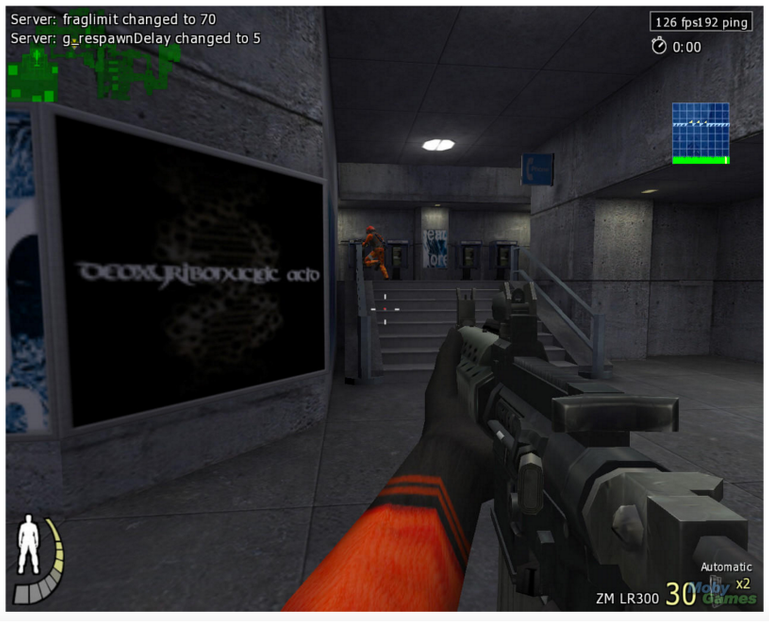
- The attribute ***action*** sets the web URI that will handle the request
- The attribute ***method*** will set the HTTP request method ("get" or "post")

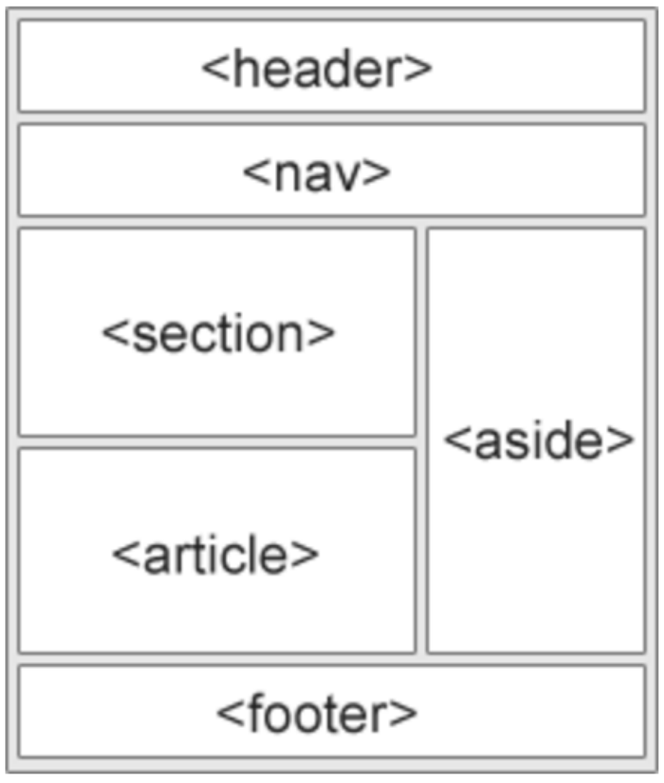
```
<!DOCTYPE html>
<html>
<body>

<form action="action_page.php" method="post">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

<p>If you click the "Submit" button, the form-data will be sent to a page called
"action_page.php".</p>

</body>
</html>
```



HTML5

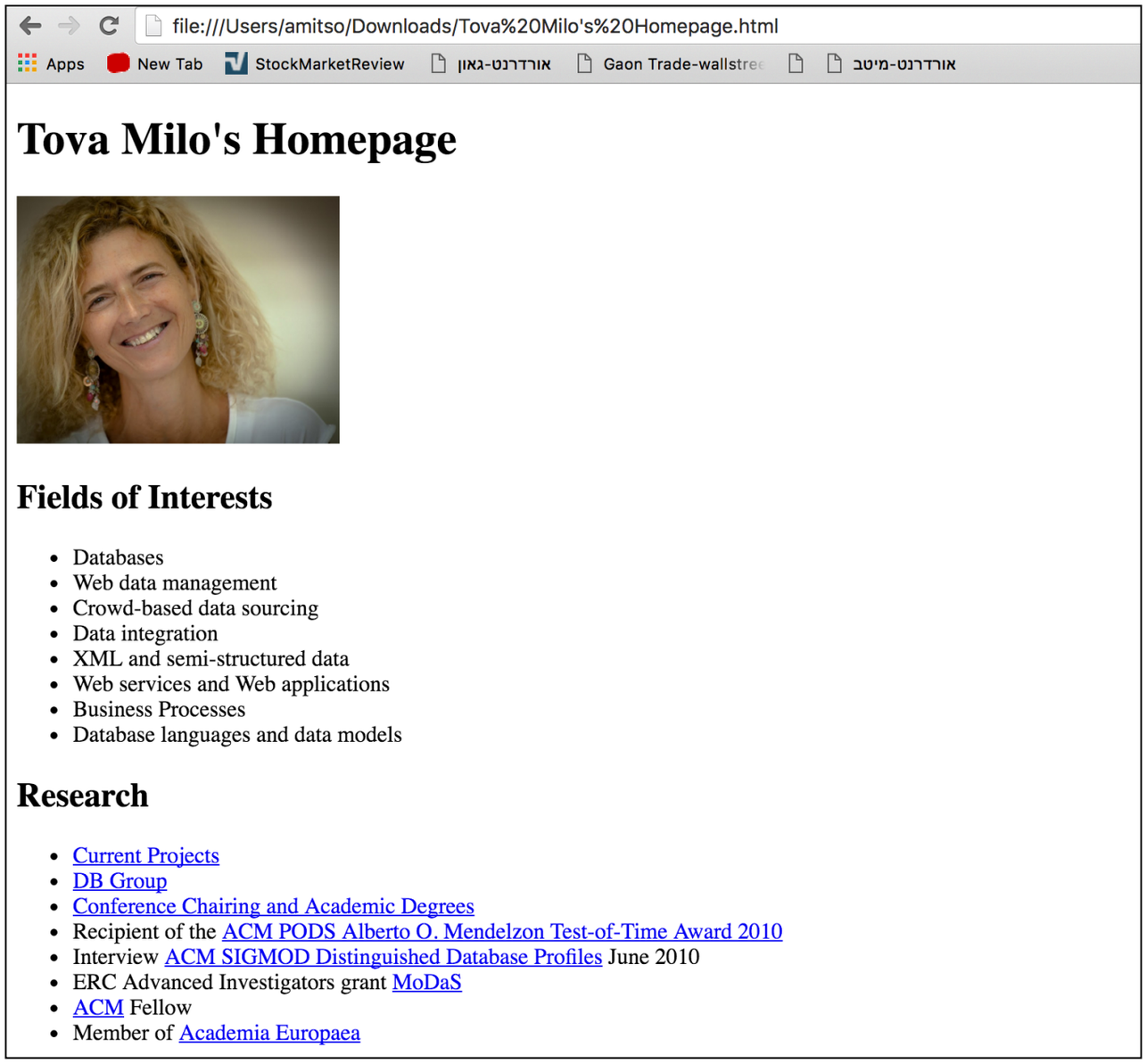
- HTML was pretty "basic" and needed many 3rd party plugins (e.g. Adobe Flash)
- HTML was not standardised and the programmer had to check the rendering of her code in all browser and handle irrational browser e.g., Internet Explorer.
- HTML5 was introduced in 2014 and includes new tags, attributes and cool features such as:
  - **Graphic elements**: <canvas>, <svg>, <video>, <audio>
  - **Semantic elements:** <footer>, <article>, <section>
  - **APIs:** Geolocation, Drag and Drop, Local Storage

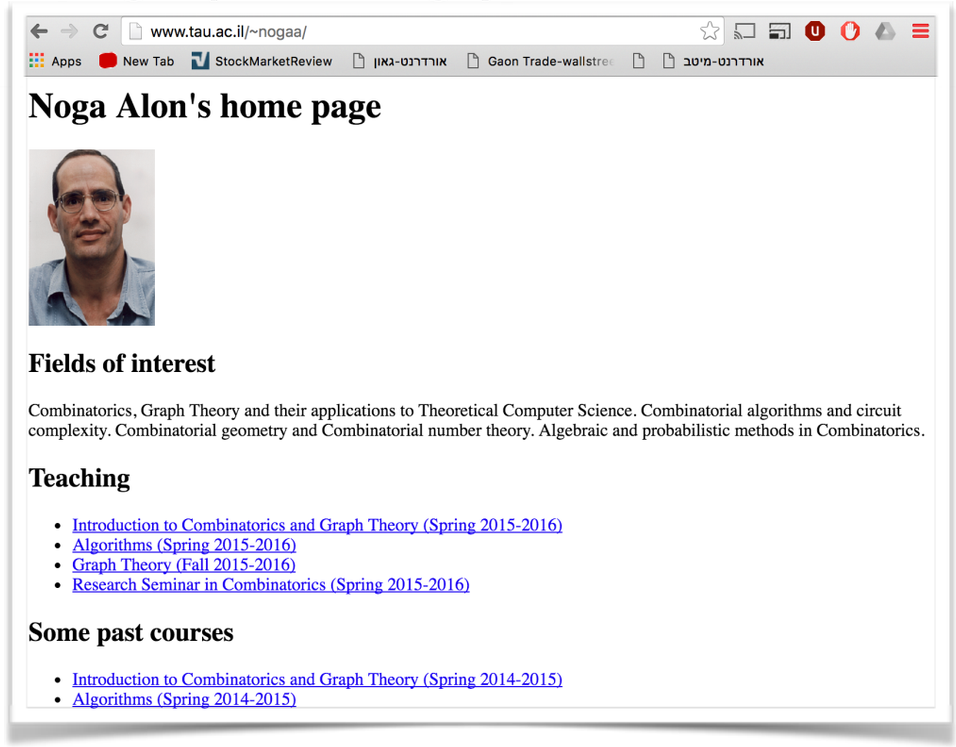




A STATIC WEB PAGE (WITH STYLE)

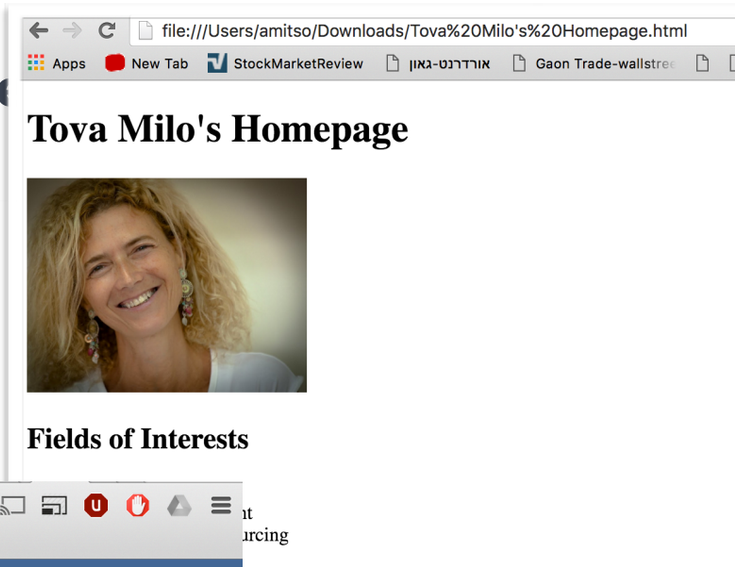
- **So far we learned how to structure the content of a webpage (Like Noga)**
- **This is Tova's website without style. Looks familiar?**

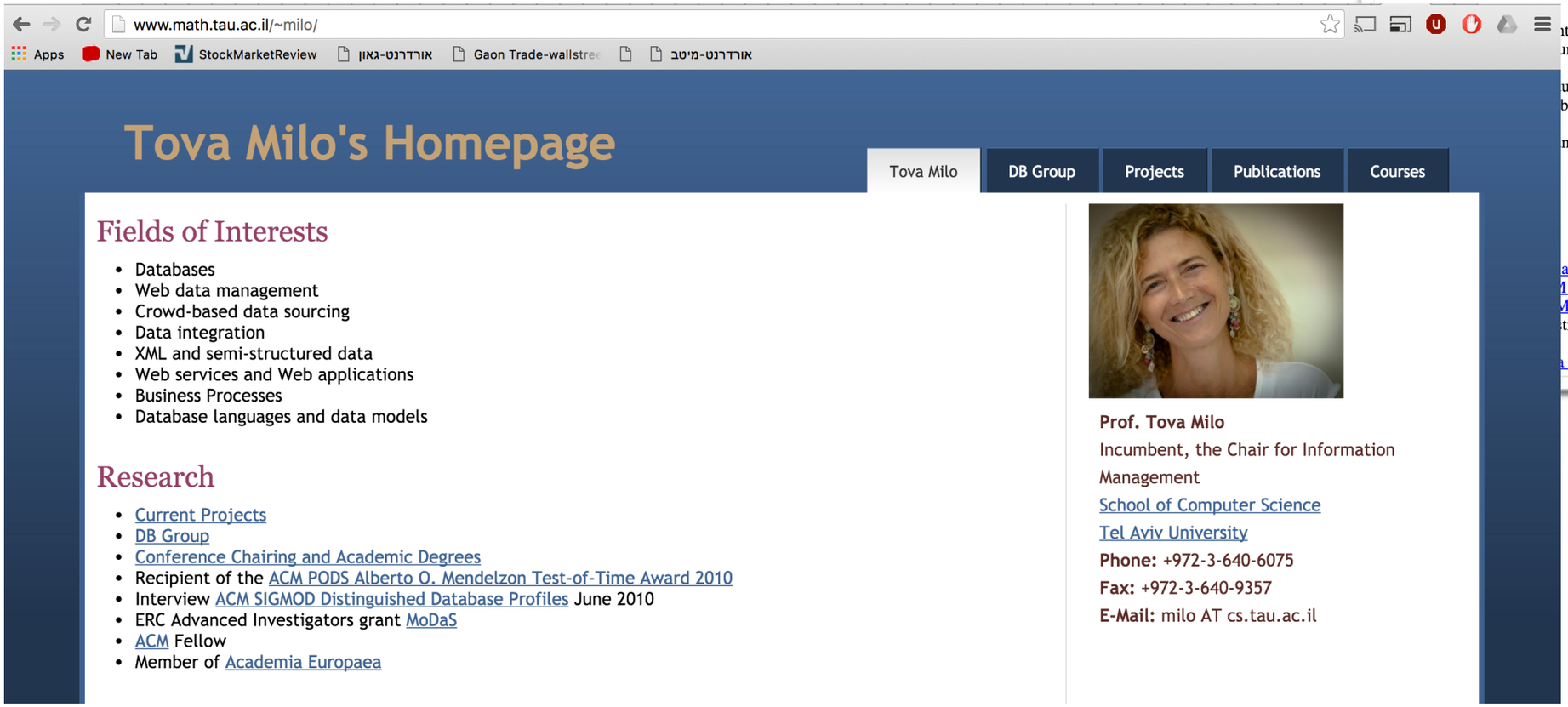




A STATIC WEB PAGE (WITH STYLE)

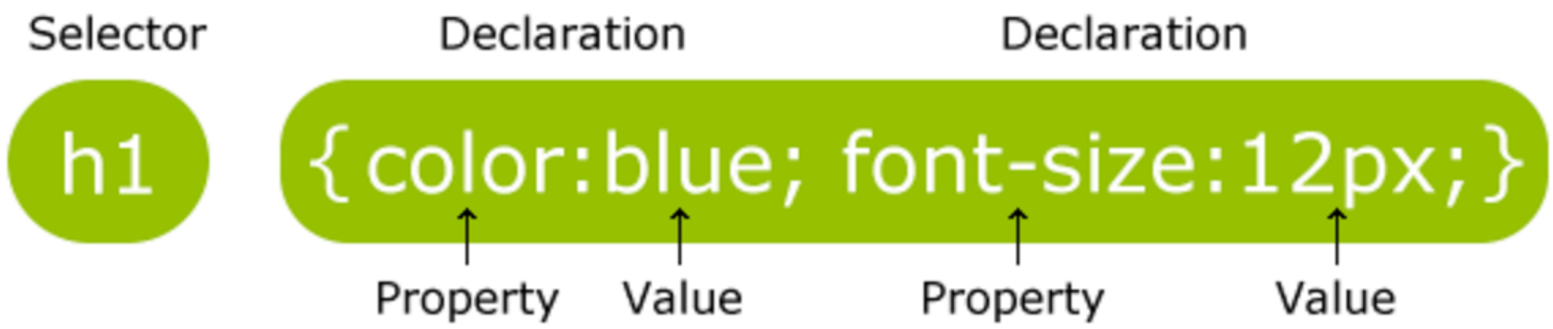
- **For adding some “style”, we use a CSS (Cascading Style She**





CSS FORMAT

How to set the style of an element:



Example of a CSS file:

```
html, * {
        margin:0 auto;
        padding:0;
}
body {
        background-image:url('images/bg-gradient.png');
        background-repeat:repeat-x;
        background-color:#23364E;
        margin:0 auto;
        padding:0;
        font-size:1.0em;
        font-family:"Trebuchet MS", Verdana, Arial;
}


/* table */
table
{
        margin:0;
}


/* GROUP MEMBER IMAGE */
.group_image {
        height: 120px;
        float: right;
}
```

EMBEDDING STYLING IN HTML

- **You can use multiple style sheets.**
- **FYI: Your browser has its own CSS file that is used by default.**
- **Cascading order (first one has the highest priority):**

1. **Inline style (inside an HTML element)**
2. **External and internal style sheets (in the head section)**
3. **Browser default**

EMBEDDING STYLING IN HTML

1.External CSS file: Include a link to the stylesheet file under the <head> tag of your HTML file:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Tova Milo's Homepage</title>
<link rel="stylesheet" href="http://www.cs.tau.ac.il/~milo/design/styles.css" type="text/css" />
</head>
```

2.Internal Stylesheet: Include a tag <style> under the <head> tag:

```
<head>
<style>
body {
    background-color: linen;
}

h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
```

EMBEDDING STYLING IN HTML

3.Inline styling: by adding the attribute *style*:

```
<h1 style="color:blue;margin-left:30px;">This is a heading.</h1>
```

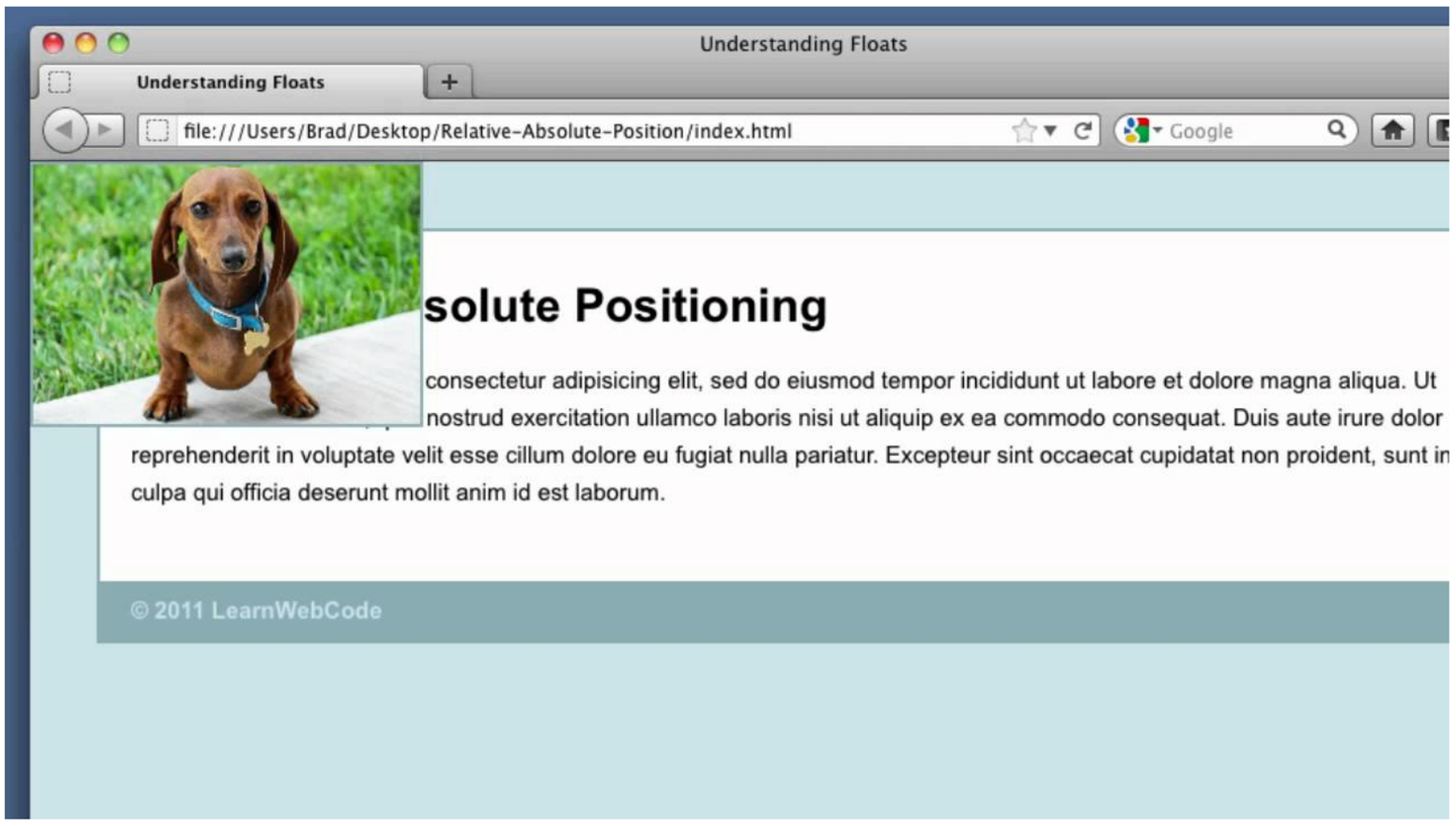
WHY PEOPLE HATE CSS?

1. Inheritance of style. ("I changed the font size but I can't see the changes")

   The best tip in this lecture: use **!important**

2. Positioning of elements ("This stupid DIV keeps floating over the title")

3. The box model ("Wait, is it margin-right? or padding-right? I'll try both and see what happens")



CSS: DISPLAY AND POSITION

- **These are the most important attributes in CSS.**
  - FYI: It is a nightmare to deal with. Go through this tutorial : http://www.w3schools.com/css/css_positioning.asp
- **There are 2 types of elements: *Block* and *Inline***
  - **Block (e.g. DIV, FORM,H1..H6):** starts in new line , always extend to the full width available.
  - **Inline (e.g. SPAN, IMG, A)** does not start on a new line and only takes up as much width as necessary
- **The *Display* attribute:** can alter the element's type or hide it completely.

Value	Description
inline	Default value. Displays an element as an inline element (like)
block	Displays an element as a block element (like <p>)
inline-block	Displays an element as an inline-level block container. The inside of this block is formatted as block-level box, and the element itself is formatted as an inline-level box
list-item	Let the element behave like a element
none	The element will not be displayed at all (has no effect on layout)
initial	Sets this property to its default value. Read about *initial*
inherit	Inherits this property from its parent element. Read about *inherit*

THE BOX MODEL

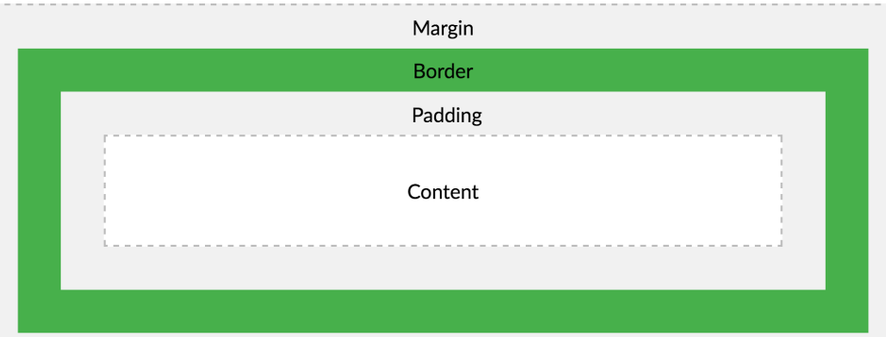
All HTML elements are considered as “boxes” . The box model allows us to add a border around elements, and to define space between elements:

Content: The content of the box, where text and images appear

Padding: Clears an area around the content. Padding is transparent

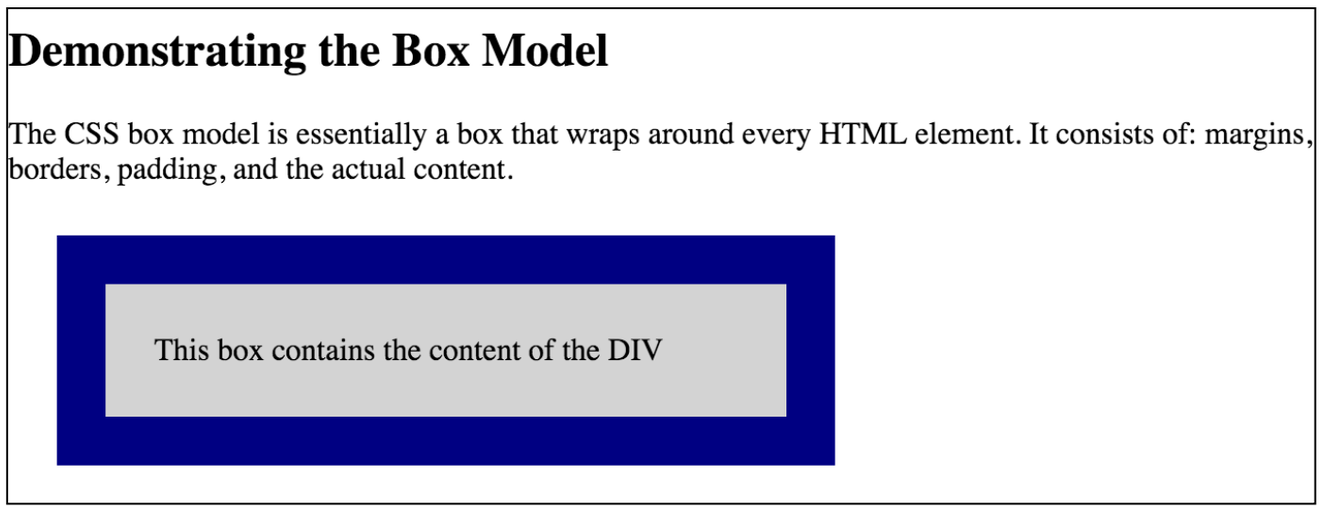
Border: A border that goes around the padding and content

Margin: Clears an area outside the border. The margin is transparen



For Example:

```
div {
    width: 300px;
    padding: 25px;
    border: 25px solid navy;
    margin: 25px;
}
```



CSS: DISPLAY AND POSITION

Positioning of elements:

static	Default value. Elements render in order, as they appear in the document flow
absolute	The element is positioned relative to its first positioned (not static) ancestor element
fixed	The element is positioned relative to the browser window
relative	The element is positioned relative to its normal position, so "left:20px" adds 20 pixels to the element's LEFT position
initial	Sets this property to its default value. Read about *initial*
inherit	Inherits this property from its parent element. Read about *inherit*

Example:

```
div.relative {
    position: relative;
    width: 400px;
    height: 200px;
    border: 3px solid #73AD21;
}

div.absolute {
    position: absolute;
    top: 80px;
    right: 0;
    width: 200px;
    height: 100px;
    border: 3px solid #73AD21;
}
```

This element has position: relative;

This element has position: absolute;

CSS SELECTORS

- **Selection by element ID:** (Use when addressing unique elements)

HTML

```
<div id="content">
   Text
</div>
```

CSS

```
#content {
        width: 200px;
}
```

- **Selection by element class:** (can be used for multiple elements)

HTML

```
<div class="big">
   Text
</div>
<div>
   <span class="big">some text </span>
</div>
```

CSS

```
.big{
        width: 200px;
}
```

MORE CSS SELECTORS

- **Selection by tag:**

HTML

```
<div>
   Text
</div>
<div>
   <span>some text </span>
</div>
<span>some other text </span>
```

CSS

```
div {
        width: 200px;
}
span {
        font-size:130%;
}
```

- **Grouping selection:**

```
H1, P , .main {
   font-weight:bold;
}
```

- **Descendant selection:**

HTML

```
<div class="abc">
  <div>
    <P>
      Hello there!
    </p>
  </div>
</div>
```

CSS

```
DIV.abc P {
   font-weight:bold;
}
```

ONE MORE CSS SELECTOR

- **Attributes selection** (Attribute selectors selects elements based upon the attributes present in the HTML Tags and their value)**:**

```
IMG[src="small.gif"] {
        border: 1px solid #000;
 }
```

CSS PSEUDO-CLASSES

Use to refer elements in different stages of execution.

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a specified range
:invalid	input:invalid	Selects all <input> elements with an invalid value
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"

```
/* unvisited link */
a:link {
    color: #FF0000;
}

/* visited link */
a:visited {
    color: #00FF00;
}

/* mouse over link */
a:hover {
    color: #FF00FF;
}

/* selected link */
a:active {
    color: #0000FF;
}
```

CSS PSEUDO-ELEMENTS

- **Used to generate HTML content automatically.**

Selector	Example	Example description
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::selection	p::selection	Selects the portion of an element that is selected by a user

- **Using the special attribute *Content*:**

```
p::after {
    content: " - Remember this";
}
```

RESPONSIVE WEB DESIGN

- **Responsive web design makes your web page look good on all devices**
- **Use only HTML & CSS to resize, hide, shrink, enlarge, or move the content to make it look good on any screen**

RESPONSIVE WEB DESIGN

- **Viewoports:**
  - **1. Do NOT use large fixed width elements**
  - **2. Use CSS media queries to apply different styling for small and large**

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

Without the viewport meta tag

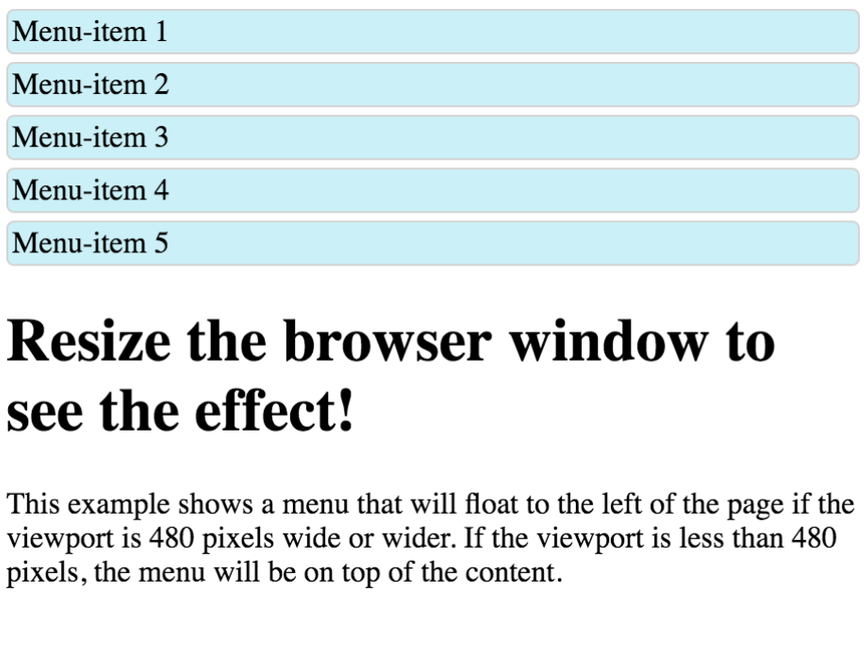
With the viewport meta tag

RESPONSIVE WEB DESIGN

- **Media queries:**

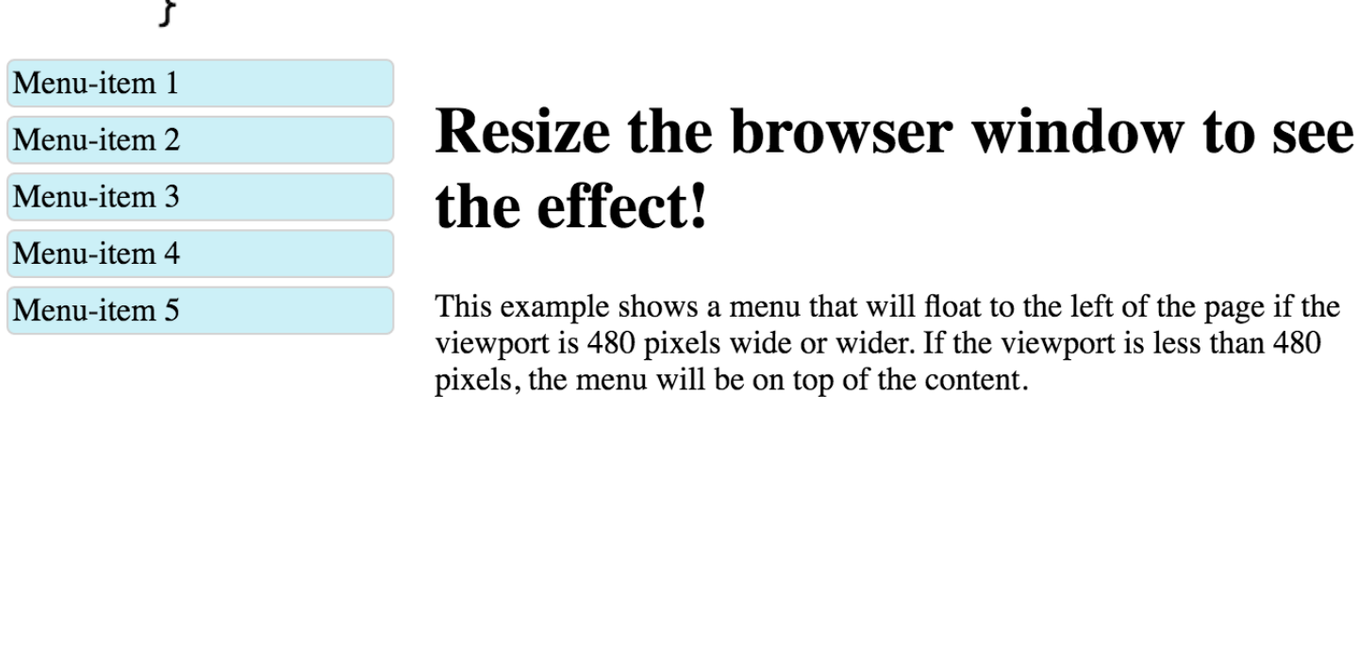
Mobile (default)

```
#main {margin-left: 4px;}
#leftsidebar {float: none;width: auto;}
```



Wider screens media query

```
@media screen and (min-width: 480px) {
    #leftsidebar {width: 200px; float: left;}
    #main {margin-left:216px;}
}
```



JAVASCRIPT: WHAT, WHY, HOW

- **Each HTML web page is standalone. Using JavaScript we can make a dynamic, single page and self-contained web application.**
- **What is Javascript? It is a client-side programming language.**
  - It is not Java and not related to Java by nothing. (Sun was involved somehow and therefore the name) . Actually the syntax is based on C.
  - Code is evaluated by (and only by) the web browser
- **Why Javascript?**
  - Make your website "dynamic":
    - Handle browser events (e.g. click on a link, pressing a key)
    - Send asynchronous HTTP requests
  - Mine bitcoins, and basically do any complex, logical operation.
- **To embed javascript, include the tag <script>**
  - Internal: Just type your JS code in between the script tags
  - External (recommended) **<script src=external.js > </script>**

JAVASCRIPT: HELLO WORLD

- **Javascript basic features:**
  - **Traverse the tree using the *document* reserved word.**
  - **Function *getElementByID("<id>"):*** finds the HTML element
  - **Variable *innerHTML*** : holds the element HTML content

HelloWorld example

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

<h1>My First Page</h1>

<p id="demo"> This is going to be overwritten by javascript </p>

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

</body>
</html>
```

JAVASCRIPT: BROWSER EVENTS

This are the main events that happen in the web browser (there are more):

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

With these events JS can do:

- Things that should be done every time a page loads/closed.
- Action that should be performed when a user clicks a button.

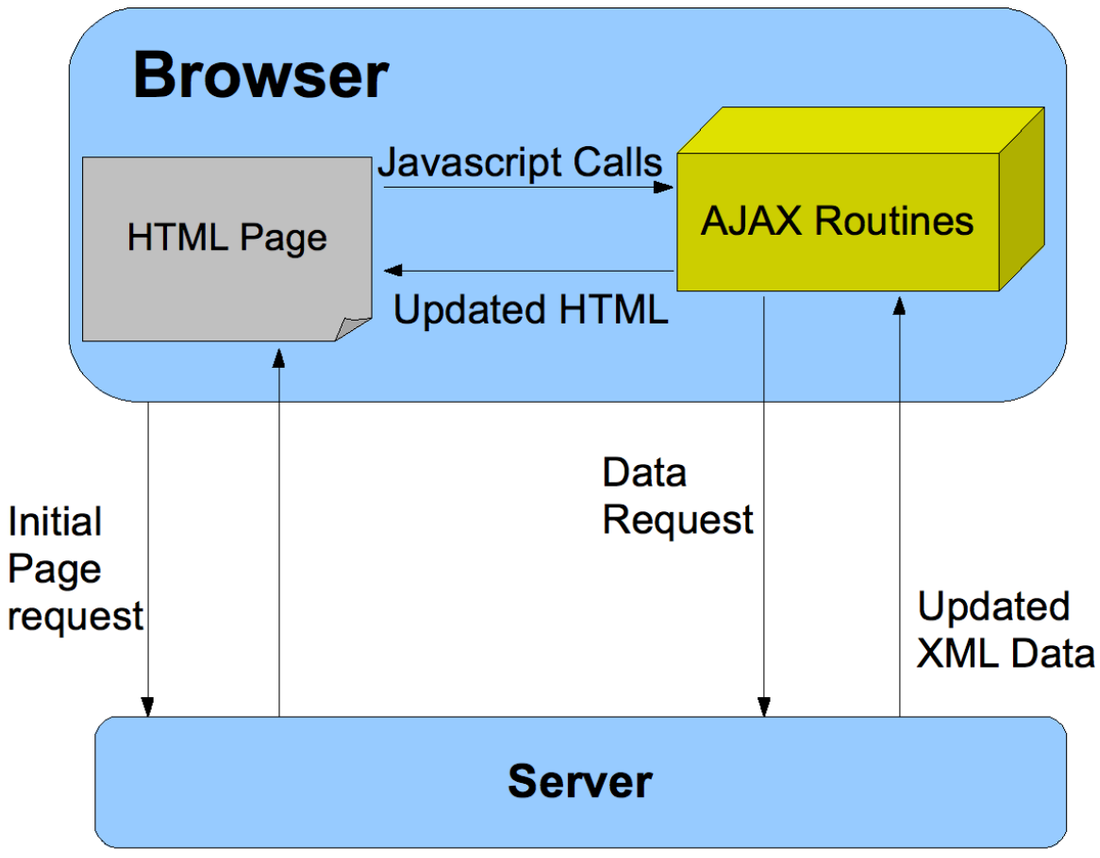
Important:

- HTML event attributes can execute JavaScript code directly / call JavaScript Functions.
- You can assign your own event handler functions to HTML elements/ prevent handling events

ASYNCHRONOUS REQUESTS

AJAX: asynchronous JavaScript and XML. Lets you:

- Update a web page without reloading the page
- Request and receive data from a server - after the page has loaded
- Send data to a server - in the background

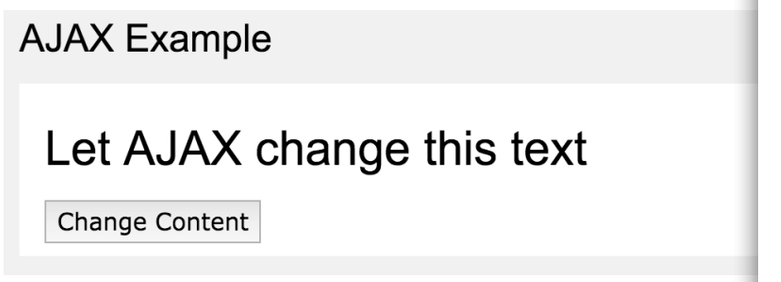


JAVASCRIPT: AJAX

An example showing everything :

- The HTML page contains a section and a <button>.
- The section is used to display information from a server.
- The <button> calls a function (if it is clicked).
- The function requests data from a web server and displays it:

•BEFORE CLICKING




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

<div id="demo"><h2>Let AJAX change this text</h2></div>

<button type="button" onclick="loadDoc()">Change Content</button>

</body>
</html>
```

•AFTER CLICKING

JAVASCRIPT: AJAX

The JavaScript Code:

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
     document.getElementById("demo").innerHTML = xhttp.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

Ready states:

onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

FRONT-END FRAMEWORKS

Why designing a page from scratch when you can rely on existing libraries that extend HTML, CSS and JavaScript?

★**Bootstrap:** An HTML+CSS+JavaScript framework for developing **Responsive** websites

★**AngularJS/ ReactJS:** extends HTML by adding new tags and features

BOOTSTRAP

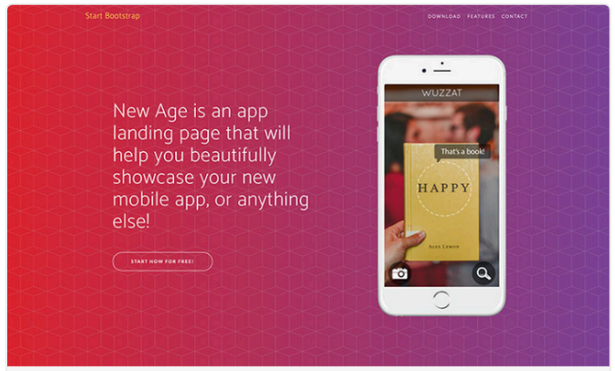
- **Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, “mobile-first" web applications**
- **It's completely free**
- **Many beautiful templates are available online**

Start Bootstrap / All Templates

Resume

A simple CV/resume theme.

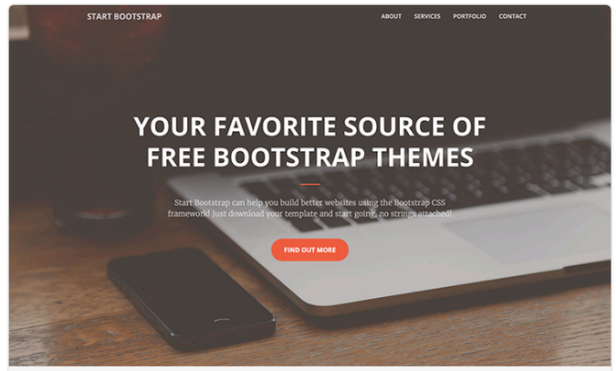
PREVIEW & DOWNLOAD



New Age

An app landing page theme.

PREVIEW & DOWNLOAD



Creative

A one page creative theme.

PREVIEW & DOWNLOAD

BOOTSTRAP

“installing” Bootstrap:

Just include these lines in your HTML header:

```
<head>
  <title>Hello, Bootstrap!</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</head>
```

BOOTSTRAP

Grid system:

The core of Bootstrap is the grid system, allowing you to define up to 12 responsive columns.

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4				
span 4				span 8								
span 6						span 6						
span 12												

- **The Bootstrap grid system has four classes:**

  xs (for phones - screens less than 768px wide)

  sm (for tablets - screens equal to or greater than 768px wide)

  md (for small laptops - screens equal to or greater than 992px wide)

  lg (for laptops and desktops - screens equal to or greater than 1200px wide)

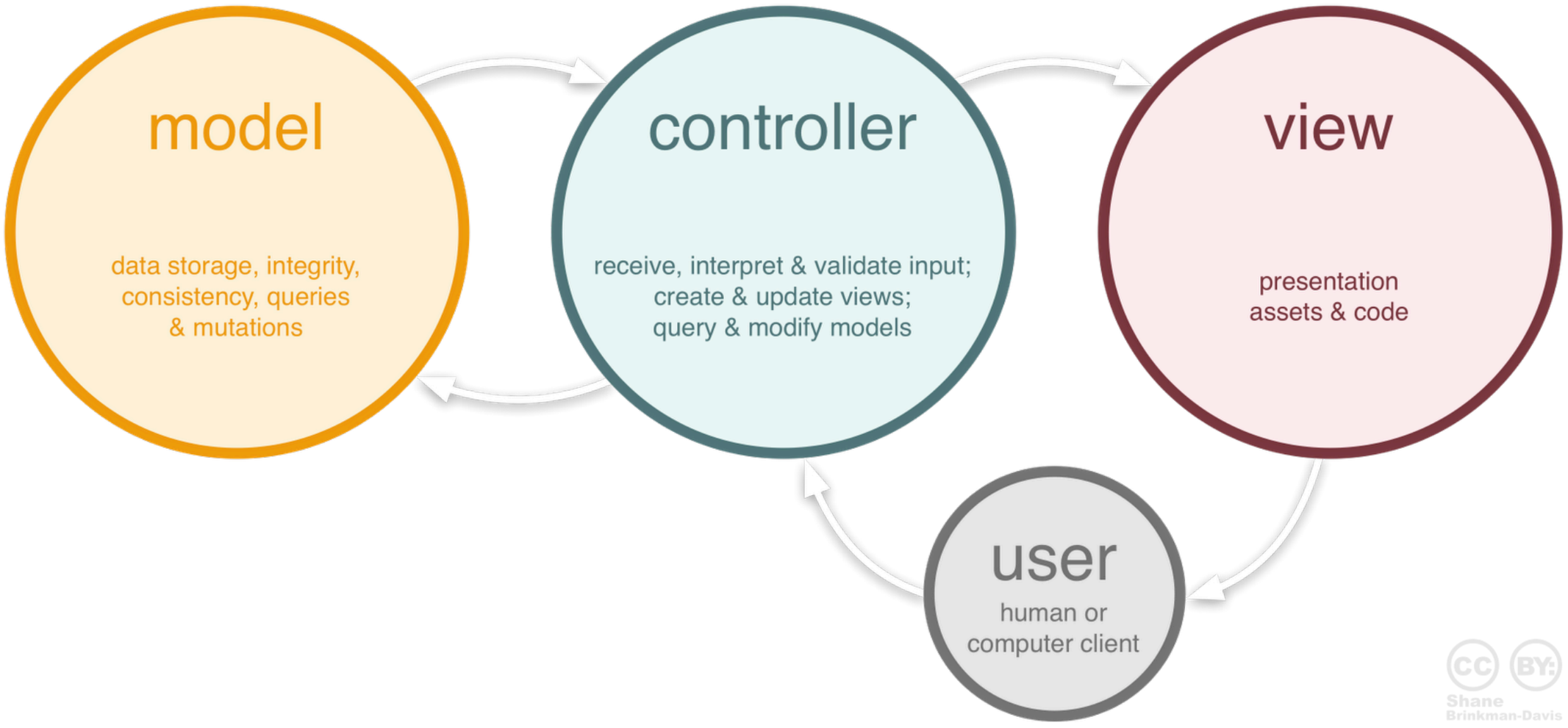
BOOTSTRAP

Hello world:

```
<div class="jumbotron text-center">
  <h1>Our first nice-looking web app</h1>
  <p>If you resize the window, you will see how "responsive" it is</p>
</div>
<div class="container">
  <div class="row">
    <div class="col-sm-4">
      <h3>Column 1</h3>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>
      <p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 2</h3>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>
      <p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>
    </div>
    <div class="col-sm-4">
      <h3>Column 3</h3>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>
      <p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>
    </div>
  </div>
</div>
```

MVC FRAMEWORKS

- **The MVC approach separates the UI (views) from the data and logics (models) and let them communicate via designated I/O methods (controllers)**
- **Examples: Angular-js, React-js, Redux, etc.**



ANGULAR JS: EXAMPLE

- **How to install Angular? Simply include the following script in your <head>**

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

- **A bit complicated example for creating a table with data from the server:**

```
<div ng-app="myApp" ng-controller="customersCtrl">

<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>

</div>

<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
    $http.get("http://www.w3schools.com/angular/customers.php")
    .then(function (response) {$scope.names = response.data.records;});
});
```

```
records [15]
▼ 0  {3}
    Name : Alfreds Futterkiste
    City : Berlin
    Country : Germany
▼ 1  {3}
    Name : Ana Trujillo Emparedados y helados
    City : México D.F.
    Country : Mexico
```

ANGULAR JS: EXAMPLE

- **The results:**

Alfreds Futterkiste	Germany
Ana Trujillo Emparedados y helados	Mexico
Antonio Moreno Taquería	Mexico
Around the Horn	UK
B's Beverages	UK
Berglunds snabbköp	Sweden
Blauer See Delikatessen	Germany
Blondel père et fils	France
Bólido Comidas preparadas	Spain
Bon app'	France
Bottom-Dollar Marketse	Canada
Cactus Comidas para llevar	Argentina
Centro comercial Moctezuma	Mexico
Chop-suey Chinese	Switzerland
Comércio Mineiro	Brazil

AGENDA FOR TODAY

- **Part #1: Client side programming:**
  - HTML, CSS Javascript
  - Responsive design
  - Front-end frameworks
- **Part #2: Server side programming**
  - Python (Flask): Install, hello world, requests, cookies, sessions.
- Using the university web servers.
- **Part #3: Using the university servers (python/ php)**

PYTHON (FLASK) - HELLO

Flask is a web-server library. Let's hello world it:

1.Install flask - you already know by now :-)

2.Create server.py:

```python
from flask import Flask, render_template
import datetime
app = Flask(__name__)


@app.route('/')
def hello_world():
    return "Hello dear DB students!"


if __name__ == '__main__':
    app.run(port=8888, host="0.0.0.0", debug=True)
```

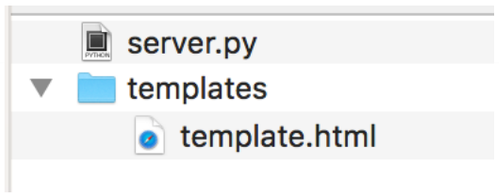
3. Run the server by:

```
Projects python3 server.py
Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
```

TEMPLATES

1.We use static HTML templates. For everything.

2.Create a template folder :



4.Create template.html:

```
1  <html>
2      <body>
3          <h1>Today is {{the_day}}</h1>
4      </body>
5  </html>
```

5. Edit server.py

```
1  from flask import Flask, render_template
2  import datetime
3  app = Flask(__name__)
4
5
6  @app.route('/template')
7  def use_templates():
8      return render_template('template.html', the_day=datetime.datetime.today())
9
```

TEMPLATES (ADVANCED)

Template file

```
<html>
    <body>
        <h1> Behold! results for something: </h1>
        <table border = 1>
            {% for key, value in result.iteritems() %}
                <tr>
                    <th> {{ key }} </th>
                    <td> {{ value }} </td>
                </tr>
            {% endfor %}
        </table>
    </body>
</html>
```

Server.py

```
@app.route('/advanced_template')
def advanced_template():
    dict = {'python': 100, 'php': 90, 'ruby': 70}
    return render_template('table.html', result=dict)
```

HANDLING REQUESTS

The HTTP request is passed to the web server and Flask gives it to you as a request object.

Important attributes:.

- **args** – parsed contents of query string which is part of URL after question mark (?).
- **form** – It is a dictionary object containing key and value pairs of form parameters and their values
- **cookies** – dictionary object holding Cookie names and values.
- **files** – data pertaining to uploaded file.
- **method** – current request method (GET/POST/…)

HANDLING REQUESTS

Login page template:

```
<html>
   <body>
      <h1> This is our login page!</h1>
      <form action = "http://localhost:8888/login" method = "post">
         <p>Enter your name:</p>
         <p><input type = "text" name = "name" /></p>
         <p>Enter your password:</p>
         <p><input type = "password" name = "pass" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

HANDLING REQUESTS

Server side code:

```python
from flask import Flask, render_template, redirect, url_for, request
import datetime
app = Flask(__name__)


@app.route('/login', methods=['POST', 'GET'])
def login():

    if request.method == 'GET':
        return render_template('login.html')

    elif request.method == 'POST':
        user = request.form['name']
        password = request.form['pass']
        if user == 'amit' and password == '1234':
            return redirect(url_for('post_login', name=user))
        else:
            return redirect(url_for('post_login', name='fail'))


@app.route('/after_login/<name>')
def post_login(name):
    if name == 'fail':
        return 'failedddddddd'
    else:
        return 'welcome %s' % name
```

COOKIES

A browser cookie is a small piece of data stored in the web browser

- Useful since the internet (and HTTP) (and PHP) are **STATELESS** .
- Example use: Your shopping cart in **amazon.com**.
- Cookies are saved per web page.
- The browser will send the cookie content to the web-page if it contains one.

In Flask:

- The **Request** object contains a cookie's attribute.
- cookies are set on **Response** object

COOKIES

Server side code:

```python
@app.route('/login2', methods=['POST', 'GET'])
def login2():

    if request.method == 'GET':
        return render_template('login2.html')

    elif request.method == 'POST':
        user = request.form['name']
        password = request.form['pass']
        if user == 'amit' and password == '1234':
            is_successful = 'True'
        else:
            is_successful = 'False'
            user = 'fail'
        resp = make_response(redirect(url_for('post_login2', name=user)))
        resp.set_cookie('successful_login', is_successful)
        return resp


@app.route('/after_login2/<name>')
def post_login2(name):
    is_successful = request.cookies.get('successful_login')
    if is_successful == 'True':
        return 'welcome %s' % name
    else:
        return 'failedddddddd'
```

SESSIONS

Unlike cookies, “sessions” are stored on the server…..

- Session is the time interval when a client logs into a server and logs out of it.
- The data, which is needed to be held across this session, is stored in a temporary directory on the server.

In Flask:

- A session with each client is assigned a Session ID.
- The Session data is stored on top of cookies and the server signs them cryptographically.
- For this encryption, a Flask application needs a defined SECRET_KEY.

SESSIONS

```python
from flask import Flask, render_template, redirect, url_for, request, \
    make_response, session
import datetime
app = Flask(__name__)
app.secret_key = 'itsasecret'


@app.route('/login3', methods=['POST', 'GET'])
def login3():

    if request.method == 'GET':
        if 'is_logged_in' in session:
            if session['is_logged_in'] is True:
                return redirect(url_for('post_login3',
                                        name=session['user_name']))
        return render_template('login3.html')

    elif request.method == 'POST':
        user = request.form['name']
        password = request.form['pass']
        if user == 'amit' and password == '1234':
            session['is_logged_in'] = True
            session['user_name'] = user
        else:
            session['is_logged_in'] = False
        return redirect(url_for('post_login3', name=user))

    return ''


@app.route('/after_login3/<name>')
def post_login3(name):
    if 'is_logged_in' in session:
        if session['is_logged_in'] is True:
            return 'welcome %s' % session['user_name']

    return 'failedddddddd'
```

UNIVERSITY SERVERS

PYTHON

1. Full instructions: http://www.cs.tau.ac.il/system/django
2. Supports Flask and Django
3. Server name: **delta-tomcat-vm** (accessible from uni only)
4. Instructions in a brief:
   1. SSH to delta-tomcat-vm and run: sudo create-my-django-dir
   2. Copy your application files to your django dir: /specific/scratch/<username>/django/
   3. Choose an available port and use it in your configuration file.
   4. Run your app *detached*:

```
nohup python3 -m "from my_flask_web_server import run_server;
run_server.run_server()" &
```

UNIVERSITY SERVERS

1. Connect to NOVA
2. Create a new directory called html
3. Create your project dir under ~/html
4. Create a “hello.php” file under ~/html/project
5. Access the url from your web browser: cs.tau.ac.il/~<your_username>/project/hello.php

THIS IS THE END?

- **We overviewed the world of web programming in a brief**
  - "hello-world"-ing
  - Bad news: This is barely enough to get started
  - Good news: You know enough to continue learning this by yourself
- **Use the online resources:**
  - w3schools
  - Google
  - system@cs.tau.ac.il
  - Our class forum
- **Come to projects day**
- **Bye.**

: INTRODUCTION

PHP stands for *"PHP: Hypertext Preprocessor"*

Why PHP:

★PHP is one of the leading web development languages.

★PHP is compatible with almost all servers used today (Apache, IIS, etc.)

★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

What can PHP do:

★PHP can generate dynamic HTML content

★PHP can collect and process user input from GET and POST requests

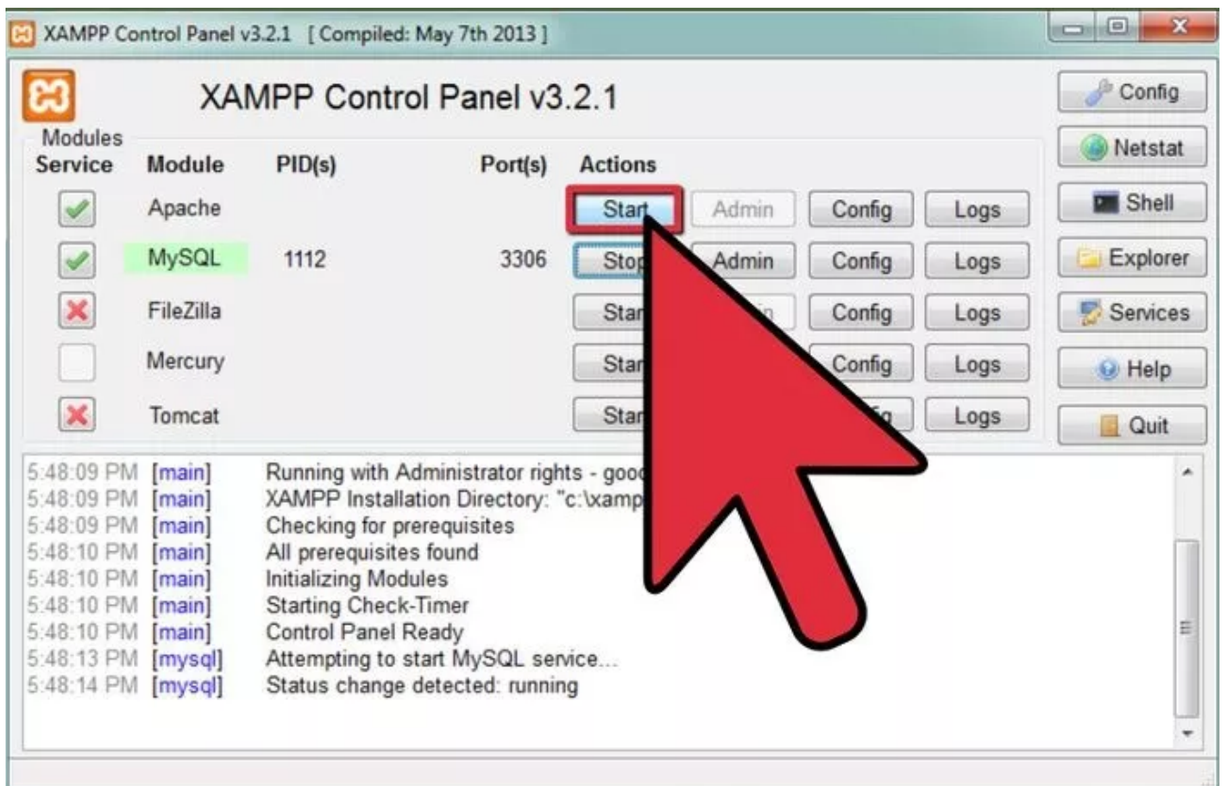
★PHP can send and receive cookies

★PHP can add, delete, modify data in your database

INSTALL PHP

On your local machine (Windows):

1. Install XAMPP (or WAMP) from https://www.apachefriends.org/download.html
2. It will install Apache (web server) + MySQL database + PHP + phpMyAdmin
3. Open the XAMPP control panel and click **Start** on everything.
4. Open the folder C:/xampp/htdocs and create hello.php
5. Access from your web browser:
    1. http://localhost/hello.php



PHP: HELLOWORLD

hello.php

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

PHP: HANDLING REQUESTS

1.This is a simple HTML form in a static page (form.html)

- Note that it contains the parameters **Name** and **Email**.
- These are sent to **welcome.php** via a POST request

```
<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>
```

2.What does welcome.php looks like?

- Using $_POST superglobal array to access parameters

```
<html>
<body>

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

</body>
</html>
```

3.What does the user see?

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

PHP: SUPER GLOBALS

SuperGlobals: PHP has several predefined arrays that are "super globals": means they are available in all scopes throughout a script without using any special prefix.

And they are:

- ***$GLOBALS*** stores all global variables
- ***$_SERVER*** stores information about the current server e.g. path of the script, server name.
- ***$_GET*** stores the parameters that are passed via HTTP GET
- ***$_POST*** stores the parameters that are passed via HTTP POST
- ***$_FILES*** stores **files** that are uploaded to the server via HTTP POST
- ***$_COOKIE*** stores the parameters of the HTTP cookie
- ***$_SESSION*** stores information for a user in a **session**.
- ***$_REQUEST*** *stores all data passed via GET and POST*

PHP: MISSING REQUEST PARAMETERS

What if some of the user didn't send a required parameter?

- Never trust the user, always validate input **on the server side!**

Like that:

- Use **$_SERVER** superglobal to read meta-data about the current request

```
<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>
```

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

PHP: INCLUDE FILES

PHP allows the programmer to include (or require) other scripts.

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

footer.php can be

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

When to use:

★Separate DB handling from the logics.

★Separate HTML generation from the logic.

★Dedicated files for methods and functions

PHP: COOKIES

HTTP Cookie (Browser cookie) is a small piece of data stored in the web browser

★Useful since the internet (and HTTP) (and PHP) are **STATELESS** .

★Example use: Your shopping cart in **amazon.com**.

★Cookies are saved per web page.

★The browser will send the cookie content to the web-page if it contains one.

PHP: SET COOKIES

Using setcookie(): setcookie(*name, value, expire, path, domain, secure, httponly*);

Example:

```php
<?php
$cookie_name  = "name";
$cookie_value = "John";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if (!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
```

The browser will sent the request :

Request Headers view source

Accept: text/css,*/*;q=0.1
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,he;q=0.6
Cache-Control: no-cache
Connection: keep-alive
Cookie: __gads=ID=121eef0c11a56bf3:T=1404899365:S=ALNI_MaB_krXY86lDuSUam-1wYC
IftA; __utma=119627022.1086267766.1404899365.1409010569.1409052375.4; _ga=GA
1086267766.1404899365; ASPSESSIONIDCCSSSCDQ=LAFHDOOBHNNHCANLLOOLHKCO; ASPSES
IDACRTSDCR=NBLBIODCHGJDHMMAOPKDMMPP; user=Alex+Porter

PHP: SESSIONS

- PHP Sessions: are a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored in the browser but in the server!
- How to:

1.Session_start() , the first thing on the page **even if the session is not new!**

2. Read and write to **$_SESSION** super global

3.session_unset() will delete all session variables

4.session_destroy() will destroy the session

PHP: SESSIONS

How is a PHP session created?

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called PHPSESSID is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

How does PHP retrieves session information?

- PHP automatically gets the unique session identifier string from the PHPSESSID cookie.
- then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

How do sessions end?

- When the cookie is lost.
- the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

PHP + MYSQL

PHP can use one of two methods for Database handling:

- **MySQLi** extension (the "i" stands for improved)
- **PDO** (PHP Data Objects)

```php
<?php
$servername = "mysqlsrv.cs.tau.ac.il";
$username   = "sakila";
$password   = "sakila";
$dbname     = "sakila";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$sql    = "SELECT rental_id,rental_date FROM rental WHERE inventory_id = 10 AND customer_id = 3";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while ($row = $result->fetch_assoc()) {
        echo "id: " . $row["rental_id"] . " - Date: " . $row["rental_date"] . "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

PHP + MYSQL

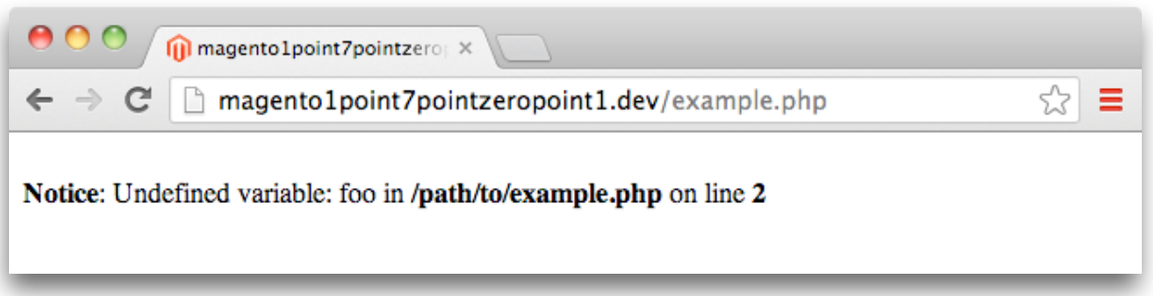
Prepared statements using:

- **prepare**
- **bind**
- **execute**

```php
// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname  = "Doe";
$email     = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname  = "Moe";
$email     = "mary@example.com";
$stmt->execute();
```

PHP ERROR HANDLING

- Remember: **PHP is not always configured to display errors and warnings**
- PHP stores all error and warning to a log.
- Depends on the configuration, it also prints annoying messages to the screen such as :



- Theses are the available error levels: use error_reporting() to control it:

Value	Constant	Description
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

APACHE+PHP ERROR LOGS

Are store in a file called *error.log* that can be found in the apache directory

- Sadly , the log is restricted in the uni web-server
- You will have to configure your php file to display error via the method ***error_reporting***

```
[31-Oct-2013 09:14:18] PHP Notice:  wp_register_script was called
<strong>incorrectly</strong>. Scripts and styles should not be registered or
enqueued until the <code>wp_enqueue_scripts</code>, <code>admin_enqueue_scripts</
code>, or <code>login_enqueue_scripts</code> hooks. Please see <a href="http://
codex.wordpress.org/Debugging_in_WordPress">Debugging in WordPress</a> for more
information. (This message was added in version 3.3.) in /var/www/vhosts/
ipadboardgames.org/httpdocs/wp-includes/functions.php on line 3012
[31-Oct-2013 09:14:18] PHP Notice:  add_custom_background is <strong>deprecated</
strong> since version 3.4! Use add_theme_support( 'custom-background', $args )
instead. in /var/www/vhosts/ipadboardgames.org/httpdocs/wp-includes/functions.php
on line 2871
[31-Oct-2013 09:14:18] PHP Notice:  register_widget_control is
<strong>deprecated</strong> since version 2.8! Use wp_register_widget_control()
instead. in /var/www/vhosts/ipadboardgames.org/httpdocs/wp-includes/functions.php
on line 2871
[31-Oct-2013 09:14:18] PHP Notice:  register_sidebar_widget is
<strong>deprecated</strong> since version 2.8! Use wp_register_sidebar_widget()
instead. in /var/www/vhosts/ipadboardgames.org/httpdocs/wp-includes/functions.php
on line 2871
```

APACHE+PHP CONF. FILES

Apache configurations are stored in a file called httpd.conf

- To make changes: You can make changes in the httpd.conf file then restart the server
- Holds informations such as the server port, supported modules etc.

PHP configurations are stored in a file called PHP.ini

```
; any text on a line after an unquoted semicolon (;) is ignored
[php] ; section markers (text within square brackets) are also ignored
; Boolean values can be set to either:
;    true, on, yes
; or false, off, no, none
register_globals = off
track_errors = yes

; you can enclose strings in double-quotes
include_path = ".:/usr/local/lib/php"

; backslashes are treated the same as any other character
include_path = ".;c:\php\lib"
```