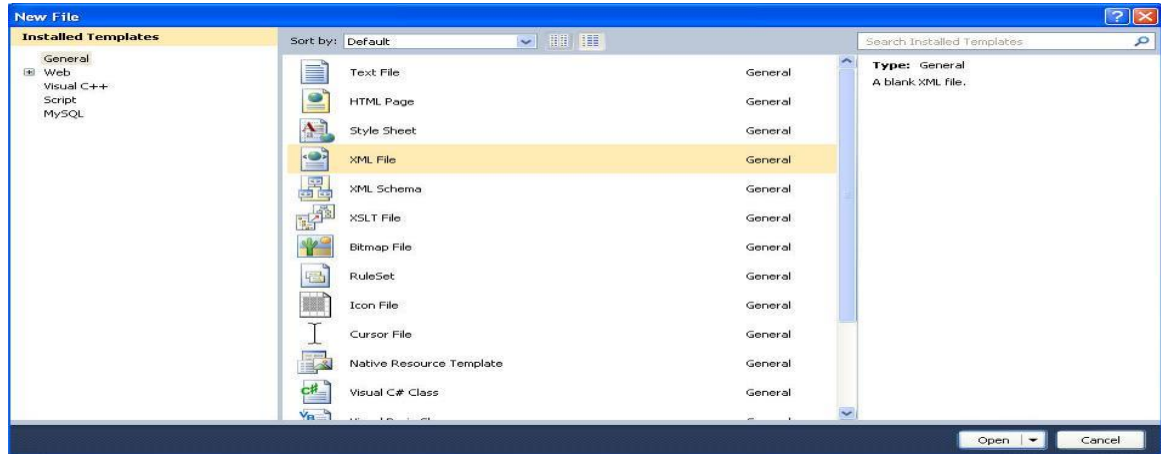


CIT270 Integrative Programming (Lab Assignment 5) (100 Points)

Exercise 1: XML, XSLT

Task 1: Creating an XML

1. Launch Microsoft Visual Studio.
2. Click **File> New File**.
3. Choose **XML File**, as follows and click **Open**.



4. Click **File> Save XMLFile1.xml**.
5. Add your name as a comment immediately beneath the XML declaration.
6. Add a root element named **Restaurants**.
7. Add a child element named **Restaurant**. Add the following attributes to the element you created in Step 6:

Attribute	Value
Name	Little Cabin Inn
QualityRating	3
PriceRating	2

8. Add a child element named **Location** to the **Restaurant** element.
9. Add the following child elements to the Location element:

Element	Value
Address	10 Main Street
City	Chicago
State	Illinois

10. Add a child element named **PhoneNumber** to the **Restaurant** element.
11. Add a child element named **Menu** to the **Restaurant** element.
12. Add a child element named **MenuItem** to the **Menu** element. Set its value to **Hamburger**.
13. Add the following attributes to the **Hamburger MenuItem**:

Attribute	Value
Price	\$4.95
Calories	450
IsVegetarian	false

14. Add another **MenuItem** with a value of **Fruit Salad**.
15. Add the following attributes to the **Fruit Salad MenuItem**:

Attribute	Value
Price	\$5.50
Calories	<300
IsVegetarian	true

16. Save the file.

Task 2: Editing the XML

1. Open browser.
2. Right-click **nnRestaurants.xml** and choose **Open With> Chrome**.
3. Verify that the results look like the following document. If they do not, correct any problems.

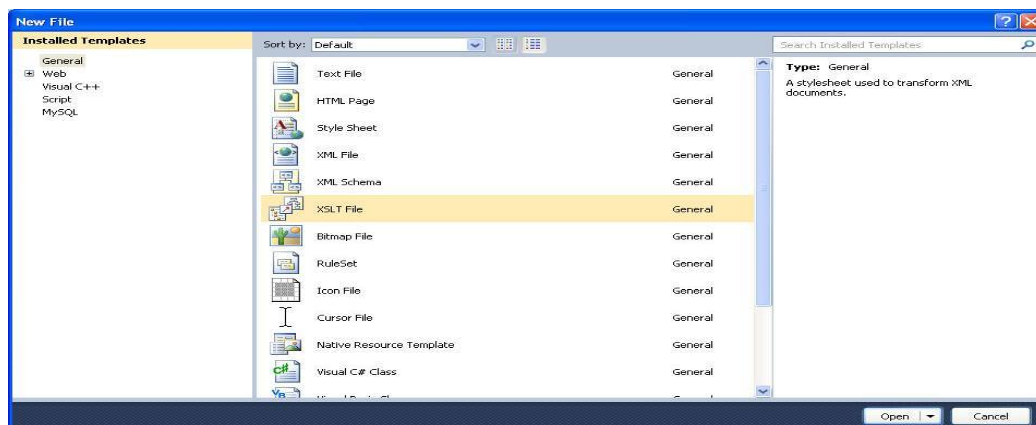
```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Student name -->
- <Restaurants>
- <Restaurant Name="Little Cabin Inn" QualityRating="3" PriceRating="2">
- <Location>
  <Address>10 Main Street</Address>
  <City>Chicago</City>
  <State>Illinois</State>
</Location>
- <Menu>
  <MenuItem Price="4.95" Calories="450" IsVegetarian="false">Hamburger</MenuItem>
  <MenuItem Price="5.50" Calories="<300" IsVegetarian="true">Fruit Salad</MenuItem>
</Menu>
</Restaurant>
</Restaurants>
```

4. Add two more restaurants, each with at least three menu items. One of the restaurants should not have any vegetarian menu items. At least two restaurants should be located in Illinois, and one should be located in California.
4. Save the file.
5. Open *nnRestaurants.xml* in Internet Explorer to verify that it is displayed correctly.
6. Expand or collapse one of the restaurants (HINT: You might need to enable JavaScript before you can perform this step).
7. Take a screenshot and paste it in your Word document.

Restaurant	
Name	City Dinner
QualityRating	4
PriceRating	3
Location	
Address	IL 605010
City	Naperville
State	Illinois
Menu	
PintoPosole	
Price	\$6.75
Calories	<150
IsVegetarian	true
CabbageSoup	
Price	\$3.50
Calories	120
IsVegetarian	true
OkraCorn	
Price	\$2.50
Calories	<400
IsVegetarian	true

Task 3: Transforming with XSLT

1. Click **File> New File**.
2. Select **XSLT File** and click **Open**.



3. Change the XSLT so that it looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-
result-prefixes="msxsl">

    <xsl:output method="html"/>
    <xsl:template match="/" >
        <html>
            <head>
                <title>Restaurants</title>
            </head>
            <body>
                <h1>Recommended Restaurant</h1>
                <p>In Chicago, I recommend
                <xsl:value of
select="Restaurants/Restaurant" />
                </p>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

4. Right-click the XSLT window and choose **Properties**.
5. Set the Input field to your XML file.
6. Save the file as *nnFirstTransform.xslt*.
7. Click **XML> Start XSLT Without Debugging**.
8. When prompted to select a file, select *nnRestaurants.xml*.
9. Take a screen shot of the results and paste it in the Word document.

Recommended Restaurant

In Chicago, I recommend 10 Main StreetChicagoIllinois

Question 1: *Is the name of the restaurant displayed? Why or why not?*

Answer: *No, because the xslt cannot get the name of the restaurant from the list defined in the xml document.*

-
10. Modify the select expression as follows:

```
select="Restaurants/Restaurant/@Name"
```

11. Click **XML> Start XSLT Without Debugging**. Take a screen shot of the results and paste it in your Word document.

Recommended Restaurant

In Chicago, I recommend SmokeHouse

12. Add the following statements beneath the paragraph block:

```
<h1>Restaurants by location</h1>

<table border="1">
  <tr>
    <th>Restaurant Name</th>
    <th>City</th>
    <th>State</th>
  </tr>
  <xsl:for-each
    select="Restaurants/Restaurant"> <tr>
    <td>
      <xsl:value-of select="@Name"/>
    </td>
    <td>
      <xsl:value-of select="Location/City"/>
    </td>
    <td>
      <xsl:value-of select="Location/State"/>
    </td>
  </tr>
</xsl:for-each>
</table>
```

13. Click **XML> Start XSLT Without Debugging**. Your output should look something like this:

Recommended Restaurant

In Chicago, I recommend Little Cabin Inn.

Restaurants by location

Restaurant Name	City	State
Little Cabin Inn	Chicago	Illinois
Main Street Grill	Chicago	Illinois
Emile's	San Jose	California

14. Take a screen shot and paste it into your Word document.

Recommended Restaurant

In Chicago, I recommend SmokeHouse

Restaurants by location

Restaurant Name	City	State
SmokeHouse	Chicago	Illinois
City Dinner	Naperville	Illinois
AtelierCream	San Francisco	California
Berkeley	Chicago	Illinois

15. Add the following element directly beneath the xsl:for-each statement:

```
<xsl:sort select="Location/State"/>
```

16. Test your change and take a screen shot.

Restaurants by location

Restaurant Name	City	State
AtelierCream	San Francisco	California
SmokeHouse	Chicago	Illinois
City Dinner	Naperville	Illinois
Berkeley	Chicago	Illinois

Recommended Restaurant

In Chicago, I recommend Little Cabin Inn

play output similar

Restaurants by location

Restaurant Name	City	State	QualityRating	PriceRating
Vernon	San Francisco	California	2	3
Little Cabin Inn	Chicago	Illinois	3	2
City Dinner	Naperville	Illinois	4	3
Delaware	Chicago	Illinois	3	4

Restaurants with Vegetarian Choices

Little Cabin Inn

Recommended Restaurant

In Chicago, I recommend SmokeHouse

Restaurants by location

Restaurant Name	City	State	QualityRating	PriceRating
AtelierCream	San Francisco	California	4	3
SmokeHouse	Chicago	Illinois	4	2
City Dinner	Naperville	Illinois	4	3
Berkeley	Chicago	Illinois	3	2

18. Add the following statements beneath the `</table>` tag:

```
<!--Restaurants that serve vegetarian-->
<h1>Restaurants with Vegetarian Choices</h1> <xsl:for-each
    select="Restaurants/Restaurant">
    <xsl:if test="Menu/MenuItem/@IsVegetarian='true'">
        <h3>
            <xsl:value-of select="@Name"/>
        </h3>
    </xsl:if>
</xsl:for-each>
```

19. Test your change to verify that restaurants that include vegetarian menu choices are output.

20. Take a screen shot and paste it in your Word document.

No output displayed

Restaurant Name	City	State	QualityRating	PriceRating
AtelierCream	San Francisco	California	4	3
SmokeHouse	Chicago	Illinois	4	2
City Dinner	Naperville	Illinois	4	3
Berkeley	Chicago	Illinois	3	2

Restaurants with Vegetarian Choices

21. Modify the XSLT to display a table that shows the menu items and the prices and calories of each vegetarian menu item. Your output should look something like this:

Restaurants with Vegetarian Choices

Little Cabin Inn

Menu item Price Calories

Fruit Salad 5.50 <300

Joe's

Menu item Price Calories

Veggie Melt 13.50 700

22. Test your change.

Exercise 2: Task 1: Using XPath to Set the Context

1. Launch Visual Studio.
2. Open the wonders_master.xml file from the Lab 5 Files folder.
3. Create a new XSLT file.
4. Save it as *nnLab5.xslt*, where the letters *nn* stand for your initials.
5. Open 05-05.xsl.
6. Copy the contents and paste them in *nnLab5.xslt*.

Question 1: *If you were to use this .xslt to process wonders_master.xml, which would be the current node when the file begins processing?*

Answer: *Since <XSL: apply-template /> is a recursive call to the template rule for the contents of the current element, then, "ancient_wonders" would be the current node to be processed in the XPath.*

-
7. Test the .xslt file using wonders_master.xml. Take a screen shot and paste it in your worksheet.

Seven Wonders of the Ancient World

Overview

Wonder Name	Location	Height
Great Pyramid of Giza	Giza, Egypt	455
Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας)	Alexandria, Egypt	384
Mausoleum at Halicarnassus (Μαυσωλεῖον Ἀλικαρνασσεύς)	Bodrum, Turkey	135
Colossus of Rhodes (Κολοσσός της Ρόδου)	Rhodes, Greece	107
Temple of Artemis at Ephesus (Ἄρτεμισιον)	Ephesus, Turkey	60
Statue of Zeus at Olympia (Δίας μυθολογία)	Olympia, Greece	39
Hanging Gardens of Babylon	Al Hillah, Iraq	unknown

8. Modify the first xsl:apply-templates statement as follows:

```
<xsl:apply-templates select="/ancient_wonders/wonder">
```

Question 2: *Test your change. Did the output change? Why or why not?*

***Answer:** No, the output did not change because the foreslash added on the XPath did not make any difference. It just indicated that the element should look for the child element to be styled in the root node.*

9. Modify the first xsl:apply-templates statement as follows:

```
<xsl:apply-templates select="/ancient_wonders/wonder  
[position()=last()]">
```

10. Test your change. Take a screen shot and paste it in your worksheet.

Seven Wonders of the Ancient World

Overview

Wonder Name	Location	Height
Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας)	Alexandria, Egypt	384

11. Add the following markup immediately beneath the </table> tag:

```
<h2>History</h2> <xsl:apply-  
templates  
select="ancient_wonders/wonder/history">  
    <xsl:sort select="year_built" order="descending" data-  
type="number" />  
</xsl:apply-templates>
```

***Question 3:** What is the <history> element's relationship to the <wonder> element?*

***Answer:** Wonder Element is the first node to be transformed in the block while <history> is the second element in the Xpath. The <xsl:apply-templates/> is a recursive call that will apply the rules to the current element.*

12. Add the following markup immediately before the `</xsl:stylesheet>` tag.

```
<xsl:template match="history">
  <a><xsl:attribute name="name"><xsl:value-of
select="../name[@language='English']"/></xsl:attribute></a>
The <xsl:value-of select="../name[@language='English']"/>
<xsl:apply-templates select="../name[@language!='English']"/>
was built in <xsl:value-of select="year_built"/><xsl:text>
</xsl:text><xsl:value-of select="year_built/@era"/>
  <xsl:choose>
    <xsl:when test="year_destroyed != 0">
      and was destroyed by <xsl:value-of select="how_destroyed"/>
in <xsl:value-of select="year_destroyed"/> <xsl:text>
</xsl:text><xsl:value-of select="year_destroyed/@era"/>.
    </xsl:when>
    <xsl:otherwise>
      and is still standing
today. </xsl:otherwise>
    </xsl:choose>
  <br /><br />
</xsl:template>
>
```

Question 4: Which node is the current node when the `year_built` attribute is being evaluated?

Answer: the current node when the `year_built` attribute is being evaluated will be "History".

Question 5: What is the relationship between the `<history>` element and the `<name>` element?

Answer: Name element is an attribute of the element `<history>` and, therefore, it gives more information about it.

13. Test your change and take a screen shot.

Seven Wonders of the Ancient World

Overview

Wonder Name	Location	Height
Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας)	Alexandria, Egypt	384

History

The Great Pyramid of Giza was built in 2570 BC and is still standing today.

The Hanging Gardens of Babylon was built in 600 BC and was destroyed by earthquake in 226 BC.

The Temple of Artemis at Ephesus (Ἀρτεμίσιον) was built in 550 BC and was destroyed by fire in 356 BC.

The Statue of Zeus at Olympia (Δίας μυθολογία) was built in 430 BC and was destroyed by fire in 426 AD.

The Mausoleum at Halicarnassus (Μανσωλεῖον Ἀλικαρνασσεύς) was built in 352 BC and was destroyed by earthquake in 1404 AD.

The Colossus of Rhodes (Κολοσσός της Ρόδου) was built in 282 BC and was destroyed by earthquake in 226 BC.

The Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας) was built in 280 BC and was destroyed by earthquake in 1323 AD.

Question 6: What XPath notation should you use to access the <main_image> element from the current context?

XPath: ancient_wonders/wonder/pyramid.jpg

14. Add markup to display the image associated with each wonder above the statement that describes it.

15. Change the Output directory to create the HTML file in the starter folder.

16. Test your change. Take a screen shot and paste it in your worksheet.

TIP: If the pictures don't appear when you browse in Visual Studio, open the HTML output file in a web browser.

Task 2: Using XPath Functions

1. Comment out the <xsl:apply-templates> markup you modified in Task 1.
2. Add a new <xsl:apply-templates> element that selects wonders that have a height greater than 100. Sort by height in descending order. Refer to the following figure:

```
***
<h2>Overview</h2>
<table border="1"><tr><th>Wonder
  Name</th><th>Location</th>
  <th>Height</th></tr>

  <xsl:apply-templates
    select="ancient_wonders/
      wonder[height > 100]">

    <xsl:sort select="height"
      order="descending"
      data-type="number" />
  </xsl:apply-templates>

</table>
***
```

3. Test your change. Take a screen shot and paste it in the worksheet.

Overview

Wonder Name	Location	Height
Great Pyramid of Giza	Giza, Egypt	455
Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας)	Alexandria, Egypt	384
Mausoleum at Halicarnassus (Μαυσωλεῖον Ἀλικαρνασσεύς)	Bodrum, Turkey	135
Colossus of Rhodes (Κολοσσός της Ρόδου)	Rhodes, Greece	107

4. Add a new column named Years Standing to the table in the Overview section.

5. Use the code given in the following figure as a guide to populate the column with data. Use the current year in your calculations for the wonders that have not been destroyed.

```
...
<td valign="top">
  <xsl:choose>
    <xsl:when test="
      history/year_destroyed != 0">

      <xsl:choose>
        <xsl:when test="
          history/year_destroyed/@era =
            'BC'">
          <xsl:value-of select="
            history/year_built -
            history/year_destroyed"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="
            history/year_built +
            history/year_destroyed - 1"/>
        </xsl:otherwise>
      </xsl:choose>

    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="
        history/year_built + 2008 - 1"/>
    </xsl:otherwise>
  </xsl:choose>
</td>
...
```

6. Test your change. Take a screen shot and paste it in your worksheet.
7. Modify the Height column so that it shows the height in both feet and meters. Show the height in meters using one decimal place. Refer to the following figures:

```
...
<xsl:when test="height != 0">
  <xsl:value-of select="
    height"/> feet<br />
  (<xsl:value-of select="
    height * 0.3048"/> m)
...
```

Wonders of the World - Windows Internet Explorer

Wonders of the World

Overview

Wonder Name	Location	Years Standing	Height
Great Pyramid of Giza	Giza, Egypt	4577	455 feet (138.684 m)
Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας)	Alexandria, Egypt	1602	384 feet (117.0432 m)
Mausoleum at Halicarnassus (Μαυσωλεῖον Ἀλικαρνασσεύς)	Bodrum, Turkey	1755	135 feet (41.148 m)

8. Test your change. Take a screen shot and paste it in your worksheet.

Overview

Wonder Name	Location	Height
Great Pyramid of Giza	Giza, Egypt	455feet (138.684 m)
Lighthouse of Alexandria (ὁ Φάρος τῆς Ἀλεξανδρείας)	Alexandria, Egypt	384feet (117.043200000000001 m)
Mausoleum at Halicarnassus (Μαυσωλεῖον Ἀλικαρνασσεύς)	Bodrum, Turkey	135feet (41.148 m)
Colossus of Rhodes (Κολοσσός της Ρόδου)	Rhodes, Greece	107feet (32.6136 m)

Question 7: Which function would you use to round the number to an integer value?

Answer: Round function is used to round the number to an integer value

9. Modify the table so that all seven wonders are displayed.

10. <Add markup immediately after the <xsl:apply-templates> markup. This sets the current context to ancient-wonders/wonder, which adds a row to the table displaying the average height of the wonders in feet. Refer to the following figure:

```

...
<tr>
<td valign="top" align="right"
  colspan="4">Average Height: </td>
<td valign="top">
<xsl:value-of select="
format-number(
  sum(/ancient_wonders/wonder/height)
div
  count(/ancient_wonders/wonder/
    height[.!=0]),
'##0.0') " />
ft</td>

```

Question 8: Why do you need to add the markup here?

11. Test your change. Use a calculator to verify that the value calculated is correct. Take a screen shot and paste it in your worksheet.

12. Add markup to modify the table so that the location is displayed in two separate columns—City and Country. Refer to the following figure:

```
...
<h2>Overview</h2>
<table border="1"><tr><th>Wonder Name
  </th><th>City</th><th>Country</th>
  <th>Years<br />Standing</th>
  <th>Height</th></tr>
...
<td valign="top">
  <xsl:value-of select="
    substring-before(location, ',')"/>
</td>
<td valign="top">
  <xsl:value-of select="
    substring-after(location, ',')"/>
</td>
...
```

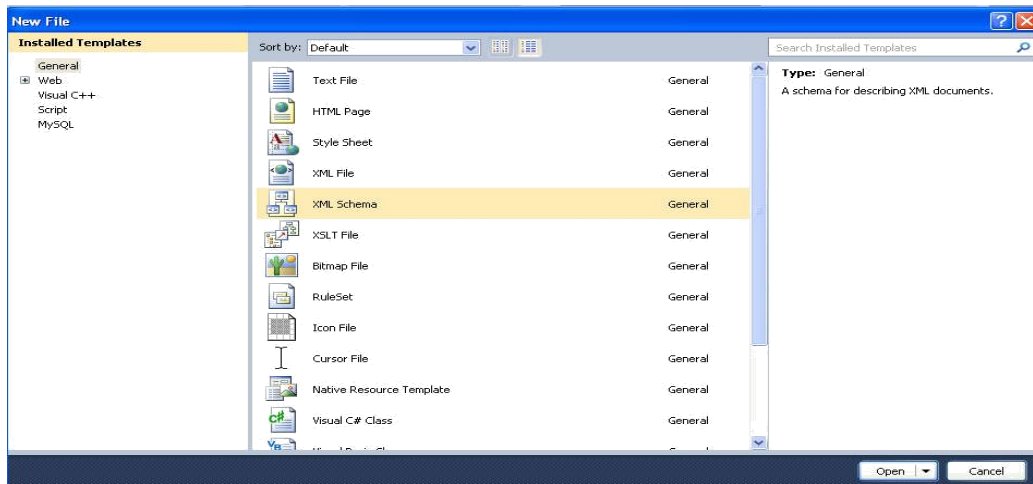
13. Test your change. Take a screen shot and paste it in your worksheet.

Question 9: Which function would you use to remove leading and trailing whitespace from a string?

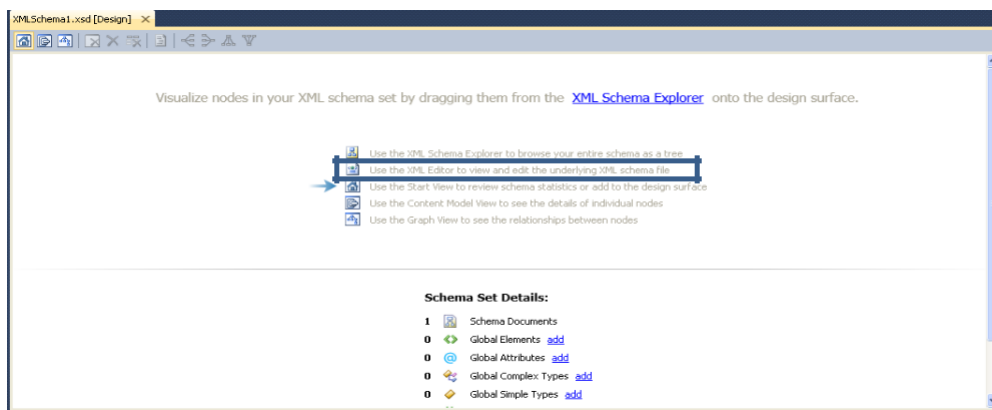
14. Submit the .xslt file along with your worksheet.

Task 1: Creating a Schema and Linking it to an XML

1. Launch Visual Studio.
2. Create a new **XML Schema** file.



3. Click **Use XML Editor** to view and edit the underlying schema.



4. Save it as nnLab5.xsd, where the letters *nn* stand for your initials.

Question 1: Which element is added by default?

Answer: The root element name is `xsi:noNamespaceSchemaLocation=".xsd"`. The value is the location of the xsd document.

5. Define an element named <wonder> that includes child elements named "name," "location," and "height" of type string. Refer to the following figure:

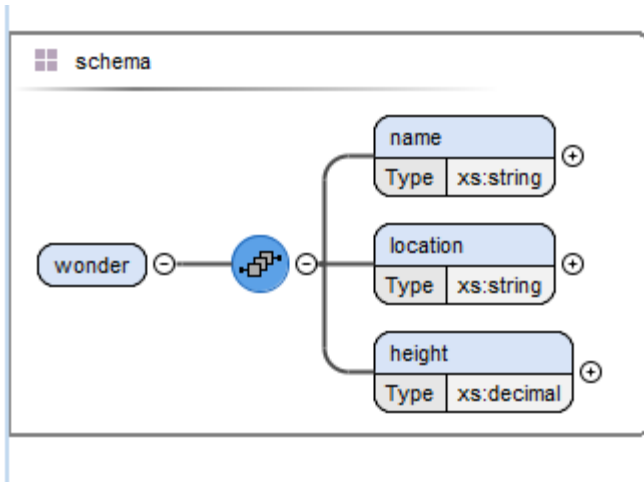
```
<?xml version="1.0"?>
<xs:schema xmlns:xs=
  "http://www.w3.org/2001/XMLSchema">

  <xs:element name="wonder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name"
          type="xs:string"/>
        <xs:element name="location"
          type="xs:string"/>
        <xs:element name="height"
          type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

6. Save your changes.
7. Open wonders_master.xml in your Lab 5 Files folder.
8. Modify the <wonder> element to reference the schema file you saved.
9. Add the following documentation to your schema (replace Student_Name with your name): This schema was created by Student_Name
10. Modify the <height> element to be type decimal.
11. Click **View | XML Schema** and expand the nodes. Take a screen shot and paste it in the worksheet.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="wonder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name"
          type="xs:string"/>
        <xs:element name="location"
          type="xs:string"/>
        <xs:element name="height"
          type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



Results of the schema as shown on Oxygen XML Editor

12. Save your changes.

Task 2: Defining the <history> Element

1. Define an element named <history> within the <wonder> element sequence.
2. Define <year_built> and <year_destroyed> elements of type integer. Use the <wonder> element as a model.
3. Define the <how_destroyed> element as type string. Set the default to fire. Refer to the following figure:

x s d
<pre><xs:element name="how_destroyed" type="xs:string" default="fire"/></pre>

4. Define a simpleType named story_type immediately after your </xs:annotation> tag. The type should be based on the xs:string type and have a maximum length of 1,024. Refer to the following figure:

x s d
<pre><xs:simpleType name="story_type"> <xs:restriction base="xs:string"> <xs:length value="1024"/> </xs:restriction> </xs:simpleType></pre>

5. Define an element named <story> of story_type as a child of the <history> element.

Question 2: What is the advantage of defining a named type?

Answer: the advantage of defining a named type is that although defined once, it can be reused as many times as possible in several schemas.

6. Take a screen shot of the <history> element's schema definition and paste it in your worksheet.

```
xs:schema
1  <?xml version="1.0" encoding="utf-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="wonder">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element name="name" type="xs:string"/>
7          <xs:element name="location" type="xs:string"/>
8          <xs:element name="height" type="xs:decimal"/>
9          <xs:element name="history" type="xs:decimal"/>
10
11         <xs:element name="year_built" type="xs:integer"/>
12         <xs:element name="year_destroyed" type="xs:integer"/>
13         <xs:element name="how_destroyed" type="xs:string" default="fire"/>
14       </xs:sequence>
15     </xs:complexType>
16   </xs:element>
17   <xs:simpleType name="story_type">
18     <xs:restriction base="xs:string"/>
19   </xs:simpleType>
20
21 </xs:schema>
22
```

Text Grid Author Design

Validation successful U+0009 17 :

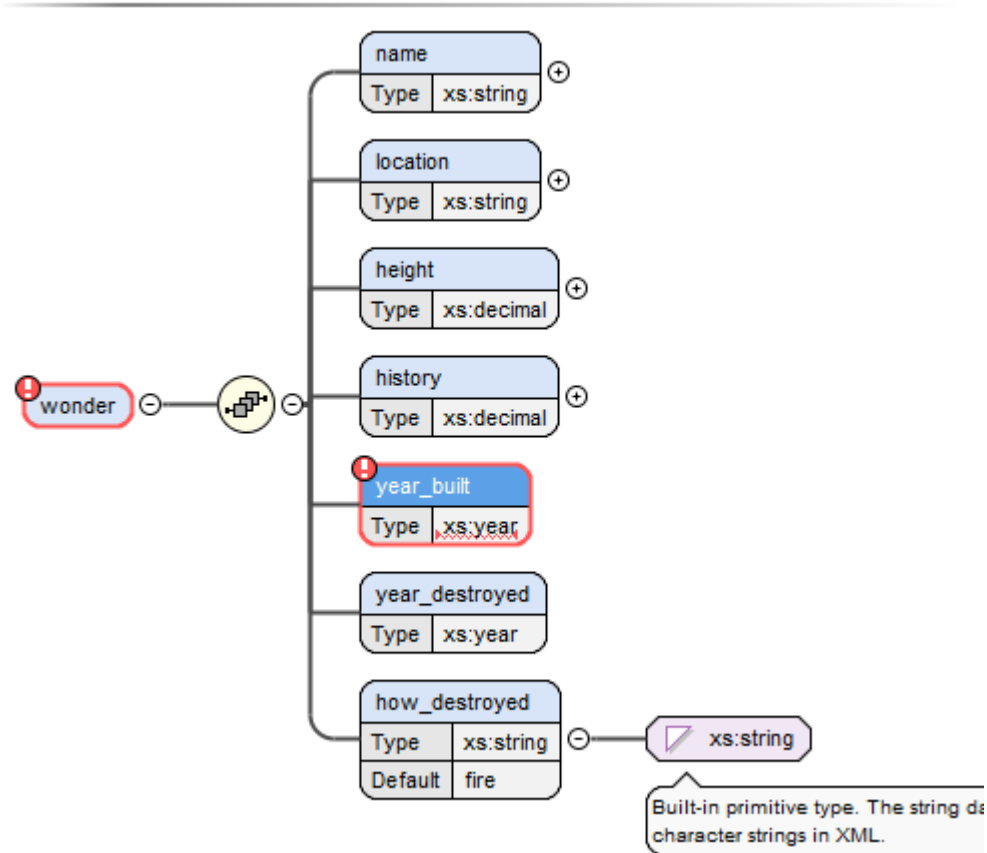
7. Define a simple type named `year_type` that is a whole number and has a minimum value of 0 and a maximum value of 5000.

Question 3: What did you use as the base type?

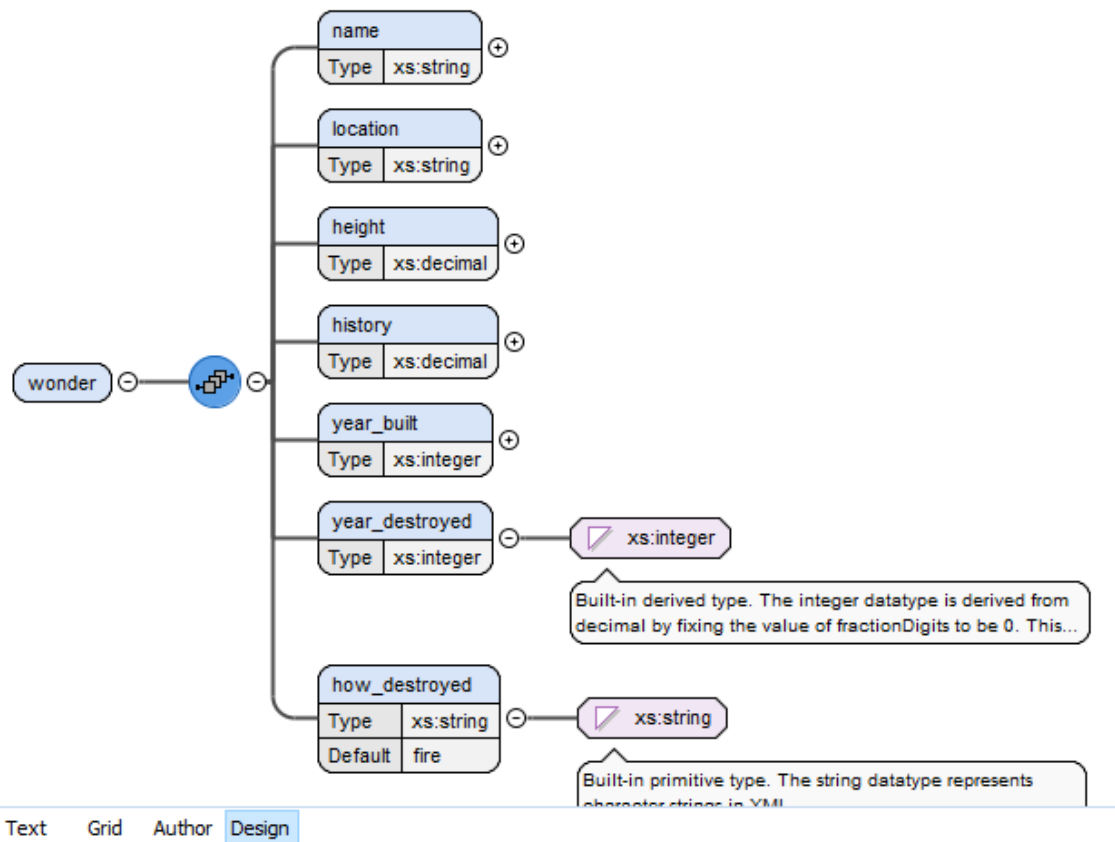
Answer: The base type I used was `SimpleType`

8. Modify the `<year_built>` and `<year_destroyed>` elements to be of type `year_type`.
9. Click **View | XML Schema** and expand the nodes. Take a screen shot and paste it in the worksheet.

After modifying the `<year_built>` and `<year_destroyed>` elements to be of type `year_type`, the schema could not validate the elements because they are not referencable by the schema document. Please see the screenshot below.



The screenshot provided below shows that when the type for the `year_built` are set to integers, the schema is working just fine.



10. Save your changes.

Task 3: Creating Complex Types

1. Save your file as `nnlab5.xsd`. Remember, you need to submit both the files for this lab.
2. Convert the definition for the `<wonder>` element to a complex type named `wonderType`.
3. Define a root element named `<ancient_wonders>` that contains a sequence of elements of type `wonderType`. Refer to the following figure:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="ancient_wonders">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="wonder"
          type="wonderType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

4. Convert year_type to a complex type. It should be defined as a positive integer that has an attribute named era of type string. Refer to the following figure:

x s d
<pre><xs:complexType name="yearType"> <xs:simpleContent> <xs:extension base= "xs:positiveInteger"> <xs:attribute name="era" type="xs:string"/> </xs:extension> </xs:simpleContent> </xs:complexType></pre>

5. Convert story_type to a complex type that can contain a mix of text and elements. Refer to the following figure:

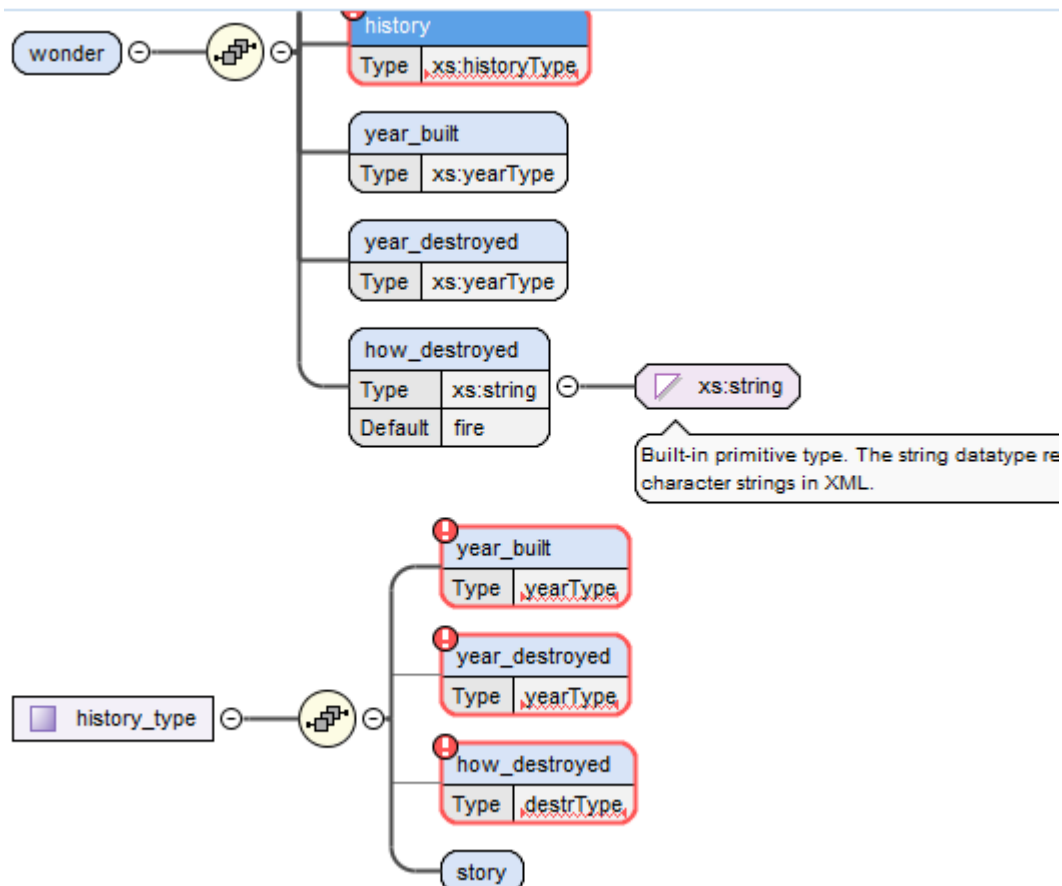
x s d
<pre><xs:complexType name="story" mixed="true"> <xs:sequence> <xs:element name="para" maxOccurs="unbounded"> <xs:complexType/> </xs:element> </xs:sequence> </xs:complexType></pre>

6. Create a complex type named `history_type` that contains `<year_built>` and `<story>` elements, which may or may not have a `year_destroyed` and `how_destroyed` child elements. Refer to the following figure:

```
xsd
<xs:complexType name="historyType">
  <xs:sequence>
    <xs:element name="year_built"
      type="yearType"/>
    <xs:element name="year_destroyed"
      type="yearType" minOccurs="0"/>
    <xs:element name="how_destroyed"
      type="destrType" minOccurs="0"/>
    <xs:element name="story"
      type="storyType"/>
  </xs:sequence>
</xs:complexType>
```

7. Modify the `<history>` element definition in the `wonderType` to be of type `history_type`.
8. Click **View | XML Schema** and expand the nodes. Take a screen shot and paste it in the worksheet.

There are so many errors in the schema because some of the components such as `yearType`, `destrType`, and `historyType` referred by the schema could not be found on the main document.



Exercise 3: JavaScript (5 Points)

Task 1: JavaScript Example

Sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code.

```
<!DOCTYPE html>
<html>
<body>
<h1 id="myH"></h1>
<p id="myP"></p>
<script>
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";
// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
<p><strong>Note:</strong> The comments are not executed.</p>
</body>
</html>
```

Task 2: JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows –

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
functionsayHello()
{ alert("Hello World") }
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello"
/> </body>
</html>
```

Task 3: JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<script type="text/javascript">
document.write("Hello World")
</script>
<p>This is web page body </p>
</body>
</html>
```

Task 4: Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

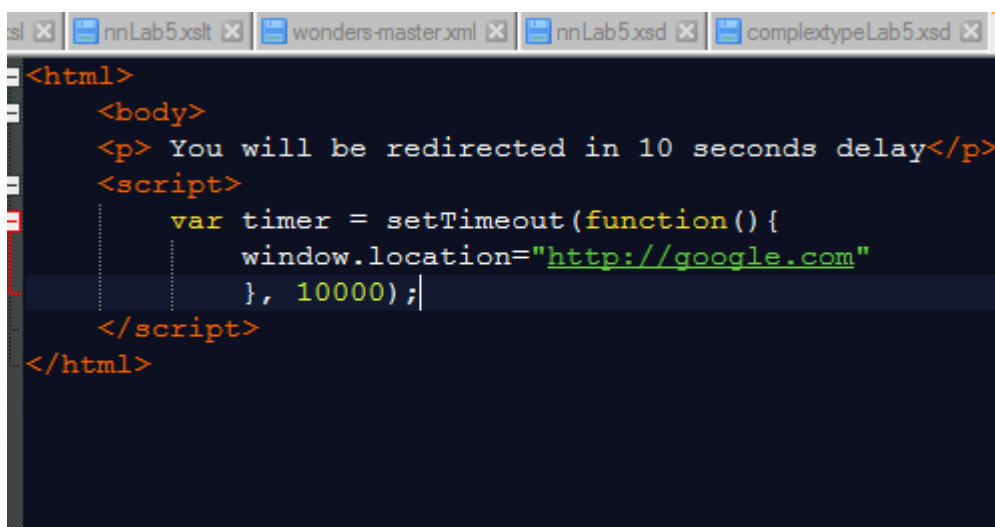
Example

Modified sayHello function here. Now it takes two parameters.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
functionsayHello(name, age)
{
document.write (name + " is " + age + " years old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say
Hello"> </form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Challenge Questions: (15 Points)

1. Write a JavaScript Redirect browser to different web page after 10 sec delay.

A screenshot of a web browser window with multiple tabs. The active tab shows a code editor with the following HTML and JavaScript code:

```
<html>
  <body>
    <p> You will be redirected in 10 seconds delay</p>
    <script>
      var timer = setTimeout(function() {
        window.location="http://google.com"
      }, 10000);
    </script>
  </html>
```

2. Write a JavaScript to create
 - a. Alert Dialog Box

```

<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
          document.write ("This is a warning message!");
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the button below to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick = "Warn();" />
    </form>
  </body>
</html>

```

b. Confirmation Dialog Box

```

<html>
  <head>
    <script type = "text/javascript">
      <!--
        function getConfirmation() {
          var retVal = confirm("Do you want to continue with this process?");
          if( retVal == true ) {
            document.write ("Continue!");
            return true;
          } else {
            document.write ("Exit!");
            return false;
          }
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick = "getConfirmation();" />
    </form>
  </body>
</html>

```

c. Prompt Dialog box

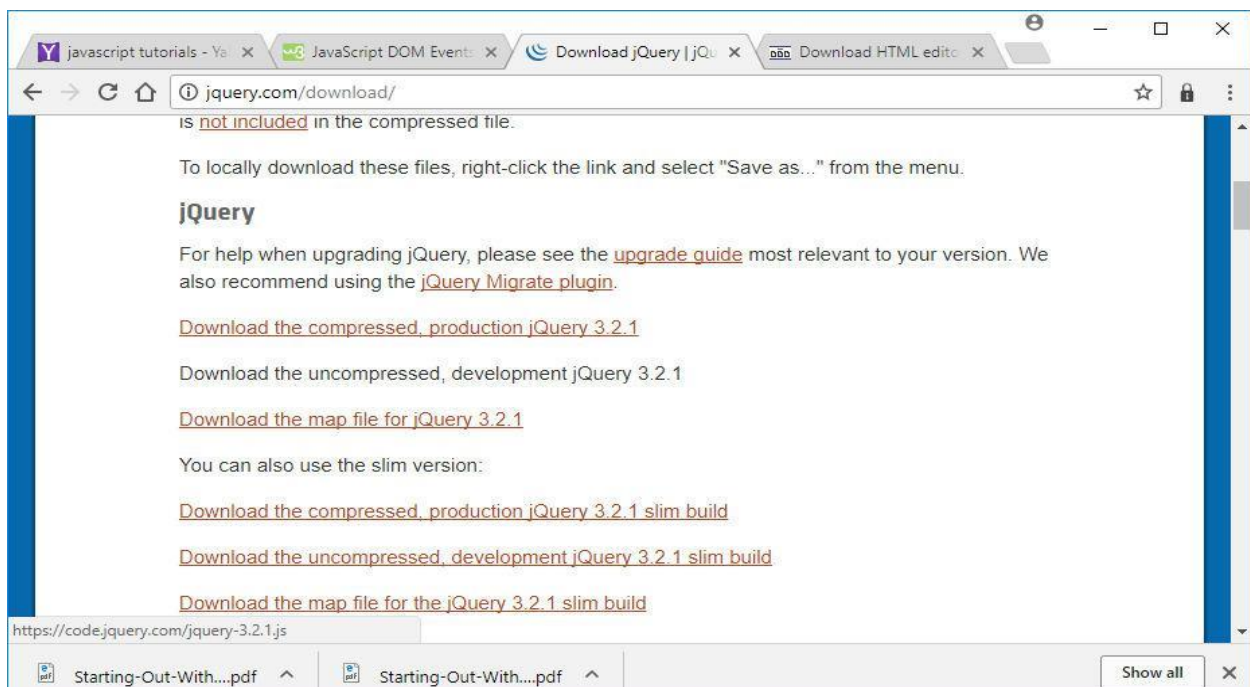
```
<html>
<head>
  <script type = "text/javascript">
    <!--
      function getValue() {
        var retVal = prompt("Enter your first name : ", "your name here");
        document.write("You have entered : " + retVal);
      }
    //-->
  </script>
</head>

<body>
  <p>Click the following button to see the result: </p>
  <form>
    <input type = "button" value = "Click Me" onclick = "getValue();" />
  </form>
</body>
</html>
```

3. Write a JavaScript to Change an image when a user holds down the mouse button.
4. Write a JavaScript to display all the cookies set on your machine.
5. Write a JavaScript to disable right click on the page

Exercise 4: JQuery (20 Points)

Task 1: Downloading jQuery

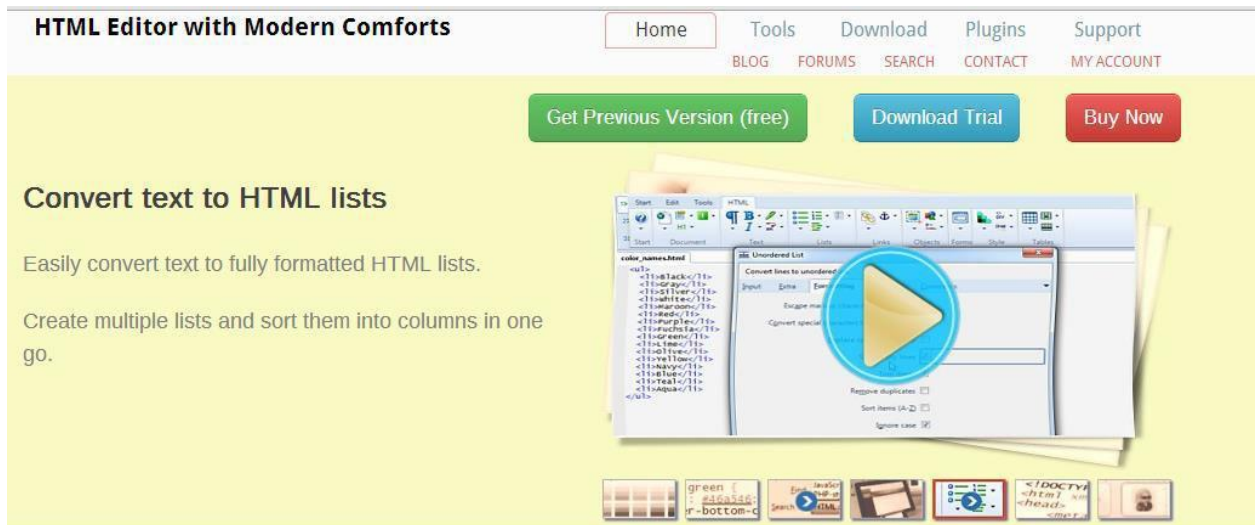


1. Right-click the link to download the uncompressed latest version of jQuery

2. Copy the `jQuery.x.x.x.js` file in the folder and specified as source file.

Task 2: Download and install HTML-Kit

1. Navigate to **htmlkit.com**.



2. Click **Download HTML-Kit 292**.

HTML-Kit 292 FREE

HTML-Kit 292, the version before **HTML-Kit Tools**, is available [for free](#) with all of its original features. No ads. No nags.

Find out why it has become a recognized [application](#) among [web developers](#) with millions of downloads and **300+** book mentions.

How did you hear about HTML-Kit?

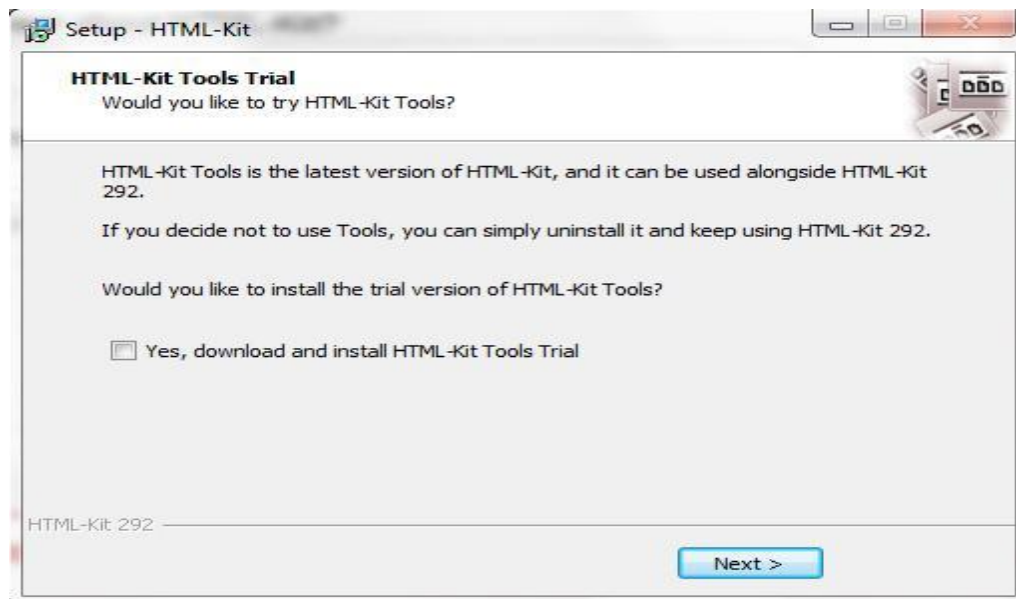
Stay up-to-date with update notices:

Email address (optional)

You can also follow HTML-Kit on [Twitter](#), [Facebook](#), and [Google+](#)

Runs on Windows, Linux with Wine, and OS X with virtual machine.
This setup includes **HTML Tidy** and **HTML5** updates posted 12-Jan-2012.

3. After it downloads, launch **HKSetup.exe**.
4. Choose **Full installation** (the default)
5. Uncheck **Yes, download and install HTML-Kit Tools Trial**.



6. Click **Next>Finish**.

Task 3: Creating a Simple jQuery Application

1. Launch HTML-Kit.
2. Create a new HTML file and save it as **nnLab8.htm**.
3. Add the following HTML to the file:

```
<!DOCTYPE HTML>
```

```

<html>
<head>
<title>Hello World - jQuery Style</title>
</head>
<body>
    <div id="first"></div>
    <div id="second"></div>
    <a href="#" id="link">Click Me!</a><br />
    <span id="greeting"></span>
</body>
</html>

```

4. Add the following `<script>` element to the `<head>` section. **NOTE:** Use the jQuery version number that matches the file name of the file you downloaded in Step 1.

```

<script type="text/javascript" SRC="JQUERY-X.X.X.JS"></script>

```

5. Add the following `<script>` element beneath the one you added in Step 4:

```

<script type="text/javascript">
$(document).ready(function() {
/*write 'Hello World! to the first div */
$('#first').html('<h1> Hello World!</h1>');
});
</script>

```

6. Preview in the browser window to verify that “Hello World!” displays. **TIP:** You might need to enable scripts to run on your browser.
7. Take a screenshot and paste it on your worksheet.
8. Add the following function beneath the selector you wrote in step 5:

```

/*a clickable 'Hello World!' example */
$('#link').click(function(){
$('#greeting').html('<h1>Hello
Again!</h1>'); });

```

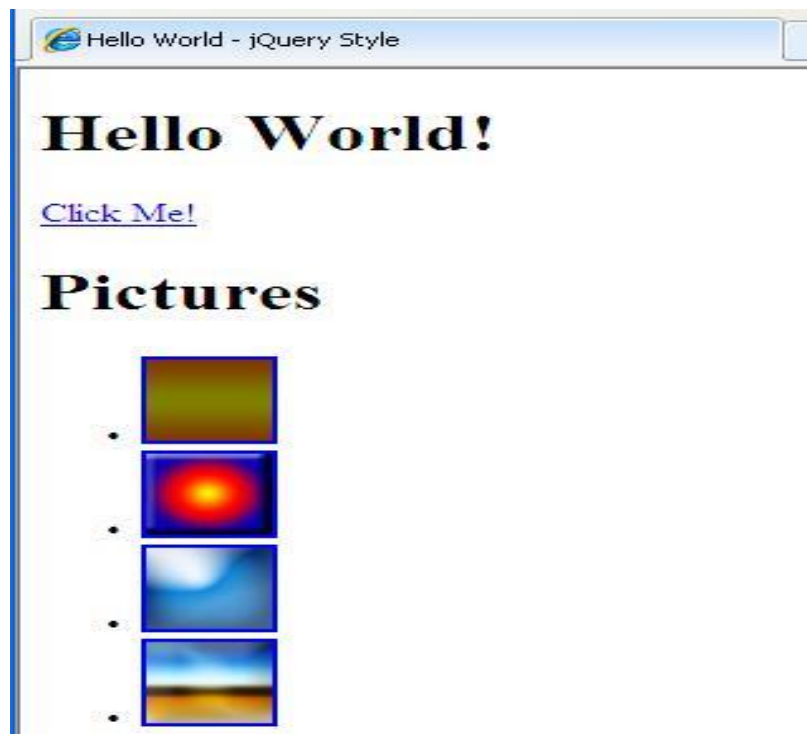
9. Test to verify that “Hello Again” displays when you click the link.
10. Take a screenshot and paste it on your worksheet.

Task 4: Navigating a List

1. Copy **Lab6pics.zip** to your virtual machine and extract the .jpg files to the folder you created in Task 1.
2. Add the following HTML beneath the HTML you added in Task 2:

```
<div>
    <h1>Pictures</h1>
    <ul>
        <li><a class="pic" href="#"></a></li>
        <li><a class="pic" href="#"></a></li>
        <li><a class="pic" href="#"></a></li>
        <li><a class="pic" href="#"></a></li>
    </ul>
    <div id="showCurrentPicName"> </div>
    <div id="showNextPicName"></div>
</div>
```

3. Preview in a browser to verify it looks like this.



4. Take a screenshot and paste it on your worksheet.

5. Add the following jQuery function:

```
/*Output the filename of the clicked picture*/
$('.pic').click(function(e) {
    var currentImage = $(this)
        .closest('li')
        .find('img')
        .attr('src');

    $('#showCurrentPicName').html('<h1>You clicked ' + currentImage
    + '</h1>');
});
```

6. Test your code and take a screenshot of the results of clicking each picture.

Task 5: Using jQuery to Validate an Email Address

1. Launch **HTML-Kit**.
2. Create a new HTML file.
3. Modify the **<head>** element as follows:
 - a. Add **“Registration Form”** as the title.
 - b. Add a **<script>** element that references the jQuery file.
 - c. Add an empty **<script>** element where you will place your jQuery script.
4. Add the following **<form>** element to the **<body>** section:

```
<form name="register" id="registerForm" action="submit"
method="post">
<label class="label" for="email">Email: </label>
<input type="text" name="email" id="email" size="48" />
<span class="error">please enter a valid email
address</span>
</form>
```

5. Preview in the browser.
6. Take a screenshot and paste it on the worksheet.
7. Add the following **<style>** element to the **<head>**section.

```
<style type="text/css">
.error {
display: none;
color: #FF0000;
```

```
font-size: 0.7em;
margin: 0px 0px0px 5px;
}
</style>
```

8. Add the following jQuery code:

```
$('#email').blur(function() {
var regexEmail= /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
var inputEmail = $(this).val();
var resultEmail = regexEmail.test(inputEmail);
if(!resultEmail) {
    $(this).next('.error').css('display', 'inline');
    }
    else {
        $(this).next('.error').css('display', 'none');
    }
    });
```

9. Test the code to verify that the following conditions exist. **TIP:** Test the code by viewing it in the default browser. Do not preview it in the browser window because Tab does not cause the blur event to occur.
 - a. The error displays when an invalid email is typed and you press **Tab**.
 - b. The error displays when nothing is typed and you press **Tab**.
 - c. The error does not display when you type a valid email address and press **Tab**.

Task 6: Using jQuery to Validate the Length of a Field

1. Add the following fields to the form:

```
<br /><br />
<label class="label" for="password">Password: </label>
<input type="text" name="password" id="password" size="48"
tabindex="1" />
<span class="error">please enter a valid password</span>
```

2. Write jQuery code that verifies that the password is not empty. The message should display when the form is loaded and should not be hidden until the user types in the field. Here are some tips:

3. Test to verify that the following conditions exist. Take a screenshot of each test and paste it on your worksheet.
 - a. The message should be displayed when the form is loaded.
 - b. Leaving the password field blank and clicking on the Email field causes the password error to remain displayed.
 - c. Typing a character and clicking away from the field causes the error to disappear.