Web Languages and Technologies

Faculdade de Engenharia da Universidade do Porto
12th January 2017

Duration: 2h / With Consultation

Name: _____

Number: _____

1. Consider the following HTML code:

```
1 <div class="widget">
2   <ul id="todo">
3     <li>Buy Bread</li>
4     <li>Learn Guitar</li>
5     <li class="important">Pay Bills</li>
6     <li class="postponed">Wash Car</li>
7   </ul>
8 </div>
```

And the following CSS code:

```
1 li:first-child {color: blue}            /* R1 */
2 div li {color: red}                     /* R2 */
3 div.widget ul#todo {color: cyan}        /* R3 */
4
5 div > ul#todo .important {color: green} /* R4 */
6 ul li.postponed {color: inherit}        /* R5 */
7 li + li + li {color: magenta}           /* R6 */
```

1½ val. (a) Calculate the specificity of each one of the rules (e.g. 0,2,2,1):

R1	R2	R3	R4	R5	R6
(0,0,1,1)	(0,0,0,2)	(0,1,1,2)	(0,1,1,2)	(0,0,1,2)	(0,0,0,3)

1 val. (b) Taking into consideration only the rules **R1 to R3**, indicate the color of each of the texts in the page:

Buy Bread	Learn Guitar	Pay Bills	Wash Car
blue	red	red	red

1 val. (c) Taking into consideration **all** the rules, indicate the color of each of the texts in the page:

Buy Bread	Learn Guitar	Pay Bills	Wash Car
blue	red	gree	cyan

2. Consider the following *string*:

   `Washing the washing machine while watching the washing machine washing washing`

   For each one of the regular expressions shown below, underline the **first** match:

½ val. (a) `/wa.*ing/`

   `Washing the washing machine while watching the washing machine washing washing`

½ val. (b) `/[a-z]{3}\b/`

   `Washing the washing machine while watching the washing machine washing washing`

½ val. (c) `/(ing).*?\1/`

   `Washing the washing machine while watching the washing machine washing washing`

½ val. (d) `/^.{3}/`

   `Washing the washing machine while watching the washing machine washing washing`

½ val. (e) `/(sh|ch)(?!ing)/`

   `Washing the washing machine while watching the washing machine washing washing`

½ val. (f) `/(.+?).*?\1/`

   `Washing the washing machine while watching the washing machine washing washing`

3. Consider the following HTML code excerpt:

```
1 <div id="products">
2   <ul>
3     <li>Apple: <span class="qty">3</span> <a href="#">+</a></li>
4     <li>Banana: <span class="qty">5</span> <a href="#">+</a></li>
5     <li>Pear: <span class="qty">6</span> <a href="#">+</a></li>
6   </ul>
7   <a href="#" class="buy">Buy</a>
8   <p class="total">0</a>
9 </div>
```

   Also consider that the complete page can have other *a*, *ul* and *li* elements. Write the *jQuery* code needed so that:

1 val. (a) When the *link* at the end of each list item is clicked, the quantity of that item is incremented by one.

```
function addEventListeners(){
   let plusButtons = document.querySelectorAll('div#products ul li a');
   plusButtons.forEach(element =>{
      element.addEventListener('click', addQty.bind(element));
   });
}

function addQty(){
   this.parentNode.querySelector('span.qty').innerText++;
}

addEventListeners();
```

Name: _____

Number: _____

2 val.

(b) When the *link* having a class *buy* is clicked, an array called *products*, containing a list of all products and their quantities, should be sent in an *Ajax POST* request to the address *calculatetotal.php*.

When the result of that request is received, the text of the paragraph *total* should be replaced by the received result (or by *not enough stock* if the received result is less than 0).

Example of the array to be sent:

```
[{"name":"Apple","qty":3},{"name":"Banana","qty":5},{"name":"Pear","qty":6}]
```

```
function buy(){
    let products = [];
    let productsList = document.querySelectorAll('div#products ul li');
    productsList.forEach(element =>{
        let product = new Object();
        product.name = element.innerText.match(/^(.*?)(?=:)/)[0];
        product.qty = element.querySelector('span.qty').innerText;
        products.push(product);
    });
    let request = new XMLHttpRequest;
    request.open('POST', 'calculatetotal.php');
    request.onload = function(data){
        let replyValue = data.target.response;
        if(replyValue < 0)
            document.querySelector('p.total').innerText = "not enough stock";
        else
            document.querySelector('p.total').innerText = replyValue;
    };
    request.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    request.send("products=" + JSON.stringify(products));
}
```

(Continues on the other side...)

4. Consider the following XML document:

```
<authors>
  <author country="Spain" name="Miguel de Cervantes">
    <book year="1605" type="Novel">Don Quixote</book>
  </author>
  <author country="England" name="William Shakespeare">
    <book year="1599" type="Tragedy">Hamlet</book>
    <book year="1606" type="Tragedy">Macbeth</book>
  </author>
  <author country="Russia" name="Leo Tolstoy">
    <book year="1865" type="Novel">War and Peace</book>
  </author>
  <author country="Portugal" name="Jose Saramago">
    <book year="1995" type="Novel">Ensaio sobre a Cegueira</book>
    <book year="1997" type="Novel">Todos os Nomes</book>
  </author>
</authors>
```

Consider that the context node is the document root. Write the XPath expressions that select the following elements:

½ val. (a) The name of all authors.

```
authors/author/@name
```

½ val. (b) The title of all books with type *Novel*.

```
//book[@type="Novel"]/text()
```

½ val. (c) The name of all authors that wrote more than one book.

```
authors/author[count(book)>1]/@name
```

1 val. (d) The country of origin of the author that wrote *Ensaio sobre a Cegueira*.

```
//book[text()="Ensaio sobre a Cegueira"]/parent::author/@country
```