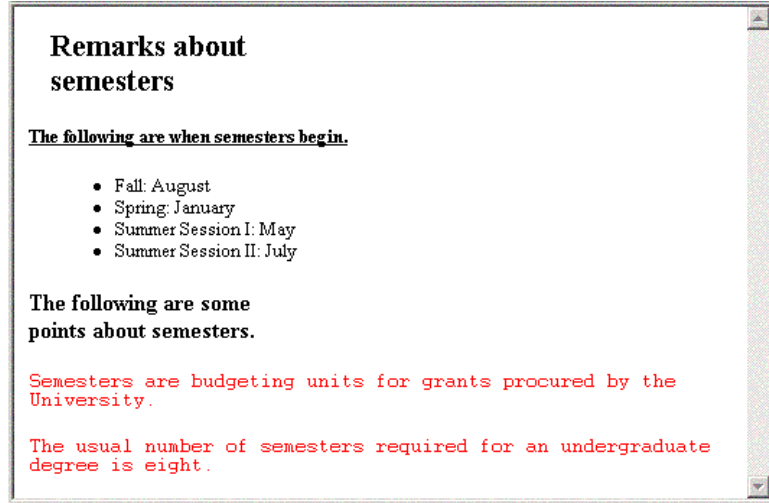


1 (4.5 pts.). Generally, what has been the fate of first movers in e-commerce? Why?

2 (4.5 pts.). What is time starvation and what is its relevance to B2C e-commerce?

3 (4.5 pts.). What is circuit switching? Why would it not be good infrastructure for the Internet?

4 (11 pts.). The figure shown here is the rendering of an HTML page that uses an external style sheet (file `styles.css`). The last two sentences are red with a Courier font. The HTML document is listed below with gaps labeled with Greek letters. The letters are repeated after the listing with descriptions of the missing code. You then supply the code. You also write the entire style sheet. The style sheet has the following three rules:



- h2 elements have a left margin of 15 pixels and occupy 40% of the width of the window.
- h3 elements are bold and occupy 40% of the width of the window.
- ul elements are 20 pixels to the right of their normal positions on the page.

The style sheet also defines the following classes:

- bold-under specifies underlined bold content.
- red1 specifies red with Courier font.

```
<html>
<head>
<title>Exam 1, Problem 4</title>
<link _α_____
_____
_____>
</head>
<body>
<h2>Remarks about semesters</h2>
<p _β_____>
  The following are when semesters begin.</p>
<ul>
  <li>Fall: August</li>
  <li>Spring: January</li>
  <li>Summer Session I: May</li>
  <li>Summer Session II: July</li>
</ul>
<h3>The following are some points about semesters.</h3>
<p _γ_____>Semesters are budgeting units for
  grants procured by the University.</p>
<p _γ_____>The usual number of semesters
  required for an undergraduate degree is eight.</p>
</body>
</html>
```

α (2 pts.): Fill in the attributes for the link element.

Answer

```
rel = "stylesheet"
type = "text/css"
href = "styles.css"
```

β (1 pt.): Apply the bold-underline class.

Note: In the previous page, the class was called bold-under. In the answer, either bold-under or bold-underline is acceptable.

Answer

```
class = "bold-under"
```

γ (1 pt.) (occurs twice): Apply the red1 class.

Answer

```
class = "red1"
```

Now write the external style sheet (7 pts.):

Answer

```
h2      { margin-left: 15; width: 40% }
h3      { width: 40%; font-weight: bold }
ul      { position: relative; left: 20 }
.bold-under { font-weight: bold; text-decoration: underline }
.red1   { font-family: Courier; color: red }
```

5 (14 pts.). The following .html file (with gaps) first declares and allocates for an array `scores` with three rows and five columns. This array is used for recording the grades of students on three exams (exam 1, exam 2, and exam 3) and their totals on all three. For a given row, the element at column 0 is the student's name, the elements at columns 1-3 are the scores on exams 1-3, respectively, and the element at column 4 is the sum of the elements at columns 1-3. The values for columns 0-3 are input, and the value in column 4 is computed. For simplicity, this is hard-coded to record only three students; a more practical implementation would add rows dynamically as students are added.

After allocating memory for array `scores`, the program passes `scores` to the function `inScores`, which prompts the user for names and exam scores. Then `scores` is passed to function `findTotal`, which records the total for each student in the last column. Next, `scores` is passed to function `sortScores`, which sorts the rows in array `scores` according to the names (column 0); this is in non-decreasing lexicographical order. Finally, `scores` is passed to function `tableTotals`, which produces a table of names and total scores from the sorted array. For example, suppose the user enters the following values in the order shown. Then the table produced is as shown at right.

Name	exam 1	exam 2	exam 3
Fred	8	7	9
Sue	6	7	8
Bill	9	9	8

Total Score	
Name	Score
Bill	26
Fred	24
Sue	21

The document is listed below. Missing fragments are labeled with Greek letters. These letters are repeated later with a description of what is missing. You there supply the missing code.

```

<html>
<head>
<title>Exam 1, Problem 4</title>
<script>
  α _____;

  _____;

  inScores( scores );
  findTotal( scores );
  sortScores( scores );
  tableTotals( scores );

```

Continued


```
function tableTotals( scores )
{
  document.writeln( "<table border='1'>" );
  document.writeln( "<caption>Total Score</caption>" );
  document.writeln( "<thead>" );
  document.writeln( "<tr><th>Name</th><th>Score</th></tr>" );
  document.writeln( "</thead>" );
  document.writeln( "<tbody>" );

  for ( var i in scores )
    document.writeln( _____
                      _____ );

  document.writeln( "</tbody>" );
  document.writeln( "</table>" );
}
</script>
</head>
<body>
</body>
</html>
```

α (4 pts.) (3 lines in the listing): Declare and allocate for a two-dimensional array scores with three rows and five columns.

Answer

```
var scores = new Array(3);

for ( var i=0; i < scores.length; i++ )
  scores[i] = new Array(5);
```

β (6 pts.): Fill in the body of the `sortScores` function. This uses an insertion sort to sort the rows of scores in non-decreasing lexicographical order of the contents in their first columns (column index 0). In case you have forgotten insertion sort, the following is a JavaScript function that uses an insertion sort to sort an array `ar` of numbers. (Note that JavaScript uses the usual relational operators (`==`, `<`, `>`, etc.) for comparing strings.)

```
function sortNums( ar )
{
  for ( var i=1; i < ar.length; i++ ) {
    var j = i-1,
        temp = ar[i];

    while ( j >= 0 && ar[j] > temp )
      ar[j+1] = ar[j--];

    ar[j+1] = temp;
  }
}
```

Answer

```
for ( var i=1; i < scores.length; i++ ) {
  var j = i-1,
      temp = scores[i];

  while ( j >= 0 && scores[j][0] > temp[0] )
    scores[j+1] = scores[j--];

  scores[j+1] = temp;
}
```

I have underlined that parts that have changed (other than replacing the name `ar` with the name `scores`).

γ (2 lines in the listing): Write the argument to the `document.writeln` statement that outputs a row of the table. This is in a loop that runs the control variable `i` over the rows of `scores`. The table row has two cells, a head cell and a body cell. The content of the head cell is the student's name in the current row (`i`) of `scores`, and the content of the body cell is the total for that student.

Answer

```
"<tr><th>" + scores[i][0] + "</th>"
+ "<td>" + scores[i][4] + "</td></tr>"
```

6 (11.5 pts.). The JavaScript file listed below (with gaps) consists mostly of the definition of the function `start()`. This function uses a prompt box to input what is supposed to be a time period expressed as two times, both expressed in minutes and seconds, separated by a hyphen (-). (This indicates the starting and ending times of the period.) It checks that the input is in this pattern and, if so, that the numbers supplied are valid, that is, that the numbers are all less than 60 (the pattern excludes negative numbers), and that the first time is not later than the second time. If the input does not match the pattern, then an alert box stating **Invalid pattern** is raised; if the pattern is matched but (some of) the numbers are invalid, an alert box stating **Invalid values** is raised; and, if the input is valid, an alert box stating **OK** is raised.

In more detail, the pattern for the input is `mm:ss-mm:ss`, that is, two substrings of the form `mm:ss` (one for the start time, one for the end time) separated by a hyphen. Each substring `mm:ss` consists of two digits (for the minutes), a colon (:), then two more digits (for the seconds). We allow any amount of white space before or after the entire pattern and immediately before or immediately after the hyphen. No other characters, however, are allowed before or after the pattern, that is, the entire input string must match the pattern. The following are some valid strings. (Each could be preceded or followed by any amount of white space).

```
01:23-01:31
25:07 - 32:03
```

The following are some strings that do not fit the pattern along with the reasons they do not.

```
1:23-1:31           Each minutes field must have exactly two digits.
25:7 - 32:3         Each seconds field must have exactly two digits
25 : 17 - 26 : 20   No white space is allowed on either side of a colon.
c. 01:23-01:31     No characters besides white space are allowed before the period.
```

Once it is verified that the input matches the pattern, the four numbers are passed (in the order of their appearance in the time period) to function `isValid()`, which is defined (without gaps) in this file. For example, if `12:45-13:15` is input, then the four actual parameters in the call to `isValid()` are 12, 45, 13, and 15 in that order.

The document is listed below. Missing fragments are labeled with Greek letters. These letters are repeated later with a description of what is missing. You there supply the missing code.

```

function start()
{
  var
    reg = α_____,
    str = window.prompt(
      "Enter the period in the format mm:ss-mm:ss.",
      "00:00-59:59" );

  if ( str.match( reg ) )
    if ( β_____ )
      window.alert( "OK" );
    else
      window.alert( "Invalid values" );
  else
    window.alert( "Invalid format" );
}

function isValid( mins1, secs1, mins2, secs2 )
{
  var
    inRange = mins1 < 60 && secs1 < 60 && mins2 < 60 && secs2 < 60,
    ordered = mins1 < mins2 || mins1 == mins2 && secs1 <= secs2;

  return inRange && ordered;
}

```

α (7.5 pts.): A regular expression the pattern for the input is `mm:ss-mm:ss`, that is, two substrings of the form `mm:ss` (one for the start time, one for the end time) separated by a hyphen. Each substring `mm:ss` consists of two digits (for the minutes), a colon (:), then two more digits (for the seconds). We allow any amount of white space before or after the entire pattern and immediately before or immediately after the hyphen. No other characters, however, are allowed before or after the pattern, that is, the entire string must match the pattern. (Hint: You need anchors at the beginning and end.) Each of the four two-digit numbers must be remembered for later use. (Note that the regular expression does not determine whether the numbers are less than 60 or whether the first period is no later than the second.)

Answer

```

/^\

```

β (4 pts.): Invoke `isValid()` passing it the four numbers remembered from the match. These occur in the call in the same order in which they occur in the pattern.

Answer

```

isValid( RegExp.$1, RegExp.$2, RegExp.$3, RegExp.$4 )

```