

## COMP 322 Internet Systems Fall 2007 Assignment 4

Due in the digital drop box by Wednesday, Oct. 17 at 11:00 PM

1. The following code (with fragments missing) prompts the user for his/her name and age. It then attempts to match the string input from the prompt against a regular expression that defines the legal input. If this match succeeds, it echoes back the name and age entered. Otherwise, it flags the input as illegal. The regular expression does not distinguish between upper- and lowercase letters. It defines the following pattern:

- the beginning-of-string anchor (so that the user can't enter superfluous characters at the beginning),
- the sequence of letters (upper- or lowercase) 'n', 'a', 'm', 'e',
- a colon (':'),
- zero or more spaces or a tab (`\t`),
- one or more word characters (remembered for backreference),
- a semicolon (';'),
- zero or more spaces or a tab (`\t`),
- the sequence of letters (upper- or lowercase) 'a', 'g', 'e',
- a colon (':'),
- zero or more spaces or a tab (`\t`),
- a sequence of one to three digits (remembered for backreference), then
- the end-of-string anchor (so that the user can't enter superfluous characters at the end).

The following are examples of valid input:

```
name: Bob; age: 21
name: Fred; age: 33
```

Since the match is case insensitive, the following are also valid input:

```
Name: Bob; Age: 21
NAME: bOB; aGe: 21
```

When you echo back the name and age, you'll use two of the implicit variables `RegExp.$1` to `RegExp.$9`. Recall that these have the values that were matched to the parts of the regular expression that were enclosed in parentheses. The thing to watch out for here is that you will have to use parentheses also to group alternatives (the operands of `|`). Either you count parenthesized expressions when you determine which of these implicit variables to use, or you make use of the following detail about regular expression syntax. To match a substring without causing the matched part to be remembered, within the parentheses, preface the pattern with `?:`. For example, `(?:\d+)` matches one or more numeric characters but does not remember the matched characters.

Each missing fragment is labeled with a Greek letter. Each Greek letter is repeated after the code with a description of the missing code at the indicated position. This file (with the fragments missing) is available on the Web page where you got this assignment.

```
<html>
<head>
<title>Assignment 4, Problem 1</title>
<script type = "text/javascript">
  var str,
      reg = α_____ ;

  str = prompt( "Enter your first name and age in the format\n"
              + "Name: FirstName; Age: 99",
              "Name: FirstName; Age: 99" );
  if ( str.match( reg ) )
    document.write( "<p>Your name is " + β_____
                  + " and your age is " + γ_____ + "</p>" );
  else
    document.write( "<p>Illegal input</p>" );
</script>
</head>
<body>
</body>
</html>
```

$\alpha$ : The required regular expression

$\beta$ : The implicit variable (one of `RegExp.$1` to `RegExp.$9`) that matches the user's name.

$\gamma$ : The implicit variable that matches the user's age.

2. This is an exercise in using the DOM and will not work with Internet Explorer; you should probably test it out using Mozilla/FireFox. Produce a page that does the following. When the page is loaded, the text element **Falling**, in blue, starts in the upper left-hand corner. It moves down at a constant speed of five pixels every tenth of a second. When it has moved 100 pixels down, its color changes to red. When the text element gets 200 pixels down, it jumps 100 pixels to the right, changes to **Rising**, but remains red, and moves up at the same constant speed with which it moved down. When it gets to 100 pixels from the top, its color changes back to blue. When the text element reaches the top, it changes back to **Falling** and jumps back to the left-hand corner, and the whole process repeats. This behavior repeats indefinitely. In addition, when the moving text is clicked, a bulleted list item is added to the bottom of the page (say, 210 pixels from the top, to stay out of the way of the moving text) that gives the coordinates of the part of the browser window that was clicked (hence indicated where the text was when it was clicked). The screen shot at right shows the text as it moves up on the right and is less than 100 from the top. The text has been clicked twice: once while moving down, approaching the midpoint, and once while moving up, just after passing the midpoint.



This is similar to the example using `window.setInterval()` on pp. 144-5 of the Course Notes but has some additional features (and lacks some of the routine features of that example). You should put an empty `ul` element at the bottom of the page—use absolute positioning—and give it an `id` attribute. Add an event listener for this `ul` element, say, `hit()`, that uses `document.createElement()` to create a `li` element. Assign a string to the `innerHTML` property of this `li` element constructed using the values of the appropriate properties of the `Event` object passed to `hit()`, and append the `li` element as a child of the `ul` element.

The following is a solution for this problem, but there are gaps in it, each identified by a Greek letter. After the listing, the letters are repeated along with descriptions of the missing code. You may download this file (with the gaps) at the same site where you accessed this assignment. .

```

<html>

<head>
<title>Assignment 4, Problem 2</title>

<script type = "text/javascript">
  var speed = 5;
  var count = 0;
  var pObj;
  var ulObj;

  function start()
  {
    pObj = α _____;
    pObj.β _____;
    ulObj = γ _____;
    δ _____;
  }

  function run()
  {
    count += speed;

    if ( count == 100 )
      ε _____ =
      ( speed < 0 ) ? "blue" : "red" ;

    if ( ( count % 200 ) == 0 ) {
      speed *= -1;

      ζ _____ =
      _____;
      η _____ =
      _____;
    }

    θ _____;
  }

  function hit( event )
  {
    var liObj = ι _____;

    liObj.innerHTML =
      κ _____;

    λ _____;
  }
</script>
</head>

```

Continued

Continued from previous page

```
<body onload="start()">
<p id = "pText"
  style=_u_____>
Falling</p>

<ul id="list" style=_v_____>
</ul>

</body>
</html>
```

- $\alpha$ : Get a reference to the object corresponding to the `p` element with `id="pText"`.
- $\beta$ : Make the function `hit` a listener for `click` events for object `pObj`.
- $\gamma$ : Get a reference to the object corresponding to the `ul` element with `id="list"`.
- $\delta$ : Have the function `run` executed every tenth of a second.
- $\epsilon$ : The CSS property of the `p` element (accessed by `pObj`) that controls its color
- $\zeta$  (2 lines in the code): Assign 100 or 0 (depending on the sign of `speed`) to the CSS property of the `p` element that controls its offset from the left margin. (Use a conditional expression.)
- $\eta$ : Assign "Rising" or "Falling" (depending on the sign of `speed`) to the attribute of the `p` element that determines the text (possibly with formatting tags) rendered for the element. (Use a conditional expression.)
- $\theta$ : Assign the value of `count` to the CSS property of the `p` element that controls its offset from the top margin
- $\iota$ : Create an `li` element.
- $\kappa$ : A string that consists of an opening parenthesis, the X coordinate of the mouse cursor in the browser window when the `click` event happened, a comma, a space, the Y coordinate of the cursor, then a closing parenthesis.
- $\lambda$ : Make the `li` object the new last child of the `ul` object.
- $\mu$ : Use absolute positioning. Position this element in the upper left hand corner of the window. Make its color blue.
- $\nu$ : Again use absolute positioning. Position this element 210 pixels from the top of the window.