

JAVASCRIPT PRACTICAL REVISION QUESTIONS

Term: one

SECTION I. Functions

1. Write a JavaScript function that reverse a number.

Example x = 32243;

Expected Output : 34223

2. Write a JavaScript function that checks whether a passed string is palindrome or not?

A palindrome is word, phrase, or sequence that reads the same backward as forward, e.g., madam or nursesrun.

3. Write a JavaScript function that generates all combinations of a string

Example string : 'dog'

Expected Output : d,do,dog,o,og,g

4. Write a JavaScript function that returns a passed string with letters in alphabetical order.

Example string : 'webmaster'

Expected Output : 'abeemrstw'

Assume punctuation and numbers symbols are not included in the passed string.

5. Write a JavaScript function that accepts a string as a parameter and converts the first letter of each word of the string in upper case.

Example string : 'the quick brown fox'

Expected Output : 'The Quick Brown Fox '

6. Write a JavaScript function that accepts a string as a parameter and find the longest word within the string.

Example string : 'Web Development Tutorial'

Expected Output : 'Development'

7. Write a JavaScript function that accepts a string as a parameter and counts the number of vowels within the string.

Note : As the letter 'y' can be regarded as both a vowel and a consonant, we do not count 'y' as vowel

here.

Example string : 'The quick brown fox'

Expected Output : 5

8. Write a JavaScript function which will take an array of numbers stored and find the second lowest and second greatest numbers, respectively.

Sample array : [1,2,3,4,5]

Expected Output : 2,4

9. Write a JavaScript function to compute the factors of a positive integer.

10. Write a JavaScript function to compute the value of b^n where n is the exponent and b is the bases. Accept b and n from the user and display the result.

11. Write a JavaScript function to extract unique characters from a string. *Example string* :

"thequickbrownfoxjumpsoverthelazydog"

Expected Output : "thequickbrownfxjimpsvlazydg"

12. Write a JavaScript function to get the number of occurrences of each letter in specified string.

Example String: The meeting is done

Example out: T=2,h=1,e=4,....

13. Write a function for searching JavaScript arrays with a binary search.

Note : A binary search searches by splitting an array into smaller and smaller chunks until it finds the desired value.

14. Write a JavaScript function that accepts two arguments, a string and a letter and the function will count the number of occurrences of the specified letter within the string.

Sample arguments : 'rwanda coding academy', 'a'

Expected output : 4

15. Write a JavaScript function to find the first not repeated character.

Sample arguments : 'abacddbec'

Expected output : 'e'

16. Write a JavaScript function to apply Bubble Sort algorithm.

Note : According to wikipedia "Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order".

Sample array : [12, 345, 4, 546, 122, 84, 98, 64, 9, 1, 3223, 455, 23, 234, 213]

Expected output : [3223, 546, 455, 345, 234, 213, 122, 98, 84, 64, 23, 12, 9, 4, 1]

17. Write a JavaScript function that accept a list of country names as input and returns the longest country name as output.

Sample function : Longest_Country_Name(["Australia", "Germany", "United States of America"])

Expected output : "United States of America" .

18. Write a JavaScript program to pass a 'JavaScript function' as parameter.

19. Write a JavaScript function to get the function name.

SECTION II. JAVASCRIPT DOM

20. Write the JavaScript code to add behavior to the following page for manipulating strings. The page UI allows the user to type a phrase into a text box. The user can click the "Go!" button to display the words in that phrase in reverse order. Each word in the phrase should be inserted as a span with a class of word, inside a div with the id of words. (The word class gives the spans their appearance in the screenshots below, giving them a border and a background and so on.) Every other word (the first, third, fifth, etc.) should also be underlined. The user can optionally specify a "filter" text by typing into a text box with the id of filter. If a non-blank filter is specified, you should exclude any words from the phrase that contain that filter text, case-insensitively. For example, if the filter text is "abc", exclude any words containing abc, ABC, aBc, etc. If any words are excluded, under the list of words you should modify the div with id of count to display text of the form, "5 word(s) filtered out". The code should work for multiple clicks of the button. On each click it should clear any previous information you injected. You may assume that words in the phrase are separated by single spaces. Do not use any JavaScript libraries such as jQuery or Prototype. Here is the relevant HTML code for the page:

These screenshots show the initial state, and after phrases have been typed and "Go!" is clicked



21. Write the JavaScript code to add behavior to the following page for finding palindromes. A palindrome is a word that is spelled the same forward as backward, such as "madam" or "Anna". The page UI allows the user to type a phrase into a text box. The user can click a "Find Palindromes" button to find palindrome words in that phrase. Match case-insensitively; for example, "rotOR" is a palindrome. You may assume that words in the phrase are separated by single spaces and contain only letters. A one-letter word such as "I" is defined to be a palindrome. Each palindrome found should be inserted as a bullet into a list with the id of palindromes. Every other palindrome (the first, third, fifth, etc.) should be given a

gray background color of #CC0000. Underneath the list of palindromes you should display text such as "5 total palindrome(s)" in the div with id of count. The user can optionally specify a minimum and maximum word length by typing integer values into two text boxes with id of min and max respectively. If a minimum is specified, you should include only palindrome words that contain at least that many letters inclusive. If a maximum is specified, you should include only palindrome words that contain at most that many letters inclusive. If the min or max is left blank, the length is unbounded in that direction. For example, a minimum of 3 and a blank maximum finds all palindromes of at least 3 letters. You may assume that the text typed in these boxes will either be blank or a valid non-negative integer, and that max will be \geq min. The code should work for multiple clicks of the button. On each click it should clear any previous found information. You may assume that Prototype and Scriptaculous are included in the page.

These screenshots show the initial state, and after phrases have been typed and "Find Palindromes" is clicked

Palindrome Finder!

Phrase:

Length: to

Palindrome Finder!

Phrase:

Length: to

- I
- did
- a

3 total palindrome(s).

Palindrome Finder!

Phrase:

Length: to

- Madam
- sTats
- SexEs
- CIVIC
- raceCar

5 total palindrome(s).

Palindrome Finder!

Phrase:

Length: to

- Madam
- did
- sTats
- SexEs
- CIVIC

5 total palindrome(s).

22. Write the JavaScript code to add behavior to the following page that has a user interface for entering grades on homework assignments. You will compute the percentage of points earned, with an optional curve. When "Compute!" is clicked, your JS code should use the values in the text boxes to compute the percentage (rounded to the nearest percent). If the "Curve +5" checkbox is checked, add +5 percent up to

a maximum of 100% total. You should insert the percentage into the page as a new div added to the end (bottom) of the existing page section with the id of resultsarea. If the overall percentage is 60% or more, give your newly created div a CSS class of pass; otherwise give it a class of fail. Each time the user clicks "Compute!", you will insert such a new div; this means that several divs would be there after several clicks of "Compute!". In the code shown there are 3 assignments, but your code should work for any number of assignments ≥ 1 . When "Clear" is clicked, all text in all of the input text boxes should be erased. Assume valid input; that is, assume that when "Compute!" is clicked, the user will have already typed valid text into every box that can be interpreted as an integer. You may assume that Prototype is also included in the page.

HTML:	CSS:
<pre> <h1>Grade Calculator</h1> <div id="assignments"> <div class="hw"> HW1 <input class="earned" type="text" size="2" /> / <input class="max" type="text" size="2" /> </div> <div class="hw"> HW2 <input class="earned" type="text" size="2" /> / <input class="max" type="text" size="2" /> </div> <div class="hw"> HW3 <input class="earned" type="text" size="2" /> / <input class="max" type="text" size="2" /> </div> ... </div> <div> <label><input id="curve" type="checkbox" /> Curve +5 ?</label> </div> <div id="resultsarea"> <button id="compute">Compute!</button> <button id="clear">Clear</button> <!-- your percentages should be inserted here --> </div> </pre>	<pre> .pass { background-color: #cfc; font-weight: bold; } .fail { background-color: #fcc; font-style: italic; } </pre>

These screenshots show the initial state, and state after scores have been typed and "Compute!" has been clicked

Grade Calculator	Grade Calculator	Grade Calculator	Grade Calculator
HW <input type="text"/> / <input type="text"/> HW <input type="text"/> / <input type="text"/> HW <input type="text"/> / <input type="text"/> <input type="checkbox"/> Curve +5 ? <input type="button" value="Compute!"/> <input type="button" value="Clear"/>	HW <input type="text" value="9"/> / <input type="text" value="10"/> HW <input type="text" value="7"/> / <input type="text" value="10"/> HW <input type="text" value="16"/> / <input type="text" value="20"/> <input type="checkbox"/> Curve +5 ? <input type="button" value="Compute!"/> <input type="button" value="Clear"/> <div style="background-color: #cfc; padding: 2px; margin-top: 5px;">80</div>	HW <input type="text" value="9"/> / <input type="text" value="10"/> HW <input type="text" value="7"/> / <input type="text" value="10"/> HW <input type="text" value="16"/> / <input type="text" value="20"/> <input checked="" type="checkbox"/> Curve +5 ? <input type="button" value="Compute!"/> <input type="button" value="Clear"/> <div style="background-color: #cfc; padding: 2px; margin-top: 5px;">80</div> <div style="background-color: #cfc; padding: 2px; margin-top: 5px;">85</div>	HW <input type="text" value="5"/> / <input type="text" value="10"/> HW <input type="text" value="11"/> / <input type="text" value="15"/> HW <input type="text" value="11"/> / <input type="text" value="21"/> <input type="checkbox"/> Curve +5 ? <input type="button" value="Compute!"/> <input type="button" value="Clear"/> <div style="background-color: #cfc; padding: 2px; margin-top: 5px;">80</div> <div style="background-color: #cfc; padding: 2px; margin-top: 5px;">85</div> <div style="background-color: #fcc; padding: 2px; margin-top: 5px;">59</div>

23. Write the Javascript code to add behavior to the following HTML code. The page is similar to the number guessing game assigned in CSE 142. When the page loads, your code should randomly choose a

number from 1-100 inclusive. (Recall that the `Math.random` function returns a random real number between 0 and 1.) When the user types a number into the number text field and then clicks the `makeguess` button, the game will compare the user's guess to your randomly chosen number, and report whether it was "Too high!", "Too low!", or correct. The information will be shown as text in the result span. If the guess is correct, you should show a message such as, "You got it right in 6 tries!" (You can still say "tries" even if it takes 1 guess.) Once the user guesses correctly, disable the button so that no more guesses can be made. The game will also show a history of all guesses made as a bulleted list. Each guess's number is added as a bullet to the end of the list. If the guess was too low, this bullet should use the `low` CSS class, which displays it in blue italic. If it's too high, the bullet should use the `high` CSS class, which is red and bold. If it is correct, it should be displayed without any class. You may assume that the text the user types in the field can be interpreted as an integer. You don't need to worry about the case where the user makes the same guess twice. You may assume that Prototype is also included in the page. The relevant HTML/CSS code for the page is the following

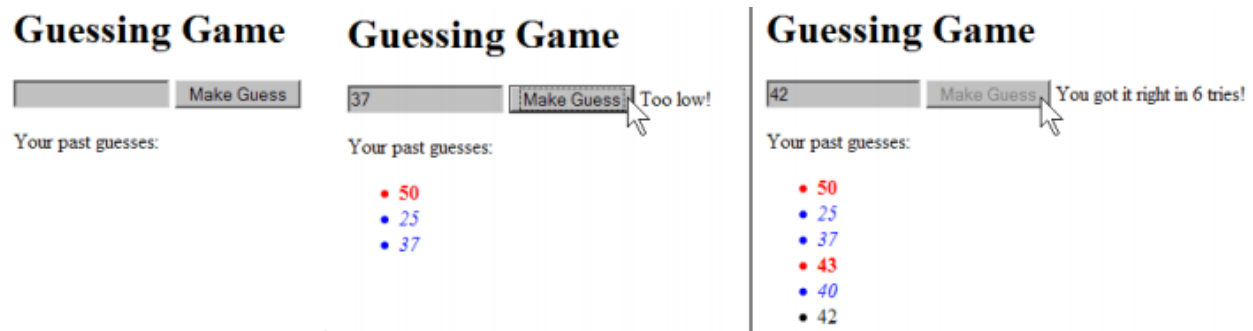
```
<h1>Guessing Game</h1>
<div>
  <input id="number" type="text" size="16" />
  <button id="makeguess">Make Guess</button>
  <span id="result"></span>
</div>

<p>Your past guesses:</p>
<ul id="guesses"></ul>
```

```
li.low {
  color: blue;
  font-style: italic;
}

li.high {
  color: red;
  font-weight: bold;
}
```

The following screenshots show the initial state and state after several guesses have been made.



24. Write the JavaScript code to add event behavior to the following web page. Assume that your code will be placed into a `.js` file that will be included in the page's header. The page contains an area with five images of baby Geneva, and a "Favorites" area. Your code should arrange it so that when one of these five images is clicked, that image will be moved to the rightmost end of the favorites area. Also, when an image is moved, a bullet is added to the end of the actions list indicating this. The bullet's text should be,

"Moved [image] to favorites." where [image] is equal to the src attribute of the image moved. (As you'll see in the output on the next page, even though the HTML's src attribute for each image is only a file name such as geneva1.jpg, when you access this information in your code, it will be a full URL. This is expected.) Nothing should happen when the user clicks on any other content on the page, such as text or other images such as the pacifier. Assume that the user won't click on an image that is already in the Favorites.

The following is the complete HTML code for the body of the page:



25. Write a JavaScript function that creates a table, accept row, column numbers from the user, and input row-column number as content (e.g. Row-0 Column-0) of a cell.

Sample HTML file :

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>Change the content of a cell</title>
<style type="text/css">
body {margin: 30px;}
</style>
</head><body>
<table id="myTable" border="1">
</table><form>
<input type="button" onclick="createTable()" value="Create the table">
</form></body></html>
```

26. Write a JavaScript program to remove items from a dropdown list.

Sample HTML file :

```
<!DOCTYPE html>
```



```

<html><head>
<meta charset=utf-8 />
<title>Remove items from a dropdown list</title>
</head><body><form>
<select id="colorSelect">
<option>Red</option>
<option>Green</option>
<option>White</option>
<option>Black</option>
</select>
<input type="button" onclick="removecolor()" value="Select and Remove">
</form></body></html>

```

27. Write a JavaScript program to count and display the items of a dropdown list, in an alert window.

Sample HTML file :

```

<!DOCTYPE html>
<html><head>
<meta charset=utf-8 />
<style type="text/css">
body {margin: 30px;}
</style>
<title>Count and display items of a dropdown list - w3resource</title>
</head><body><form>
Select your favorite Color :
<select id="mySelect">
<option>Red</option>
<option>Green</option>
<option>Blue</option>
<option>White</option>
</select>
<input type="button" onclick="getOptions()" value="Count and Output all items">
</form></body></html>

```

28. Write a JavaScript program to display a random image (clicking on a button) from the following list.

Sample Image information :

```

"http://farm4.staticflickr.com/3691/11268502654_f28f05966c_m.jpg", width: "240", height: "160"
"http://farm1.staticflickr.com/33/45336904_1aef569b30_n.jpg", width: "320", height: "195"
"http://farm6.staticflickr.com/5211/5384592886_80a512e2c9.jpg", width: "500", height: "343"

```

29. Background: The first step in brewing beer is to extract sugars from grains by soaking the grains in hot water. This step is called mashing, and the resulting sugar-water is called wort. Each variety of grain contributes a different amount of sugar (the grain's yield) to the wort. Calculating the final amount of sugar in the wort is an important part of designing a beer recipe, since the sugars will later ferment into alcohol. To measure the amount of sugar in the wort we use specific gravity (SG), a decimal number (e.g., 1.054) which represents the ratio of the sugar-water's density to that of pure water (=1.000 SG). Specific gravity measurements during the brewing process are typically between about 1.010 and 1.100. Your job is to write the JavaScript code to add behavior to the following page, which estimates the specific gravity of a list of user-entered grains:

Mash-It!

Malt bill:

lbs	Grain	PPG
8.5	American 2-Row Pale	38
.5	Cara-Pils/Dextrene	32
.5	Crystal 20	34
.5	Crystal 40	34

Estimated Specific Gravity: **1.068**
(assumes a 5.5 gallon batch)

HTML:

```
<h1>Mash-It!</h1>
<fieldset>
  <legend>Malt bill:</legend>

  <table id="malt_bill">
    <tr><th>lbs</th><th>Grain</th><th>PPG</th></tr>
    <tr>
      <td><input type="text" class="amount" /></td>
      <td><input type="text" class="name" /></td>
      <td><input type="text" class="yield" /></td>
    </tr>
    <!-- additional grain entry rows are added here -->
  </table>
</fieldset>


<div id="operations">
  <button id="add_grain">Add another grain</button>
  <button id="calculate">Calculate SG</button>
</div>

<div id="calculations">
  Estimated Specific Gravity:
  <span id="specific_gravity">
    <!-- your calculated SG goes here -->
  </span><br/>
  <span class="note">(assumes a 5.5 gallon batch)</span>
</div>
```


The page consists of a table in which the user can enter grains. Every time the Add another grain button is clicked, a new row should be added at the bottom of the table. Each row consists of three text-entry fields, each in its own table cell, for entering the following values: an amount (in pounds), a name, and a sugar yield (in “PPG”). These entry fields should have the classes amount, name, and yield respectively. When the Calculate SG button is pressed, your code should estimate the specific gravity of the user-entered grains by performing the following calculation: $SG_{est} = \text{round}([a_1y_1 + a_2y_2 + \dots + a_ny_n] / 5.5) / 1000 + 1$ (Where a_1 and y_1 represent the amount and yield of the first grain, and so on.) In other words, for each grain in the list, multiply its amount (the number in the “lbs” column) by its yield (the number in the “PPG” column), and take the sum of these products. Then divide that by 5.5 (which represents the volume of water it's dissolved in, in gallons). Round the result to the nearest integer, then to express it as a number 1 followed by 3 decimal places (e.g., 1.054), divide by 1000 and add 1. For example, using the values in the above screenshot, your calculation would be as follows: $SG_{est} = \text{round}([8.5 \cdot 38 + .5 \cdot 32 +$

$.5 \cdot 34 + .5 \cdot 34] / 5.5) / 1000 + 1 = 1.068$ You should assume all input to text boxes is valid; that is, reasonable values will always be entered in the fields for amount and yield.


These screenshots show various states of the program being used:



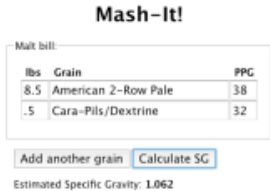
Initial state



After the first grain has been entered



After Add another grain has been clicked



After another grain has been entered, and Calculate SG has been clicked

30. **Background:** Distance runners need to pace themselves to avoid burning out before the end of a long race. It's sometimes helpful to print out and carry a card which lists exactly what time the runner should reach each mile marker if they are running at their target pace. That way, if the runner reaches a mile marker at a time substantially off from the one listed, they know they are going too fast (or too slow) and need to adjust their pace accordingly. Write the JavaScript code to add behavior to the following page, which creates a runner's pace chart based on entered values for distance and target pace:

Pace-It!

Calculate pace chart:

Distance 13.1 mi	Target pace 7:50 min/mi (mm:ss)
---------------------	---------------------------------------

Mile	Time
1	07:50
2	15:40
3	23:30
4	31:20
5	39:10
6	47:00
7	54:50
8	01:02:40
9	01:10:30
10	01:18:20
11	01:26:10
12	01:34:00
13	01:41:50
13.1	01:42:37

HTML:

```
<h1>Pace-It!</h1>

<fieldset>
  <legend>Calculate pace chart:</legend>

  <div class="distance">
    <label>Distance</label><br/>
    <input type="text" id="distance" /> mi
  </div>

  <div class="target">
    <label>Target pace</label><br/>
    <input type="text" id="target_pace" /> min/mi<br/>
    <span>(mm:ss)</span>
  </div>

  <div><button id="calculate">Calculate</button></div>
</fieldset>

<table id="pace_chart">
  <tr id="header_row"><th>Mile</th><th>Time</th></tr>
  <!-- calculated pace information will go here -->
</table>
```

The page consists of fields for entering a distance (in miles) and target pace (in minutes per mile), as well as a table for the pace chart. Initially the table contains only a header row with the column names, and no data; as such, you should hide the table as soon as the page is loaded. Then, when the user clicks the Calculate button, you should first populate the table with data, and then show it. When the chart has been generated, each row should consist of a mile number and a time at which the runner should reach that mile, each in its own table cell. Because doing arithmetic on times is very difficult, your code should make use of the following provided function that will multiply times for you:

```
// returns a 'mm:ss' or 'hh:mm:ss' time string representing
// the given time 't' multiplied by the given factor 'f'
function multiplyTime(t, f) {
  ...
}
```

Assume this function has been previously declared, and you are free to make use of it in your code. You do not need to write/implement this function, only call it. The last row in the table should always be for the exact value entered in the distance field, even if the distance is a non-integer number like 26.2. If the user clicks the Calculate button after a pace chart has previously been generated, all previous data should be cleared before the new data is added. When you do this, you must always retain the header row; it should never be removed or re-generated. You should assume all input to text boxes is valid; that is, reasonable values will always be entered in the fields for distance and pace.