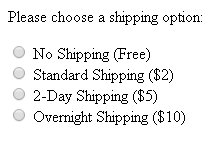
JavaScript Assignment #6

Grading Note: Grading will be based on whether your page does what it is supposed to do, but also that it is written properly with correct HTML 5, CSS, and JavaScript. If any of your assignment parts isn’t working right but you are turning it in anyway, be sure to indicate that it doesn’t work properly. I will discover this anyway, but you might save me a lot of time and keep me in a better grading mood!

**Part 1: Listen to the Radio (Buttons) and Get a Date!  
(Shipping Options for your Store and a Date for your receipt** – using A5P3 as a starting point)

Begin by making a copy of your **a5p3** file and call it **a6p1**. To your existing "store" add a group of radio buttons above your "Process Order" and "Display Quantity Pricing" buttons (just below your table of products and quantity inputs). Set up the following radio button options:



Those shipping prices are for the entire order regardless of the quantities involved. Add a new line to the receipt to reflect the shipping charge (even if they choose "no shipping"). The new line in the receipt should appear after the line for taxes because there are no taxes on the shipping charge.

Add the current date onto the end of your receipt, using the format 5/10/2019. Use the JavaScript "date" object so that your page will always show the current date. To see how the date object works, check out this demo script:   
[**http://www.faculty.rcc.edu/mcleod/js/class\_demos/js\_objects/dates.html**](http://www.faculty.rcc.edu/mcleod/js/class_demos/js_objects/dates.html)

**What to turn in:**

**Screen shot**: Into the given assignment 6 “turn it in” Word file's part 1 screen shot area, paste a copy of all of your web page displaying in the browser after putting your own name in for the name, purchase 1 item from your products, and choose one of the shipping options with a cost (do NOT choose "No Shipping") so that the added cost can be easily seen on your receipt total. (Note that it may not all fit on one screen shot. If that is the case, use 2 screen shots as needed.) The last line of the receipt should have the current date on it using the given format indicated above. Note that no sample screenshot is provided here since the changes are minimal.

**Code**: Into the given assignment 6 “turn it in” Word file, paste a copy of all of your web page’s code into the part 1 code section of the document. It should contain **ALL** of your page’s code. Do NOT paste a screenshot of code, paste the actual text of the code. **Make sure all of your code is visible** within the Word file's margins! You can format the font, spacing, tabbing, or size of the code after pasting into Word to improve its appearance and readability, just be sure not to actually change any of the code itself.

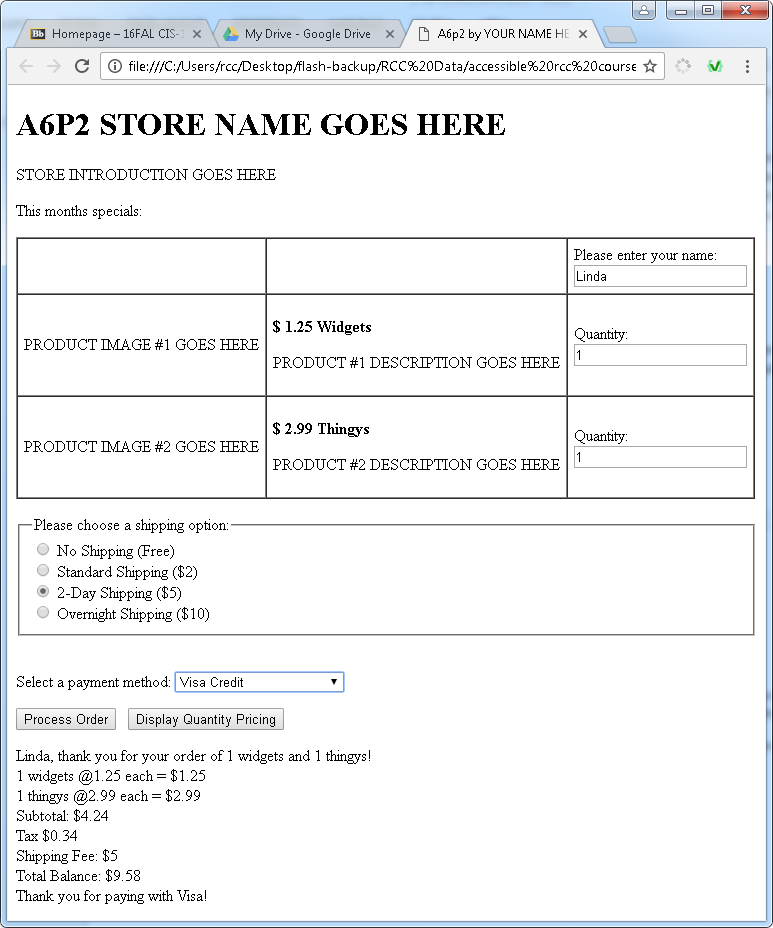
**Part 2: How are you ever going to pay for this?  
(Adding Payment Methods in a Select List, using A6P1 as a starting point)**

Begin by making a copy of your **a6p1** file and call it **a6p2**. To your existing "store" add a select list (drop down menu) of choices for the customer to identify how they are going to be paying for their order. The choices they should be choosing between are shown below:

* Visa Credit
* MasterCard Credit
* American Express Credit
* Debit Card
* Pay Pal

This select list should appear within your form just below your a6p1 shipping radio buttons and just above your "Process Order" and "Display Quantity Pricing" buttons. See the screen shot below for an example of what this form element should look like.

When the customer clicks the "Process Order" button, the customer's payment choice should appear on a new line of the receipt (just above the "date line" thanking them for paying with that particular method. See the screen shot below for an example of this new last line in the receipt. (Note that this screen shot does not include the date even though yours should.)



**What to turn in:**

**Screen shot**: Into the given assignment 6 “turn it in” Word file's part 2 screen shot area, paste a copy of all of your web page displaying in the browser after filling out the order form as shown above

(Note that it may not all fit on one screen shot. If that is the case, use 2 screen shots as needed.)

**Code**: Into the given assignment 6 “turn it in” Word file, paste a copy of all of your web page’s code into the part 2 code section of the document. It should contain **ALL** of your page’s code. Do NOT paste a screenshot of code, paste the actual text of the code. **Make sure all of your code is visible** within the Word file's margins! You can format the font, spacing, tabbing, or size of the code after pasting into Word to improve its appearance and readability, just be sure not to actually change any of the code itself.

**Part 3: Silly Strings!  
(Using String Methods and Properties to Process text box input)**

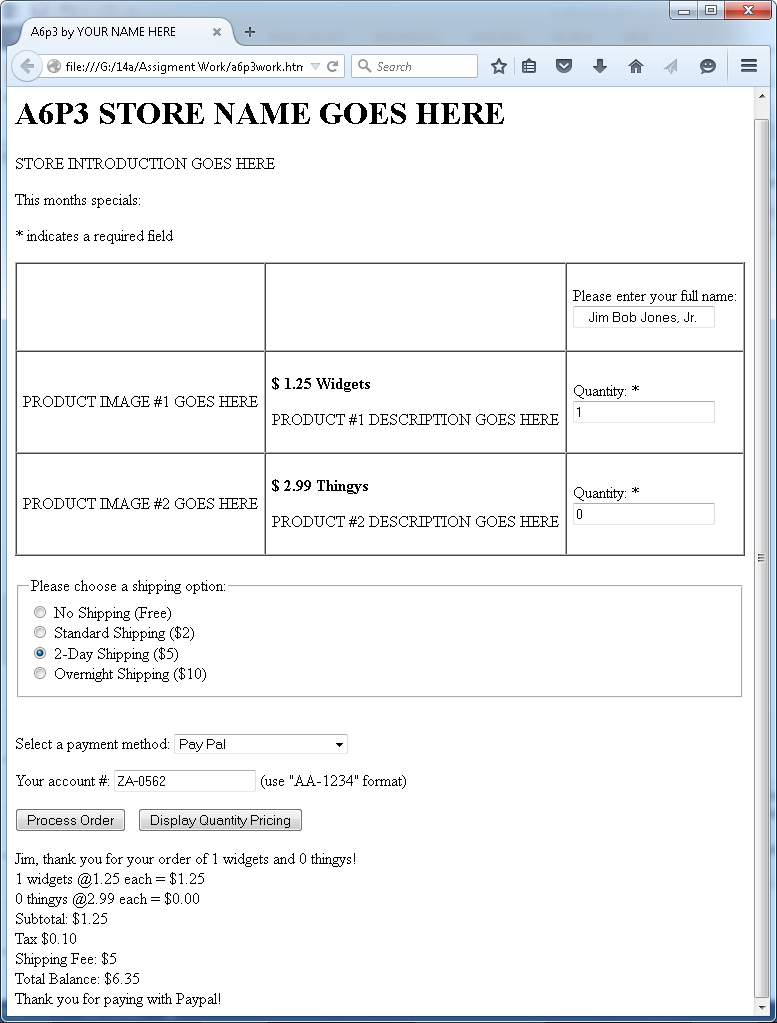
Begin by making a copy of your **a6p2** file and call it **a6p3**. To your existing "store", you will add 2 new features. These features revolve around the manipulation of strings (characters put together into one variable or expression). To do so, use JavaScript built-in string related methods and functions (and properties) such as length, toUpperCase, toLowerCase, trim, charAt, indexOf, substring, split, etc. Look up each of the string related tools mentioned here and be sure you are familiar with what they do so you can use the most appropriate tools that are available. That said, you may not use "regular expressions" to solve these 2 issues. We will explore those in part 4.

1) Add the feature of asking for a "full name" on the "name" textbox, but then when the name appears in the first line of the receipt area, it should only be the "first name" that shows up. Use string related functions, methods, and properties to do so. Also, any leading blanks entered before the first name should be ignored. See the screen shot below for an example of this. The full name entered is " Jim Bob Jones, Jr." but in the receipt the name appears as just "Jim".

2) add a new textbox to allow the customer to enter an "account number". This account number should be in the format of 2 letters followed by a dash and then 4 numbers. When the user clicks the "process order" button (or when the textbox is changed – your choice about what triggers the check), the account number text box should be validated to be in the correct format. If it is in the right format there is nothing you need to do except allow the ticket order to continue as usual. But if the account number is found to be invalid (not in the correct format), then you need to point this out somehow. In the sample screen shot (2nd screen shot) below, the text that displays the format is having its style changed if the text inside of it is not valid – the color is changed to red and the font-weight is chanced to bold. It should be changed back to black colored normal font-weight text if it is corrected and tried again. Examples of valid and invalid account numbers are shown below for clarity. The format "hint" should appear next to the textbox as shown in the 2nd screen shot below: Use "AA-1234" format.

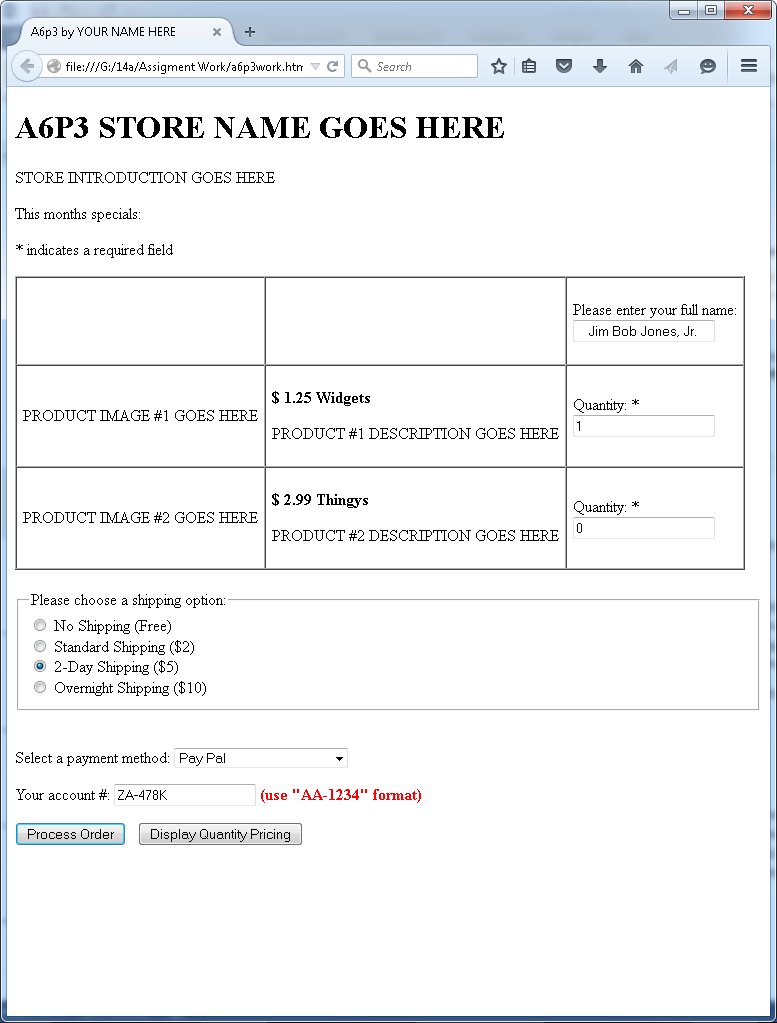
So "BB9459" is invalid because the dash is missing.  
"zz-092" is invalid because there are only 3 numbers after the dash instead of 4  
"za-0000 is valid because there are 2 alphabetic characters followed by a dash followed by 4 digit characters (numbers)

Keep in mind that you can compare one character string to another and it works in an alphabetic sense. By that it is meant that, for example, "a" < "b". This might help you to determine if a character is a letter or not, for example. Also keep in mind that a string can be considered an array of characters if that helps you.



Screen shot #1 (above) shows the form after typing in a full name (and a valid account #). In that case, your script should produce a proper receipt that includes the customer's first name only (regardless of the fact they entered their full name.) The current date should also show at the bottom of the receipt.

Screen shot #2 (below) shows the form after entering an invalid account number (last digit should not be a letter) with the text to the right of the textbox displaying as bold red text instead of its usual formatting (black and normal font-weight.) It was turned red and bold because this text is sitting within its own <span> tags and was given an "id" so that it can be accessed later as needed using "getElementById". Use that "span" object's "style" property. For example, if I wanted to change the alignment of the h1 heading that had an id of "storename": document.getElementById("storename").style.textAlign="center";



**What to turn in:**

**Screen shot:**

Into the given assignment 6 “turn it in” Word file's part 3 screen shot area, paste a copy of all of your web page displaying in the browser after filling out the order form as shown below with a bad customer # entered and hitting the "process order" button – your code should detect the bad customer # input and react appropriately and not produce a receipt but instead somehow indicate there is a problem with the customer number input.

(Note that it may not all fit on one screen shot. If that is the case, use 2 screen shots as needed.)

**Code**: Into the given assignment 6 “turn it in” Word file, paste a copy of all of your web page’s code into the part 3 code section of the document. It should contain **ALL** of your page’s code. Do NOT paste a screenshot of code, paste the actual text of the code. **Make sure all of your code is visible** within the Word file's margins! You can format the font, spacing, tabbing, or size of the code after pasting into Word to improve its appearance and readability, just be sure not to actually change any of the code itself.

**Part 4: Regular Expressions (OMG!) and Rounding Up For Charity!  
(Using Regular Expressions to test for String Patterns and an intro to the Math Object)**

Begin by making a copy of your **a6p3** file and call it **a6p4**. For this part, you are asked to do the first name and account number input matching using regular expressions instead of string methods. You are also asked to add one more line to the receipt to indicate how much the customer would pay if they wanted to "round up" for charity. These new requirements are explained further below, including demo scripts.

To "round up for charity" it means that you would donate the "change" necessary to raise your bill to the next even dollar (it is not actually rounding). So if your order comes to $25.47, for example, then you would donate 53 cents to charity and pay $26.00. In that case you would add the following to the end of the receipt (above the date): To round up for charity, donate 53 cents and pay $26.00. To make this calculation, use the JavaScript "Math" object. Here is a demo of the Math object:  
[**http://www.faculty.rcc.edu/mcleod/js/class\_demos/js\_objects/math.html**](http://www.faculty.rcc.edu/mcleod/js/class_demos/js_objects/math.html)

To become more familiar with regular expressions, you can check your textbook on pages 632-633. You can also google regular expressions and find lots of examples and explanations. Here are some other resources since our book doesn't do the subject justice:

This resource gives you more than you could ever want to know about regular expressions and it does so with good examples and explanations:

**http://eloquentjavascript.net/09\_regexp.html**

You can also create and test some regular expressions by using this script:

[**http://www.faculty.rcc.edu/mcleod/js/book\_demos/chapter\_09/script09-07.html**](http://www.faculty.rcc.edu/mcleod/js/book_demos/chapter_09/script09-07.html)

Here is another really cool tool for creating, testing, and reading descriptions of regular expressions that you write.

**https://regex101.com/**

Once you are familiar enough with regular expressions, you will use them to solve 2 issues in the a6p4 script. But before you try to write the regular expression code in your own script, first try them out in the above testing tools so you know if they work or not before getting into a problem in the browser with your code that might not be working right.

1) Rewrite your account number validation code from a6p3 (that used string functions) so that it validates the text box using a regular expression. Do so by writing a function named **isValidAccountNo( )** or **isInvalidAcctNo( )** that returns true or false depending on whether the text box input is valid. Inside this function, you can use the regular expression's "test" method, which itself returns a true/false based on whether or not the given string matches its regular expression pattern. Your new function can be called, for example, within the main function that processes orders. If the account number is valid, processing of the order continues. If the account number is not valid, then this input error should be marked in red on the form (as was done in a6p3 and showing in the screen shot above) and processing of the order stops.

2) Create another text box (below the "account number" text box) to ask for the user's "phone number". Expect this phone number to be in the following format: **123-456-7890.** Alternatively, if you prefer to accept phone numbers in the format **(123) 456-7890** instead (or even in addition to) the first format, you can do that. But whatever variation of the phone number you choose to accept, make it clear to the customer by giving an example of the format you expect right next to the text box (as was done for the account number text box). Then use regular expressions to validate that phone number in your script in the same manner as indicated above for the account number. That is, use a Boolean (true / false) function **isValidPhoneNo( )** or **isInvalidPhoneNo( )** that returns a true or false depending on the whether the given phone number in the text box matches the required format or not. Inside this function, you can use the regular expression's "test" method which itself returns a true/false based on whether or not the given string matches its regular expression pattern. Your new function can be called, for example, from within the main function that processes orders. If the phone number is valid, processing of the order continues. If the phone number is not valid, then this input error should be marked in red on the form (as was done for the account number text box in a6p3) and processing of the order stops.

**What to turn in:**

**Screen shot:**

Into the given assignment 6 “turn it in” Word file's part 4 screen shot area, paste a copy of all of your web page displaying in the browser after filling out the order form with a bad phone # entered and hitting the "process order" button – your code should detect the bad phone # input and react appropriately and not produce a receipt but instead indicate the error in red next to the customer phone number text box input.

(Note that it may not all fit on one screen shot. If that is the case, use 2 screen shots as needed.)

**Code**: Into the given assignment 6 “turn it in” Word file, paste a copy of all of your web page’s code into the part 4 code section of the document. It should contain **ALL** of your page’s code. Do NOT paste a screenshot of code, paste the actual text of the code. **Make sure all of your code is visible** within the Word file's margins! You can format the font, spacing, tabbing, or size of the code after pasting into Word to improve its appearance and readability, just be sure not to actually change any of the code itself.