

PIC 40A: Homework 2 (due 1/24 at 10pm)

Just for this homework on pure JavaScript, no submission to the PIC server is necessary! We will be creating some of the functions you will use for your merchandise page and login page.

In this assignment you will make one file called `hw2.js` and all you need to do is submit this file to Gradescope before the deadline. Start by opening `hw2.js` in `hw2.zip`.

1. Define a function called `sales_total` whose function comment reads as follows.

```
/**
Returns the sales total including a
7.25% sales tax on the total sale.

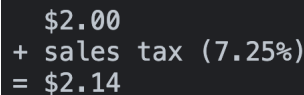
Note that sales tax is rounded normally with one exception...
Bankers rounding rounds the half-cent conditionally.
If the cent value of the tax total is odd, the 0.5 (half-cent) rounds upwards;
If the cent value is even, the half-cent rounds it downwards.
So $0.065 is rounded to $0.06 but $0.075 is rounded to $0.08.
Banker's rounding only takes place on the tax value.

@return {string} The receipt of items with tax.
*/
```

Here is an example of this function being called

```
prices = [2];
console.log(sales_total(prices));
```

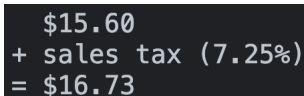
should print in Google Chrome console like:



```
$2.00
+ sales tax (7.25%)
= $2.14
```

```
prices = [1.20, 14.40];
console.log(sales_total(prices));
```

should print in Google Chrome console like:



```
$15.60
+ sales tax (7.25%)
= $16.73
```

Some comments:

- The functions should be defined as the function comments specify. Be mindful of what is returned from the function.
- Keep the total before tax in a very precise form. Banker's rounding will only occur on the tax value. Then you can do rounding to two decimal places when you compute the final total with tax.
- In addition to the functions introduced in class, you will also find this function useful: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toFixed Take care to consider limitations of floating point numbers.

For the next questions I need to make a definition.

I promise it is not as complicated as it looks!

I have not made this definition as general as possible because otherwise you would hate me:

<https://stackoverflow.com/questions/1969232/>.

Definition. A **cookie** is a special type of string.

It has the form "name1=value1; name2=value2; ...; lastName=lastValue".

The **names** and **values** can be any **non-empty** sequence of ASCII characters which are:

- alphanumeric characters: a, b, c, ..., z, A, B, C, ..., Z, 0, 1, 2, ..., 9, or
- a character appearing in the following string: " !#% '*+-. ^_` | ~".
- = is allowed in a **value**, but not a **name**.

In particular, the following characters are **not allowed**:

- white space
- , (commas)
- ; (semicolons)

For example, the following are examples of cookies:

- "first_name=Sarah; last_name=Burnett; username=burnett"
- "username=burnett; first_name=Sarah; last_name=Burnett"
- "_ga=GA1.2.34.56; dwf_sg_task_complete=False; lux_uid=888; _gid=GA1.2.88.88"
- "__stripe_mid=c4d6a-723-3ee-54d-640e5af; csrftoken=Kger31Gtcvyt%2F2ILWuQJoJ"

The following is **not** a cookie "name=Sarah Burnett; position=PIC;assistant,adjunct" because of the space between Sarah and Burnett, as well as the semicolon and comma in PIC; assistant,adjunct.

Finally, here are the actual questions...

2. Define a function called `extract_username` whose function comment reads as follows.

```
/**
This function extracts from a given cookie
the 'value' corresponding to the 'name' "username".

For example, both of the following function calls return "bur=nett":
. extract_username("first_name=Sarah; last_name=Burnett; username=bur=nett");
. extract_username("username=bur=nett; first_name=Sarah; last_name=Burnett");

If the given cookie has no 'name' called "username",
then the function returns the empty string.

For example, we have
. extract_username("common_error=Sara; " +
    "another_one=Burnet; another=Sarah_Brunette") == "";

@param {string} cookie : The cookie to extract information from.
@return {string} The 'value' corresponding to the 'name' "username";
    the empty string if "username" is not a 'name'.
*/
```

3. Define a function called `validate_username` whose function comment reads as follows.

```
/**
This function validates a username.
A string should be printed to the console after the username is evaluated.
It'll be determined acceptable or relevant username errors will be printed.

@param {string} username : The username to validate.
*/
```

Upon giving this function a username up to three things should happen.

- (a) The specified username should be checked to see if it is of a desired form:
- at least 5 characters,
 - at most 40 characters,
 - does not include spaces,
 - does not include commas,
 - does not include semicolons,
 - does not include =, or
 - does not include &;
 - each character is either an alphanumeric or contained in the following string:
!@#\$%^*()-_+[]{}: '|`~<.>/?

If the specified username is problematic one of two things will happen:

- If the specified username is problematic because it violates one of the first 7 bullet points, the user should be alerted in a useful manner through the console with the specific details. It shouldn't repeat the specifics in the message more than once. All issues should be listed. One example: if `,=` is the value assigned to `username`, the console log should say:

```
Username must be 5 characters or longer.  
Username cannot contain spaces.  
Username cannot contain commas.  
Username cannot contain =.
```

- If the specified username is problematic because it violates **only** the last bullet point (and none of the first 7), the user should be alerted with the following message:

```
Username can only use characters from the following string:  
abcdefghijklmnopqrstuvwxyz  
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
0123456789  
!@#$%^&*()-_+[]{}:;'|`~<.>/?
```

This message will not show if one or more of the first seven violations are triggered.

If the specified username is acceptable, the user should be alerted with the following message (with the username inserted):

```
The username Sarah is acceptable.
```

Grading

Here's how your work will be graded...

- For each question, the following offenses will lead to a 1-point deduction:
 - Omitting semi-colons.
 - Failing to declare a variable using either `'let'` or `'const'`.
 - Using `==` or `!=`.

Example: if you omit semi-colons in each question, you will score at most 17/20.

1. (6 pts) The grade will check that the calculations are correct for 6 test cases and the output is formatted correctly.
2. (6 pts) The grader will check 6 test cases. They will deduct at most 6 points.
Make sure to consider the existence of `"Nusername"` and `"usernameN"`.
Make sure that you return the empty string when you mean to. The empty string does not contain any characters.
3. (8 pts) The grader will check 8 test cases. 4 of them will be the same username from cookies in from Pb. 2.

Important: Delete all your test cases so that executing `hw2.js` does not cause anything to be printed to the console. Up to 2 points will be deducted.