

JavaScript Assignment 2

Question 1 [The Calculator]

- Write a function called `squareNumber` that will take one argument (a number), square that number, and return the result. It should also log a string like "The result of squaring the number 3 is 9."
- Write a function called `halfNumber` that will take one argument (a number), divide it by 2, and return the result. It should also log a string like "Half of 5 is 2.5."
- Write a function called `percentOf` that will take two numbers, figure out what percent the first number represents of the second number, and return the result. It should also log a string like "2 is 50% of 4."
- Write a function called `areaOfCircle` that will take one argument (the radius), calculate the area based on that, and return the result. It should also log a string like "The area for a circle with radius 2 is 12.566370614359172."
 - Bonus: Round the result so there are only two digits after the decimal.
- Write a function that will take one argument (a number) and perform the following operations, using the functions you wrote earlier1:
 1. Take half of the number and store the result.
 2. Square the result of #1 and store that result.
 3. Calculate the area of a circle with the result of #2 as the radius.
 4. Calculate what percentage that area is of the squared result (#3).

Question 2 [DrEvil]

Create a function called `DrEvil`. It should take a single argument, an amount, and return '<amount> dollars', except it will add '(pinky)' at the end if the amount is 1 million. For example:

```
DrEvil(10): 10 dollars
DrEvil(1000000): 1000000 dollars (pinky)
```

Question 3 [MixUp]

Create a function called `mixUp`. It should take in two strings, and return the concatenation of the two strings (separated by a space) slicing out and swapping the first 2 characters of each. You can assume that the strings are at least 2 characters long. For example:

```
mixUp('mix', 'pod'): 'pox mid'
mixUp('dog', 'dinner'): 'dig donner'
```

Look up the JavaScript string reference to find methods which may be useful!

Question 4 [FixStart]

Create a function called `fixStart`. It should take a single argument, a string, and return a version where all occurrences of its first character have been replaced with '*', except for the first character itself. You can assume that the string is at least one character long. For example:

```
fixStart('babble'): 'ba**le'
```

Question 5 [Verbing]

Create a function called `verbing`. It should take a single argument, a string. If its length is at least 3, it should add 'ing' to its end, unless it already ends in 'ing', in which case it should add 'ly' instead. If the string length is less than 3, it should leave it unchanged. For example:

```
verbing('swim'): 'swimming'  
verbing('swimming'): 'swimmingly'  
verbing('go'): 'go'
```

Question 6 [Not Bad]

- Create a function called `notBad` that takes a single argument, a string.
- It should find the first appearance of the substring 'not' and 'bad'.
- If the 'bad' follows the 'not', then it should replace the whole 'not...'bad' substring with 'good' and return the result.
- If it doesn't find 'not' and 'bad' in the right sequence (or at all), just return the original sentence.

For example:

```
notBad('This dinner is not that bad!'): 'This dinner is good!'  
notBad('This movie is not so bad!'): 'This movie is good!'  
notBad('This dinner is bad!'): 'This dinner is bad!'
```

Question 7 [Your Top Choices]

- Create an array to hold your top choices (colors, presidents, whatever).
- For each choice, log to the screen a string like: "My #1 choice is blue."
- **Bonus:** Change it to log "My 1st choice", "My 2nd choice", "My 3rd choice", picking the right suffix for the number based on what it is.

Question 8 [The Word Guesser]

You'll create a simple word guessing game where the user gets infinite tries to guess the word (like Hangman without the hangman, or like Wheel of Fortune without the wheel and fortune).

- Create two global arrays: one to hold the letters of the word (e.g. 'F', 'O', 'X'), and one to hold the current guessed letters (e.g. it would start with '_', '_', '_' and end with 'F', 'O', 'X').
- Write a function called `guessLetter` that will:
 - Take one argument, the guessed letter.
 - Iterate through the word letters and see if the guessed letter is in there.
 - If the guessed letter matches a word letter, changed the guessed letters array to reflect that.
 - When it's done iterating, it should log the current guessed letters ('F__')
 - and congratulate the user if they found a new letter.
 - It should also figure out if there are any more letters that need to be guessed,
 - and if not, it should congratulate the user for winning the game.
- Pretend you don't know the word, and call `guessLetter` multiple times with various letters to check that your program works.
- **Bonus:** Make it more like Wheel of Fortune:
 - Start with a reward amount of \$0
 - Every time a letter is guessed, generate a random amount and reward the user if they found a letter (multiplying the reward if multiple letters found), otherwise subtract from their reward.
 - When they guess the word, log their final reward amount.
- **Bonus:** Make it like Hangman:
 - Keep track of all the guessed letters (right and wrong) and only let the user guess a letter once. If they guess a letter twice, do nothing.
 - Keep track of the state of the hangman as a number (starting at 0), and subtract or add to that number every time they make a wrong guess.
 - Once the number reaches 6 (a reasonable number of body parts for a hangman), inform the user that they lost and show a hangman on the log.

Question 9 [The Recipe Card]

Never forget another recipe!

- Create an object to hold information on your favorite recipe. It should have properties for `title` (a string), `servings` (a number), and `ingredients` (an array of strings).
- On separate lines (one `console.log` statement for each), log the recipe information so it looks like:
 - Mole
 - Serves: 2
 - Ingredients:
 - cinnamon
 - cumin
 - cocoa

Question 10 [The Reading List]

Keep track of which books you read and which books you want to read!

- Create an array of objects, where each object describes a book and has properties for the `title` (a string), `author` (a string), and `alreadyRead` (a boolean indicating if you read it yet).

- Iterate through the array of books. For each book, log the book title and book author like so: "The Hobbit by J.R.R. Tolkien".
- Now use an if/else statement to change the output depending on whether you read it yet or not. If you read it, log a string like 'You already read "The Hobbit" by J.R.R. Tolkien', and if not, log a string like 'You still need to read "The Lord of the Rings" by J.R.R. Tolkien.'

Question 11 [The Movie Database]

It's like IMDB, but much much smaller!

- Create an object to store the following information about your favorite movie: **title** (a string), **duration** (a number), and **stars** (an array of strings).
- Create a function to print out the movie information like so: "Puff the Magic Dragon lasts for 30 minutes. Stars: Puff, Jackie, Living Sneezes."

Question 12 [The Cash Register]

- Write a function called **cashRegister** that takes a shopping cart object. The object contains item names and prices (itemName: itemPrice). The function should return the total price of the shopping cart.
- Example

```
// Input
var cartForParty = {
  banana: "1.25",
  handkerchief: ".99",
  Tshirt: "25.01",
  apple: "0.60",
  nalgene: "10.34",
  proteinShake: "22.36"
};

// Output
cashRegister(cartForParty); // 60.55
```

Question 13 [Credit Card Validation]

You're starting your own credit card business. You've come up with a new way to validate credit cards with a simple function called **validateCreditCard** that returns **true** or **false**.

Here are the rules for a valid number:

- Number must be 16 digits, all of them must be numbers
- You must have at least two different digits represented (all of the digits cannot be the same)
- The final digit must be even

- The sum of all the digits must be greater than 16

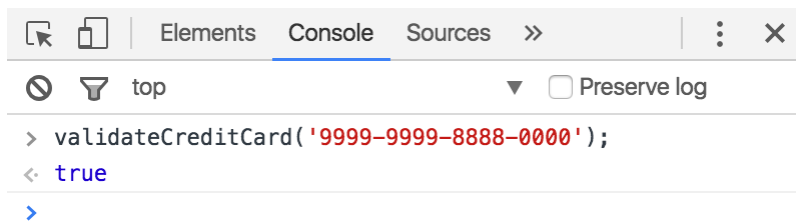
The following credit card numbers are valid:

- 9999-9999-8888-0000
- 6666-6666-6666-1666

The following credit card numbers are invalid:

- a923-3211-9c01-1112 *invalid characters*
- 4444-4444-4444-4444 *only one type of number*
- 1111-1111-1111-1110 *sum less than 16*
- 6666-6666-6666-6661 *odd final number*

In order to run the function, you'll need to load javascript on an HTML page. [Click here for instructions on how to do that](#). From there, you will open your developer console to call the function.



The screenshot shows a web browser's developer console with the 'Console' tab selected. The input field shows 'top' and there is a 'Preserve log' checkbox. The console log contains the following text:


```
> validateCreditCard('9999-9999-8888-0000');
< true
```

 A blue prompt character '>' is visible at the bottom of the console area.

Hint: Remove the dashed from the input string before checking if the input credit card number is valid.

Bonus: Return an object indicating whether the credit card is valid, and if not, what the error is

```
{ valid: true, number: 'a923-3211-9c01-1112' }
{ valid: false, number: 'a923-3211-9c01-1112', error: 'wrong_length' }
```

Double Bonus: Make your credit card scheme even more advanced! What are the rules, and what are some numbers that pass or fail? Ideas: check expiration date! Check out the [Luhn Algorithm](#) for inspiration.

Question 14 [Recursion]

We've seen that % (the remainder operator) can be used to test whether a number is even or odd by using % 2 to check whether it's divisible by two. Here's another way to define whether a positive whole number is even or odd:

- Zero is even.
- One is odd.
- For any other number N, its evenness is the same as N - 2. Define a recursive function isEven corresponding to this description. The function should accept a number parameter and return a Boolean. Test it on 50 and 75. See how it behaves on -1. Why? Can you think of a way to fix this?

Question 15 [Bean Counting]

You can get the Nth character, or letter, from a string by writing `"string".charAt(N)`, similar to how you get its length with `"s".length`. The returned value will be a string containing only one character (for example, `"b"`). The first character has position zero, which causes the last one to be found at position `string.length - 1`. In other words, a two-character string has length 2, and its characters have positions 0 and 1. Write a function `countBs` that takes a string as its only argument and returns a number that indicates how many upper case `"B"` characters are in the string. Next, write a function called `countChar` that behaves like `countBs`, except it takes a second argument that indicates the character that is to be counted (rather than counting only uppercase `"B"` characters). Rewrite `countBs` to make use of this new function.

Question 16 [Retry]

Say you have a function `primitiveMultiply` that, in 50 percent of cases, multiplies two numbers, and in the other 50 percent, raises an exception of type `MultiplicatorUnitFailure`. Write a function that wraps this clunky function and just keeps trying until a call succeeds, after which it returns the result. Make sure you handle only the exceptions you are trying to handle.

Question 17 [The Locked Box]

Consider the following (rather contrived) object:

```
var box = {
  locked : true ,
  unlock : function()
  {this.locked = false;},
  lock : function()
  {this.locked = true;},
  _content : [] ,
  get content()
  { if(this.locked)
    { throw new Error(" Locked !") ;
    }
    return this _content;
  }
};
```

It is a box with a lock. Inside is an array, but you can get at it only when the box is unlocked. Directly accessing the `_content` property is not allowed.

Write a function called `withBoxUnlocked` that takes a function value as argument, unlocks the box, runs the function, and then ensures that the box is locked again before returning, regardless of whether the argument function returned normally or threw an exception.